PCAN-UDS API

API Implementation of the UDS Standard
(ISO 14229-1:2006)

User Manual

Document version 1.4.1 (2019-11-19)

PEAK System

PEAK-System Technik GmbH
Otto-Röhm-Straße 69
64293 Darmstadt
Germany

Phone: +49 (0)6151 8173-20
Fax: +49 (0)6151 8173-29

www.peak-system.com
info@peak-system.com

Technical support:
E-mail: support@peak-system.com
Forum: www.peak-system.com/forum/

Document version 1.4.1 (2019-11-19)

Contents

1 PCAN-UDS API Documentation

Welcome to the documentation of PCAN-UDS API, a PEAK CAN API that implements ISO 14229-1:2006, UDS in CAN, an international standard that allows a diagnostic tester (client) to control diagnostic functions in an on-vehicle Electronic Control Unit (ECU or server).

In the following chapters you will find all the information needed to take advantage of this API.

2 Introduction

PCAN-UDS is a simple programming interface intended to support windows automotive applications that use PEAK-Hardware to communicate with Electronic Control Units (ECU) connected to the bus systems of a car, for maintenance purpose.

2.1 Understanding PCAN-UDS

UDS stands for Unified Diagnostic Services and is a communication protocol of the automotive industry. This protocol is described in the norm ISO 14229-1:2006.

The UDS protocol is the result of 3 other standardized diagnostic communication protocols:

- ISO 14230-3, as known as Keyword 2000 Protocol (KWP2000)
- ISO 14229-1:2006, as known as Diagnostic on CAN
- ISO 15765-2, as known as ISO-TP

The idea of this protocol is to contact all electronic data units installed and interconnected in a car, in order to provide maintenance, as checking for errors, actualizing of firmware, etc.

UDS is a Client/Server oriented protocol. In a UDS session (diagnostic session), a program application on a computer constitutes the client (within UDS, it is called Tester), the server is the ECU being *tested*, and the diagnostic requests from client to server are called services. The client always starts with a request and this ends with a positive or negative response from the server (ECU).

Since the transport protocol of UDS is done using ISO-TP, an international standard for sending data packets over a CAN Bus, the maximum data length that can be transmitted in a single data-block is 4095 bytes.

PCAN-UDS API is an implementation of the UDS on CAN standard. The physical communication is carried out by PCAN-Hardware (PCAN-USB, PCAN-PCI etc.) through the PCAN-ISO-TP and PCAN-Basic API (free CAN APIs from PEAK-System). Because of this it is necessary to have also the PCAN-ISO-TP and PCAN-Basic APIs (PCAN-ISO-TP.dll and PCANBasic.dll) present on the working computer where UDS is intended to be used. PCAN-UDS, PCAN-ISO-TP and PCAN-Basic APIs are free and available for all people that acquire a PCAN-Hardware.

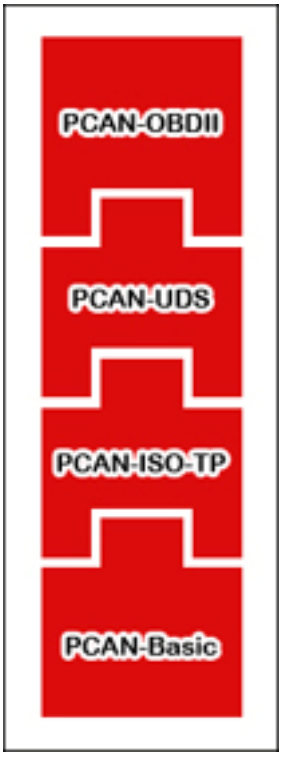


Figure 1: Relationship of the APIs.

2.2 Using PCAN-UDS

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

Initialization: In order to do UDS on CAN communication using a channel, it is necessary to initialize it first. This is done by making a call to the function UDS_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 24 functions starting with UDS_Svc **(class-methods:** starting with Svc) can be used to transmit UDS requests and the utility functions starting with UDS_WaitFor (**class-methods:** starting with WaitFor) are used to retrieve the results of a previous request. The UDS_Read and UDS_Write (**class-methods:** Read and Write) are lower level functions to read and write UDS messages from scratch. If desired, extra configuration can be made to improve a communication session, like service request timeouts or ISO-TP parameters.

Finalization: When the communication is finished, the function UDS_Uninitialize (**class-method:** Uninitialize) should be called in order to release the PUDS-Channel and the resources allocated for it. In this way the channel is marked as "Free" and can be used from other applications.

2.3 Features

- Implementation of the UDS protocol (14229-1:2006) for the communication with control units
- Windows DLLs for the development of 32-bit and 64-bit applications
- 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

2.4 System Requirements

- Windows 10, 8.1, 7 (32/64-bit)
- At least 512 MB RAM and 1 GHz CPU
- PC CAN interface from PEAK-System
- PCAN-Basic API
- PCAN-ISO-TP API

2.5 Scope of Supply

- Interface DLL, examples, and header files for all common programming languages
- Documentation in PDF format
- Documentation in HTML Help 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-UDS API.

3.1 Namespaces

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

Namespaces

	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.Light	Contains types and classes for using the PCAN-Light API from PEAK-System
{}	Peak.Can.Basic	Contains types and classes for using the PCAN-Basic API from PEAK-System
{}	Peak.Can.Ccp	Contains types and classes for using the CCP API implementation from PEAK-System
{}	Peak.Can.Xcp	Contains types and classes for using the XCP API implementation from PEAK-System
{}	Peak.Can.Iso.Tp	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-OBDII API implementation from PEAK-System
{}	Peak.Lin	Contains types and classes used to handle with LIN devices from PEAK-System
{}	Peak.RP1210A	Contains types and classes used to handle with CAN devices from PEAK-System through the TMC Recommended Practices 1210, version A, as known as RP1210(A)

3.1.1 Peak.Can.Uds

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

Remarks: Under the Delphi environment, these elements are enclosed in the PUDS-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).

Aliases

	Alias	Description
	TPUDSCANHandle	Represents a PCAN-UDS channel handle

Classes

	Class	Description
	UDSApi	Defines a class which represents the PCAN-UDS API

Structures

	Class	Description
	TPUDSMsg	Defines a CAN UDS message. The members of this structure are sequentially byte aligned
	TPUDSSessionInfo	Defines a diagnostic session information of a server
	TPUDSNetAddrInfo	Defines the Network Addressing Information of an UDS message

Enumerations

	Name	Description
	TPUDSBaudrate	Represents a PCAN Baud rate register value
	TPUDSHWType	Represents the type of PCAN hardware to be initialized
	TPUDSStatus	Represents a PCAN-UDS status/error code
	TPUDSResult	Represents the network status of a communicated PCAN-UDS message
	TPUDSParameter	Represents a PCAN-UDS parameter to be read or set
	TPUDSService	Represents a standardized UDS service
	TPUDSAddress	Represents a standardized ISO-15765-4 address
	TPUDSCanId	Represents a standardized ISO-15765-4 CAN identifier
	TPUDSProtocol	Represents a standardized and supported network communication protocol
	TPUDSAddressingType	Represents the type of message addressing of a PCAN-UDS message
	TPUDSMessageType	Represents the type of a PCAN-UDS message
	TPUDSSvcParamDSC	Represents the subfunction parameter for the UDS service DiagnosticSessionControl
	TPUDSSvcParamER	Represents the subfunction parameter for the UDS service ECUReset
	TPUDSSvcParamCC	Represents the subfunction parameter for the UDS service CommunicationControl
	TPUDSSvcParamTP	Represents the subfunction parameter for the UDS service TesterPresent
	TPUDSSvcParamCDTCS	Represents the subfunction parameter for the UDS service ControlDTCSetting
	TPUDSSvcParamROE	Represents the subfunction parameter for the UDS service ResponseOnEvent
	TPUDSSvcParamROERecommendedServiceID	Represents the recommended service to respond to for the UDS service ResponseOnEvent
	TPUDSSvcParamLC	Represents the subfunction parameter for the UDS service LinkControl
	TPUDSSvcParamLCBaudrateIdentifier	Represents the standard baudrate identifier for the UDS service LinkControl
	TPUDSSvcParamDI	Represents the data identifiers parameter for the UDS services like ReadDataByIdentifier
	TPUDSSvcParamRDBPI	Represents the subfunction parameter for the UDS service ReadDataByPeriodicIdentifier
	TPUDSSvcParamDDDI	Represents the subfunction parameter for the UDS service DynamicallyDefineDataIdentifier
	TPUDSSvcParamRDTCI	Represents the subfunction parameter for the UDS service ReadDTCInformation
	TPUDSSvcParamRDTCI_DTCSVM	Represents the DTC severity mask for the UDS service ReadDTCInformation
	TPUDSSvcParamIOCBI	Represents the subfunction parameter for the UDS service InputOutputControlByIdentifier
	TPUDSSvcParamRC	Represents the subfunction parameter for the UDS service RoutineControl
	TPUDSSvcParamRC_RID	Represents the routine identifier for the UDS service RoutineControl

3.2 Units

PEAK offers the implementation of some specific programming interfaces as Units for the Delphi's programming environment. The following units are available to be used:

Namespaces

	Alias	Description
{}	PUDS Unit	Delphi unit for using the PCAN-UDS API from PEAK-System

3.2.1 PUDS Unit

The PUDS-Unit contains types and classes to use the PCAN-UDS API within Delphi's programming environment and handle PCAN devices from PEAK-System.

Remarks: For the .NET Framework, these elements are enclosed in the Peak.Can.Uds namespace. The functionality of all elements included here is just the same. The difference between this Unit and the .NET namespace consists in the fact that Delphi accesses the Windows API directly (it is not Managed Code).

Aliases

	Alias	Description
	TPUDSCanHandle	Represents a PCAN-UDS channel handle

Classes

	Class	Description
	TUdsApi	Defines a class which represents the PCAN-ISO-TP API

Structures

	Class	Description
	TPUDSMsg	Defines a CAN UDS message. The members of this structure are sequentially byte aligned
	TPUDSSessionInfo	Defines a diagnostic session information of a serve
	TPUDSNetAddrInfo	Defines the Network Addressing Information of an UDS message

Enumerations

	Name	Description
	TPUDSBaudrate	Represents a PCAN Baud rate register value
	TPUDSHWType	Represents the type of PCAN hardware to be initialized
	TPUDSStatus	Represents a PCAN-UDS status/error code
	TPUDSResult	Represents the network status of a communicated PCAN-UDS message
	TPUDSParameter	Represents a PCAN-UDS parameter to be read or set
	TPUDSService	Represents a standardized UDS service
	TPUDSAddress	Represents a standardized ISO-15765-4 address
	TPUDSCanId	Represents a standardized ISO-15765-4 CAN identifier
	TPUDSProtocol	Represents a standardized and supported network communication protocol
	TPUDSAddressingType	Represents the type of message addressing of a PCAN-UDS message
	TPUDSMessageType	Represents the type of a PCAN-UDS message
	TPUDSSvcParamDSC	Represents the subfunction parameter for the UDS service DiagnosticSessionControl

	Name	Description
	TPUDSSvcParamER	Represents the subfunction parameter for the UDS service ECUReset
	TPUDSSvcParamCC	Represents the subfunction parameter for the UDS service CommunicationControl
	TPUDSSvcParamTP	Represents the subfunction parameter for the UDS service TesterPresent
	TPUDSSvcParamCDTCS	Represents the subfunction parameter for the UDS service ControlDTCSetting
	TPUDSSvcParamROE	Represents the subfunction parameter for the UDS service ResponseOnEvent
	TPUDSSvcParamROERecommendedServiceID	Represents the recommended service to respond to for the UDS service ResponseOnEvent
	TPUDSSvcParamLC	Represents the subfunction parameter for the UDS service LinkControl
	TPUDSSvcParamLCBaudrateIdentifier	Represents the standard baudrate identifier for the UDS service LinkControl
	TPUDSSvcParamDI	Represents the data identifiers parameter for the UDS services like ReadDataByIdentifier
	TPUDSSvcParamRDBPI	Represents the subfunction parameter for the UDS service ReadDataByPeriodicIdentifier
	TPUDSSvcParamDDDI	Represents the subfunction parameter for the UDS service DynamicallyDefineDataIdentifier
	TPUDSSvcParamRDTCI	Represents the subfunction parameter for the UDS service ReadDTCInformation
	TPUDSSvcParamRDTCI_DTCSVM	Represents the DTC severity mask for the UDS service ReadDTCInformation
	TPUDSSvcParamIOCBI	Represents the subfunction parameter for the UDS service InputOutputControlByIdentifier
	TPUDSSvcParamRC	Represents the subfunction parameter for the UDS service RoutineControl
	TPUDSSvcParamRC_RID	Represents the routine identifier for the UDS service RoutineControl

3.3 Classes

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

Classes

	Class	Description
	UDSApi	Defines a class to use the PCAN-UDS API within the Microsoft's .NET Framework programming environment
	TUdsApi	Defines a class to use the PCAN-UDS API within the Delphi programming environment

3.3.1 UDSApi

Defines a class which represents the PCAN-UDS API to be used within the Microsoft's .NET Framework.

Syntax

C#

```
public static class UDSApi
```

C++ / CLR

```
public ref class UDSApi abstract sealed
```

Visual Basic

```
Public NotInheritable Class UDSApi
```

Remarks: The UDSApi class collects and implements the PCAN-UDS API functions. Each method is called just like the API function with the exception that the prefix "UDS_" is not used. The structure and functionality of the methods and API functions are the same.

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

```
TPUDSStatus res;
// Static use, without any instance
//
res = UDSApi.Initialize(UDSApi.PUDS_USBBUS1, TPUDSBaudrate.PUDS_BAUD_500K);
```

Note: This class under Delphi is called TUdsApi.

See also: Methods on page 74, Definitions on page 330.

3.3.2 TUDSApi

Defines a class which represents the PCAN-UDS API to be used within the Delphi programming environment.

Syntax

Pascal OO

```
TUdsApi = class
```

Remarks: TUdsApi is a class containing only class-methods and constant members, allowing their use without the creation of any object, just like a static class of another programming languages. It collects and implements the PCAN-UDS API functions. Each method is called just like the API function with the exception that the prefix "UDS_" is not used. The structure and functionality of the methods and API functions are the same.

Note: This class under .NET framework is called UDSApi.

See also: Methods on page 74, Definitions on page 330.

3.4 Structures

The PCAN-UDS API defines the following structures:

Name	Description
TPUDSMsg	Defines a CAN UDS message. The members of this structure are sequentially byte aligned
TPUDSSessionInfo	Defines a diagnostic session information of a server.
TPUDSNetAddrInfo	Defines the Network Addressing Information of a UDS message.

3.4.1 TPUDSMsg

Defines a CAN UDS message.

Syntax

C++

```
#pragma pack(push, 8)
typedef struct tagTPUDSMsg
{
      TPUDSNetAddrInfo NETADDRINFO;
      BYTE RESULT;
      BYTE NO_POSITIVE_RESPONSE_MSG;
      WORD LEN;
      TPUDSMessageType MSGTYPE;
      union {
        BYTE RAW[PUDS_MAX_DATA];
        struct tagREQUEST
        {
            BYTE SI;
            BYTE PARAM[PUDS_MAX_DATA-1];
        } REQUEST;
        struct tagPOSITIVE
        {
            BYTE SI;
            BYTE PARAM[PUDS_MAX_DATA-1];
        } POSITIVE;
        struct tagNEGATIVE
        {
            BYTE NR_SI;
            BYTE SI;
            BYTE NRC;
        } NEGATIVE;
      } DATA;
} TPUDSMsg;
```

Pascal OO

```
{$A8}
TPUDSMsg = record
        NETADDRINFO: TPUDSNetAddrInfo;
        RESULT: TPUDSResult;
        NO_POSITIVE_RESPONSE_MSG: Byte;
        LEN: Word;
        MSGTYPE: TPUDSMessageType;
        DATA: array[0..4095] of Byte;
end;
PTPUDSMsg = ^TPUDSMsg;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct TPUDSMsg
{
    public TPUDSNetAddrInfo NETADDRINFO;
    [MarshalAs(UnmanagedType.U1)]
    public TPUDSResult RESULT;
    public byte NO_POSITIVE_RESPONSE_MSG;
    public ushort LEN;
    [MarshalAs(UnmanagedType.U1)]
    public TPUDSMessageType MSGTYPE;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4095)]
```

```
    public byte[] DATA;

    public bool IsPositiveResponse
    {
        get
        {
            if (DATA != null)
                return (DATA[0] & 0x40) == 0x40;
            return false;
        }
    }
    public bool IsNegativeResponse
    {
        get
        {
            if (DATA != null)
                return DATA[0] == 0x7F;
            return false;
        }
    }
    public byte ServiceID
    {
        get
        {
            if (DATA != null)
                return IsNegativeResponse ? DATA[1] : DATA[0];
            return 0;
        }
    }
}
```

C++ / CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct TPUDSMsg
{
      TPUDSNetAddrInfo NETADDRINFO;
      [MarshalAs(UnmanagedType::U1)]
      TPUDSResult RESULT;

      Byte NO_POSITIVE_RESPONSE_MSG;
      unsigned short LEN;
      [MarshalAs(UnmanagedType::U1)]
      TPUDSMessageType MSGTYPE;
      [MarshalAs(UnmanagedType::ByValArray, SizeConst = 4095)]
      array<Byte>^ DATA;

      property bool IsPositiveResponse
      {
            bool get()
            {
                  if (DATA != nullptr)
                        return (DATA[0] & 0x40) == 0x40;
                  return false;
            }
      }
      property bool IsNegativeResponse
      {
            bool get()
            {
                  if (DATA != nullptr)
```

```
                        return DATA[0] == 0x7F;
                return false;
            }
        }
        property Byte ServiceID
        {
            Byte get()
            {
                if (DATA != nullptr)
                    return IsNegativeResponse ? DATA[1] : DATA[0];
                return 0;
            }
        }
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure TPUDSMsg
    Public NETADDRINFO As TPUDSNetAddrInfo
    <MarshalAs(UnmanagedType.U1)> _
    Public RESULT As TPUDSResult
    Public NO_POSITIVE_RESPONSE_MSG As Byte
    Public LEN As UShort
    <MarshalAs(UnmanagedType.U1)> _
    Public MSGTYPE As TPUDSMessageType
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4095)> _
    Public DATA As Byte()

    Public ReadOnly Property IsPositiveResponse() As Boolean
        Get
            If (DATA Is Not Nothing) Then
                Return (DATA(0) And &H40) = &H40
            End If
            Return False
        End Get
    End Property

    Public ReadOnly Property IsNegativeResponse() As Boolean
        Get
            If (DATA Is Not Nothing) Then
                Return DATA(0) = &H7F
            End If
            Return False
        End Get
    End Property

    Public ReadOnly Property ServiceID() As Byte
        Get
            If (DATA Is Not Nothing) Then
                If IsNegativeResponse Then
                    Return DATA(1)
                Else
                    Return DATA(0)
                End If
            End If
            Return False
        End Get
    End Property
End Structure
```

Fields

Name	Description
NETADDRINFO	Network Addressing Information
RESULT	Result status of the network communication (see TPUDSResult)
NO_POSITIVE_RESPONSE_MSG	States wether Positive Response Message should be ignored upon reception (see also TPUDSMsg Member Value Definitions)
LEN	Data Length of the message.
MSGTYPE	Type of UDS Message (see TPUDSMessageType)
DATA	Represents the buffer containing the data of this message

Remarks: .NET framework provides three properties to ease UDS message data queries:

- IsPositiveResponse: states whether the response is positive or negative,
- IsNegativeResponse: states whether the response is positive or negative,
- ServiceID: returns the service identifier of the request

C++ API defines DATA as a union containing the following fields:

- RAW: represents the buffer containing the data of this message,
- REQUEST: represents the data as a message request which contains a service ID and data,
- POSITIVE: represents the data as a positive response which contains a service ID and data,
- NEGATIVE: represents the data as a negative response which contains the negative response service identifier, the requested service ID and the negative response code

See also: TPUDSMsg Member Value Definitions on page 333, TPUDSMessageType on page 50, TPUDSResult on page 30.

3.4.2 TPUDSSessionInfo

Defines a UDS Session Information.

Syntax

C++

```
#pragma pack(push, 8)
typedef struct tagTPUDSSessionInfo
{
      TPUDSNetAddrInfo NETADDRINFO;
      BYTE SESSION_TYPE;
      WORD TIMEOUT_P2CAN_SERVER_MAX;
      WORD TIMEOUT_ENHANCED_P2CAN_SERVER_MAX;
} TPUDSSessionInfo;
```

Pascal OO

```
{$A8}
TPUDSSessionInfo = record
    NETADDRINFO: TPUDSNetAddrInfo;
    SESSION_TYPE: Byte;
    TIMEOUT_P2CAN_SERVER_MAX: Word;
    TIMEOUT_ENHANCED_P2CAN_SERVER_MAX: Word;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct TPUDSSessionInfo
{
    public TPUDSNetAddrInfo NETADDRINFO;
    public byte SESSION_TYPE;
    public ushort TIMEOUT_P2CAN_SERVER_MAX;
    public ushort TIMEOUT_ENHANCED_P2CAN_SERVER_MAX;
}
```

C++ / CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct TPUDSSessionInfo
{
      TPUDSNetAddrInfo NETADDRINFO;
      Byte SESSION_TYPE;
      unsigned short TIMEOUT_P2CAN_SERVER_MAX;
      unsigned short TIMEOUT_ENHANCED_P2CAN_SERVER_MAX;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure TPUDSSessionInfo
    Public NETADDRINFO As TPUDSNetAddrInfo
    Public SESSION_TYPE As Byte
    Public TIMEOUT_P2CAN_SERVER_MAX As UShort
    Public TIMEOUT_ENHANCED_P2CAN_SERVER_MAX As UShort
End Structure
```

Fields

Name	Description
NETADDRINFO	Network Addressing Information (see TPUDSNetAddrInfo)
RESULT	Currently activated Diagnostic Session
TIMEOUT_P2CAN_SERVER_MAX	Default P2Can_Server_Max timing for the activated session (i.e. maximum time allowed for an ECU to transmit a response indication)
TIMEOUT_ENHANCED_P2CAN_SERVER_MAX	Enhanced P2Can_Server_Max timing for the activated session (i.e. maximum time allowed for an ECU to transmit a response indication after having sent a negative response code stating that more time is required: PUDS_NRC_EXTENDED_TIMING, 0x78)

See also: TPUDSNetAddrInfo below, UDS_SvcDiagnosticSessionControl on page 278 (**class-method:** SvcDiagnosticSession Control).

3.4.3 TPUDSNetAddrInfo

Defines the Network Address Information of a UDS message.

Syntax

C++

```
#pragma pack(push, 8)
typedef struct tagTPUDSNetAddrInfo
{
      BYTE SA;
```

```
      BYTE TA;
      TPUDSAddressingType TA_TYPE;
      BYTE RA;
      TPUDSProtocol PROTOCOL;
} TPUDSNetAddrInfo;
#pragma pack(pop)
```

Pascal OO

```
{$A8}
TPUDSNetAddrInfo = record
    SA: Byte;
    TA: Byte;
    TA_TYPE: TPUDSAddressingType;
    RA: Byte;
    PROTOCOL: TPUDSProtocol;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct TPUDSNetAddrInfo
{
    public byte SA;
    public byte TA;
    [MarshalAs(UnmanagedType.U1)]
    public TPUDSAddressingType TA_TYPE;
    public byte RA;
    [MarshalAs(UnmanagedType.U1)]
    public TPUDSProtocol PROTOCOL;
}
```

C++ / CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct TPUDSNetAddrInfo
{
    Byte SA;
    Byte TA;
    [MarshalAs(UnmanagedType::U1)]
    TPUDSAddressingType TA_TYPE;
    Byte RA;
    [MarshalAs(UnmanagedType::U1)]
    TPUDSProtocol PROTOCOL;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure TPUDSNetAddrInfo
    Public SA As Byte
    Public TA As Byte
    <MarshalAs(UnmanagedType.U1)> _
    Public TA_TYPE As TPUDSAddressingType
    Public RA As Byte
    <MarshalAs(UnmanagedType.U1)> _
    Public PROTOCOL As TPUDSProtocol
End Structure
```

Fields

Name	Description
SA	Represents the Source Address (see TPUDSAddress)
TA	Represents the Target Address (see TPUDSAddress)
TA_TYPE	Represents the Target Address Type (see TPUDSAddressingType)
RA	Represents the Remote Address (see TPUDSAddress)
PROTOCOL	Represents the protocol being used for communication (see TPUDSProtocol)
SA	Represents the Source Address (see TPUDSAddress)

See also: TPUDSAddress on page 43, TPUDSAddressingType on page 49, TPUDSProtocol on page 47.

3.5 Types

The PCAN-ISO-TP API defines the following types:

Name	Description
TPUDSCANHandle	Represents a PCAN-UDS channel handle
TPUDSBaudrate	Represents a PCAN Baud rate register value
TPUDSHWType	Represents the type of PCAN hardware to be initialized
TPUDSStatus	Represents a PCAN-UDS status/error code
TPUDSResult	Represents the network status of a communicated PCAN-UDS message
TPUDSParameter	Represents a PCAN-UDS parameter to be read or set
TPUDSService	Represents a standardized UDS service
TPUDSAddress	Represents a standardized ISO-15765-4 address
TPUDSCanId	Represents a standardized ISO-15765-4 CAN identifier
TPUDSProtocol	Represents a standardized and supported network communication protocol
TPUDSAddressingType	Represents the type of message addressing of a PCAN-UDS message
TPUDSMessageType	Represents the type of a PCAN-UDS message
TPUDSSvcParamDSC	Represents the subfunction parameter for the UDS service DiagnosticSessionControl
TPUDSSvcParamER	Represents the subfunction parameter for the UDS service ECUReset
TPUDSSvcParamCC	Represents the subfunction parameter for the UDS service CommunicationControl
TPUDSSvcParamTP	Represents the subfunction parameter for the UDS service TesterPresent
TPUDSSvcParamCDTCS	Represents the subfunction parameter for the UDS service ControlDTCSetting
TPUDSSvcParamROE	Represents the subfunction parameter for the UDS service ResponseOnEvent
TPUDSSvcParamROERecommendedServiceID	Represents the recommended service to respond to for the UDS service ResponseOnEvent
TPUDSSvcParamLC	Represents the subfunction parameter for the UDS service LinkControl
TPUDSSvcParamLCBaudrateIdentifier	Represents the standard baudrate identifier for the UDS service LinkControl
TPUDSSvcParamDI	Represents the data identifiers parameter for the UDS services like ReadDataByIdentifier
TPUDSSvcParamRDBPI	Represents the subfunction parameter for the UDS service ReadDataByPeriodicIdentifier

Name	Description
TPUDSSvcParamDDDI	Represents the subfunction parameter for the UDS service DynamicallyDefineDataIdentifier
TPUDSSvcParamRDTCI	Represents the subfunction parameter for the UDS service ReadDTCInformation
TPUDSSvcParamRDTCI_DTCSVM	Represents the DTC severity mask for the UDS service ReadDTCInformation
TPUDSSvcParamIOCBI	Represents the subfunction parameter for the UDS service InputOutputControlByIdentifier
TPUDSSvcParamRC	Represents the subfunction parameter for the UDS service RoutineControl
TPUDSSvcParamRC_RID	Represents the routine identifier for the UDS service RoutineControl

3.5.1 TPUDSCANHandle

Represents a PCAN-UDS channel handle.

Syntax

C++ Syntax

```
#define TPUDSCANHandle WORD
```

C++ / CLR

```
#define TPUDSCANHandle System::Int16
```

C# Syntax

```
using TPUDSCANHandle = System.Int16;
```

Visual Basic Syntax

```
Imports TPUDSCANHandle = System.Int16
```

Remarks: TPUDSCANHandle is defined for the PCAN-UDS API but it is identical to a TPCANTPCANHandle from PCAN-ISO-TP API or TPCANHandle from PCAN-Basic API.

.NET Framework programming languages:

An alias is used to represent a Channel handle under Microsoft .NET in order to originate an homogeneity between all programming languages listed above.

Aliases are defined in the Peak.Can.Uds Namespace for C# and VB .NET. 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.Uds Namespace. Otherwise, just use the native type, which in this case is a Byte.

C#

```
using System;
using Peak.Can.UDS;
using TPUDSCANHandle = System.Int16; // Alias's declaration for System.Byte
```

Visual Basic

```
Imports System
Imports Peak.Can.Uds
Imports TPUDSCANHandle = System.Int16 'Alias' declaration for System.Byte
```

See also: PCAN-UDS Handle Definitions on page 330.

3.5.2 TPUDSStatus

Represents a PUDS status/error code. According with the programming language, this type can be a group of defined values or an enumeration.

Syntax

C++ Syntax

```
#define PUDS_ERROR_OK 0x00000
#define PUDS_ERROR_NOT_INITIALIZED 0x00001
#define PUDS_ERROR_ALREADY_INITIALIZED 0x00002
#define PUDS_ERROR_NO_MEMORY     0x00003
#define PUDS_ERROR_OVERFLOW 0x00004
#define PUDS_ERROR_TIMEOUT 0x00006
#define PUDS_ERROR_NO_MESSAGE 0x00007
#define PUDS_ERROR_WRONG_PARAM 0x00008
#define PUDS_ERROR_BUSLIGHT 0x00009
#define PUDS_ERROR_BUSHEAVY 0x0000A
#define PUDS_ERROR_BUSOFF 0x0000B
#define PUDS_ERROR_CAN_ERROR 0x80000000
```

C++ / CLR

```
public enum TPUDSStatus : unsigned int
{
        PUDS_ERROR_OK = 0x00000,
        PUDS_ERROR_NOT_INITIALIZED = 0x00001,
        PUDS_ERROR_ALREADY_INITIALIZED = 0x00002,
        PUDS_ERROR_NO_MEMORY = 0x00003,
        PUDS_ERROR_OVERFLOW = 0x00004,
        PUDS_ERROR_TIMEOUT = 0x00006,
        PUDS_ERROR_NO_MESSAGE = 0x00007,
        PUDS_ERROR_WRONG_PARAM = 0x00008,
        PUDS_ERROR_BUSLIGHT = 0x00009,
        PUDS_ERROR_BUSHEAVY = 0x0000A,
        PUDS_ERROR_BUSOFF = 0x0000B,
        PUDS_ERROR_CAN_ERROR = 0x80000000,
};
```

C# Snytax

```
public enum TPUDSStatus : uint
{
        PUDS_ERROR_OK = 0x00000,
```

```
        PUDS_ERROR_NOT_INITIALIZED = 0x00001,
        PUDS_ERROR_ALREADY_INITIALIZED = 0x00002,
        PUDS_ERROR_NO_MEMORY = 0x00003,
        PUDS_ERROR_OVERFLOW = 0x00004,
        PUDS_ERROR_TIMEOUT = 0x00006,
        PUDS_ERROR_NO_MESSAGE = 0x00007,
        PUDS_ERROR_WRONG_PARAM = 0x00008,
        PUDS_ERROR_BUSLIGHT = 0x00009,
        PUDS_ERROR_BUSHEAVY = 0x0000A,
        PUDS_ERROR_BUSOFF = 0x0000B,
        PUDS_ERROR_CAN_ERROR = 0x80000000,
};
```

Pascal OO

```
TPUDSStatus = (
        PUDS_ERROR_OK = $00000
        PUDS_ERROR_NOT_INITIALIZED = $00001
        PUDS_ERROR_ALREADY_INITIALIZED = $00002
        PUDS_ERROR_NO_MEMORY = $00003
        PUDS_ERROR_OVERFLOW = $00004
        PUDS_ERROR_TIMEOUT = $00006
        PUDS_ERROR_NO_MESSAGE = $00007
        PUDS_ERROR_WRONG_PARAM = $00008
        PUDS_ERROR_BUSLIGHT = $00009
        PUDS_ERROR_BUSHEAVY = $0000A
        PUDS_ERROR_BUSOFF = $0000B
        PUDS_ERROR_CAN_ERROR = LongWord($80000000)
);
```

Visual Basic Syntax

```
Public Enum TPUDSStatus As Integer
        PUDS_ERROR_OK = &H0
        PUDS_ERROR_NOT_INITIALIZED = &H1
        PUDS_ERROR_ALREADY_INITIALIZED = &H2
        PUDS_ERROR_NO_MEMORY = &H3
        PUDS_ERROR_OVERFLOW = &H4
        PUDS_ERROR_TIMEOUT = &H6
        PUDS_ERROR_NO_MESSAGE = &H7
        PUDS_ERROR_WRONG_PARAM = &H8
        PUDS_ERROR_BUSLIGHT = &H9
        PUDS_ERROR_BUSHEAVY = &HA
        PUDS_ERROR_BUSOFF = &HB
        PUDS_ERROR_CAN_ERROR = &H80000000
End Enum
```

Remarks: The PUDS_ERROR_CAN_ERROR status is a generic error code that is used to identify PCAN-Basic errors (as PCAN-Basic API is used internally by the PCAN-UDS API). When a PCAN-Basic error occurs, the API performs a bitwise combination of the PUDS_ERROR_CAN_ERROR and the PCAN-Basic (TPCANStatus) error.

WaitForService function may return exactly PUDS_ERROR_CAN_ERROR, in this specific case the PCAN-Basic status states no error meaning that the error occurred in the ISO-TP layer. The network status result of the UDS message will provide extra information.

Values

Name	Value	Description
PUDS_ERROR_OK	0x00000 (000000)	No error. Success
PUDS_ERROR_NOT_INITIALIZED	0x00001 (000001)	Not initialized
PUDS_ERROR_ALREADY_INITIALIZED	0x00002 (000002)	Already initialized
PUDS_ERROR_NO_MEMORY	0x00003 (000003)	Failed to allocate memory
PUDS_ERROR_OVERFLOW	0x00004 (000004)	Buffer overflow occurred (too many channels initialized or too many messages in queue)
PUDS_ERROR_TIMEOUT	0x00006 (000006)	Timeout while trying to access the PCAN-UDS API
PUDS_ERROR_NO_MESSAGE	0x00007 (000007)	No message available
PUDS_ERROR_WRONG_PARAM	0x00008 (000008)	Invalid parameter
PUDS_ERROR_BUSLIGHT	0x00009 (000009)	Bus error: an error counter reached the 'light' limit
PUDS_ERROR_BUSHEAVY	0x0000A (000010)	Bus error: an error counter reached the 'heavy' limit
PUDS_ERROR_BUSOFF	0x0000B (000011)	Bus error: the CAN controller is in bus-off state
PUDS_ERROR_CAN_ERROR	0x80000000 (2147483648)	PCAN-Basic error flag (remove the flag to get a TPCANStatus error code)

3.5.3 TPUDSBaudrate

Represents a PCAN Baud rate register value. According with the programming language, this type can be a group of defined values or an enumeration.

Syntax

C++

```
#define TPUDSBaudrate WORD

#define PUDS_BAUD_1M 0x0014
#define PUDS_BAUD_800K 0x0016
#define PUDS_BAUD_500K 0x001C
#define PUDS_BAUD_250K 0x011C
#define PUDS_BAUD_125K 0x031C
#define PUDS_BAUD_100K 0x432F
#define PUDS_BAUD_95K 0xC34E
#define PUDS_BAUD_83K 0x852B
#define PUDS_BAUD_50K 0x472F
#define PUDS_BAUD_47K 0x1414
#define PUDS_BAUD_33K 0x8B2F
#define PUDS_BAUD_20K 0x532F
#define PUDS_BAUD_10K 0x672F
#define PUDS_BAUD_5K 0x7F7F
```

Pascal OO

```
{$Z2}
TPUDSBaudrate = (
  PUDS_BAUD_1M = $0014,
```

```
  PUDS_BAUD_800K = $0016,
  PUDS_BAUD_500K = $001C,
  PUDS_BAUD_250K = $011C,
  PUDS_BAUD_125K = $031C,
  PUDS_BAUD_100K = $432F,
  PUDS_BAUD_95K = $C34E,
  PUDS_BAUD_83K = $852B,
  PUDS_BAUD_50K = $472F,
  PUDS_BAUD_47K = $1414,
  PUDS_BAUD_33K = $8B2F,
  PUDS_BAUD_20K = $532F,
  PUDS_BAUD_10K = $672F,
  PUDS_BAUD_5K = $7F7F
);
```

C#

```
public enum TPUDSBaudrate : ushort
{
  PUDS_BAUD_1M = 0x0014,
  PUDS_BAUD_800K = 0x0016,
  PUDS_BAUD_500K = 0x001C,
  PUDS_BAUD_250K = 0x011C,
  PUDS_BAUD_125K = 0x031C,
  PUDS_BAUD_100K = 0x432F,
  PUDS_BAUD_95K = 0xC34E,
  PUDS_BAUD_83K = 0x852B,
  PUDS_BAUD_50K = 0x472F,
  PUDS_BAUD_47K = 0x1414,
  PUDS_BAUD_33K = 0x8B2F,
  PUDS_BAUD_20K = 0x532F,
  PUDS_BAUD_10K = 0x672F,
  PUDS_BAUD_5K = 0x7F7F,
}
```

C++ / CLR

```
public enum class TPUDSBaudrate : UInt16
{
   PUDS_BAUD_1M = 0x0014,
   PUDS_BAUD_800K = 0x0016,
   PUDS_BAUD_500K = 0x001C,
   PUDS_BAUD_250K = 0x011C,
   PUDS_BAUD_125K = 0x031C,
   PUDS_BAUD_100K = 0x432F,
   PUDS_BAUD_95K = 0xC34E,
   PUDS_BAUD_83K = 0x852B,
   PUDS_BAUD_50K = 0x472F,
   PUDS_BAUD_47K = 0x1414,
   PUDS_BAUD_33K = 0x8B2F,
   PUDS_BAUD_20K = 0x532F,
   PUDS_BAUD_10K = 0x672F,
   PUDS_BAUD_5K = 0x7F7F,
};
```

Visual Basic

```
Public Enum TPUDSBaudrate As UInt16
   PUDS_BAUD_1M = &H14
   PUDS_BAUD_800K = &H16
   PUDS_BAUD_500K = &H1C
   PUDS_BAUD_250K = &H11C
   PUDS_BAUD_125K = &H31C
   PUDS_BAUD_100K = &H432F
   PUDS_BAUD_95K = &C34E
   PUDS_BAUD_83K = &852B
   PUDS_BAUD_50K = &H472F
   PUDS_BAUD_47K = &1414
   PUDS_BAUD_33K = &8B2F
   PUDS_BAUD_20K = &H532F
   PUDS_BAUD_10K = &H672F
   PUDS_BAUD_5K = &H7F7F
End Enum
```

Values

Name	Value	Description
PUDS_BAUD_1M	20	1 MBit/s
PUDS_BAUD_800K	22	800 kBit/s
PUDS_BAUD_500K	28	500 kBit/s
PUDS_BAUD_250K	284	250 kBit/s
PUDS_BAUD_125K	796	125 kBit/s
PUDS_BAUD_100K	17199	100 kBit/s
PUDS_BAUD_95K	49998	95,238 kBit/s
PUDS_BAUD_83K	34091	83,333 kBit/s
PUDS_BAUD_50K	18223	50 kBit/s
PUDS_BAUD_47K	5140	47,619 kBit/s
PUDS_BAUD_33K	35631	33,333 kBit/s
PUDS_BAUD_20K	21295	20 kBit/s
PUDS_BAUD_10K	26415	10 kBit/s
PUDS_BAUD_5K	32639	5 kBit/s

See also: UDS_Initialize on page 259 (class method: initialize).

3.5.4 TPUDSHWType

Represents the type of PCAN (not plug & play) hardware to be initialized. According with the programming language, this type can be a group of defined values or an enumeration.

Syntax

C++

```
#define TPUDSHWType BYTE

#define PUDS_TYPE_ISA 0x01
#define PUDS_TYPE_ISA_SJA 0x09
#define PUDS_TYPE_ISA_PHYTEC 0x04
#define PUDS_TYPE_DNG 0x02
#define PUDS_TYPE_DNG_EPP 0x03
#define PUDS_TYPE_DNG_SJA 0x05
#define PUDS_TYPE_DNG_SJA_EPP 0x06
```

Pascal OO

```
{$Z1}
TPUDSHWType = (
   PUDS_TYPE_ISA = $01,
   PUDS_TYPE_ISA_SJA = $09,
   PUDS_TYPE_ISA_PHYTEC = $04,
   PUDS_TYPE_DNG = $02,
   PUDS_TYPE_DNG_EPP = $03,
   PUDS_TYPE_DNG_SJA = $05,
   PUDS_TYPE_DNG_SJA_EPP = $06
);
```

C#

```
public enum TPUDSHWType : byte
{
   PUDS_TYPE_ISA = 0x01,
   PUDS_TYPE_ISA_SJA = 0x09,
   PUDS_TYPE_ISA_PHYTEC = 0x04,
   PUDS_TYPE_DNG = 0x02,
   PUDS_TYPE_DNG_EPP = 0x03,
   PUDS_TYPE_DNG_SJA = 0x05,
   PUDS_TYPE_DNG_SJA_EPP = 0x06,
}
```

C++ / CLR

```
public enum class TPUDSHWType : Byte
{
   PUDS_TYPE_ISA = 0x01,
   PUDS_TYPE_ISA_SJA = 0x09,
   PUDS_TYPE_ISA_PHYTEC = 0x04,
   PUDS_TYPE_DNG = 0x02,
   PUDS_TYPE_DNG_EPP = 0x03,
   PUDS_TYPE_DNG_SJA = 0x05,
   PUDS_TYPE_DNG_SJA_EPP = 0x06,
};
```

Visual Basic

```
Public Enum TPUDSHWType As Byte
   PUDS_TYPE_ISA = &H1
   PUDS_TYPE_ISA_SJA = &H9
   PUDS_TYPE_ISA_PHYTEC = &H4
   PUDS_TYPE_DNG = &H2
   PUDS_TYPE_DNG_EPP = &H3
   PUDS_TYPE_DNG_SJA = &H5
   PUDS_TYPE_DNG_SJA_EPP = &H6
End Enum
```

Values

Name	Value	Description
PUDS_TYPE_ISA	1	PCAN-ISA 82C200
PUDS_TYPE_ISA_SJA	9	PCAN-ISA SJA1000
PUDS_TYPE_ISA_PHYTEC	4	PHYTEC ISA
PUDS_TYPE_DNG	2	PCAN-Dongle 82C200
PUDS_TYPE_DNG_EPP	3	PCAN-Dongle EPP 82C200
PUDS_TYPE_DNG_SJA	5	PCAN-Dongle SJA1000
PUDS_TYPE_DNG_SJA_EPP	6	PCAN-Dongle EPP SJA1000

See also: PUDS_Initialize (class method: initialize).

3.5.5 TPUDSResult

Represents the network status of a communicated UDS message (as returned by the ISO-TP layer).

Syntax

C++

```
#define PUDS_RESULT_N_OK            0x00
#define PUDS_RESULT_N_TIMEOUT_A     0x01
#define PUDS_RESULT_N_TIMEOUT_Bs    0x02
#define PUDS_RESULT_N_TIMEOUT_Cr    0x03
#define PUDS_RESULT_N_WRONG_SN      0x04
#define PUDS_RESULT_N_INVALID_FS    0x05
#define PUDS_RESULT_N_UNEXP_PDU     0x06
#define PUDS_RESULT_N_WFT_OVRN      0x07
#define PUDS_RESULT_N_BUFFER_OVFLW  0x08
#define PUDS_RESULT_N_ERROR         0x09
```

Pascal OO

```
{$Z1}
TPUDSResult = (
    PUDS_RESULT_N_OK = $00,
    PUDS_RESULT_N_TIMEOUT_A = $01,
    PUDS_RESULT_N_TIMEOUT_BS = $02,
    PUDS_RESULT_N_TIMEOUT_CR = $03,
    PUDS_RESULT_N_WRONG_SN = $04,
    PUDS_RESULT_N_INVALID_FS = $05,
    PUDS_RESULT_N_UNEXP_PDU = $06,
    PUDS_RESULT_N_WFT_OVRN = $07,
    PUDS_RESULT_N_BUFFER_OVFLW = $08,
    PUDS_RESULT_N_ERROR = $09
);
```

C#

```
public enum TPUDSResult : byte
{
    PUDS_RESULT_N_OK = 0x00,
    PUDS_RESULT_N_TIMEOUT_A = 0x01,
    PUDS_RESULT_N_TIMEOUT_BS = 0x02,
    PUDS_RESULT_N_TIMEOUT_CR = 0x03,
    PUDS_RESULT_N_WRONG_SN = 0x04,
    PUDS_RESULT_N_INVALID_FS = 0x05,
    PUDS_RESULT_N_UNEXP_PDU = 0x06,
    PUDS_RESULT_N_WFT_OVRN = 0x07,
    PUDS_RESULT_N_BUFFER_OVFLW = 0x08,
    PUDS_RESULT_N_ERROR = 0x09,
}
```

C++ / CLR

```
public enum TPUDSResult : Byte
{
      PUDS_RESULT_N_OK = 0x00,
      PUDS_RESULT_N_TIMEOUT_A = 0x01,
      PUDS_RESULT_N_TIMEOUT_BS = 0x02,
      PUDS_RESULT_N_TIMEOUT_CR = 0x03,
      PUDS_RESULT_N_WRONG_SN = 0x04,
      PUDS_RESULT_N_INVALID_FS = 0x05,
```

```
      PUDS_RESULT_N_UNEXP_PDU = 0x06,
      PUDS_RESULT_N_WFT_OVRN = 0x07,
      PUDS_RESULT_N_BUFFER_OVFLW = 0x08,
      PUDS_RESULT_N_ERROR = 0x09,
};
```

Visual Basic

```
Public Enum TPUDSResult As Byte
    PUDS_RESULT_N_OK = &H0
    PUDS_RESULT_N_TIMEOUT_A = &H1
    PUDS_RESULT_N_TIMEOUT_BS = &H2
    PUDS_RESULT_N_TIMEOUT_CR = &H3
    PUDS_RESULT_N_WRONG_SN = &H4
    PUDS_RESULT_N_INVALID_FS = &H5
    PUDS_RESULT_N_UNEXP_PDU = &H6
    PUDS_RESULT_N_WFT_OVRN = &H7
    PUDS_RESULT_N_BUFFER_OVFLW = &H8
    PUDS_RESULT_N_ERROR = &H9
End Enum
```

Values

Name	Value	Description
PUDS_RESULT_N_OK	0	No network error
PUDS_RESULT_N_TIMEOUT_A	1	Timeout occured between 2 frames transmission (sender and receiver side)
PUDS_RESULT_N_TIMEOUT_Bs	2	Sender side timeout while waiting for flow control frame
PUDS_RESULT_N_TIMEOUT_Cr	3	Receiver side timeout while waiting for consecutive frame
PUDS_RESULT_N_WRONG_SN	4	Unexpected sequence number
PUDS_RESULT_N_INVALID_FS	5	Invalid or unknown FlowStatus
PUDS_RESULT_N_UNEXP_PDU	6	Unexpected protocol data unit
PUDS_RESULT_N_WFT_OVRN	7	Reception of flow control WAIT frame that exceeds the maximum counter defined by PUDS_PARAM_WFT_MAX
PUDS_RESULT_N_BUFFER_OVFLW	8	Buffer on the receiver side cannot store the data length (server side only)
PUDS_RESULT_N_ERROR	9	General error

See also: TPUDSMsg on page 14.

3.5.6 TPUDSParameter

Represents a PCAN-UDS parameter or a PCAN-UDS Value that can be read or set. According with the programming language, this type can be a group of defined values or an enumeration. With some exceptions, a channel must first be initialized before their parameters can be read or set.

Syntax

C++

```
#define TPUDSParameter BYTE
#define PUDS_PARAM_SERVER_ADDRESS 0xC1
#define PUDS_PARAM_SERVER_FILTER 0xC2
#define PUDS_PARAM_TIMEOUT_REQUEST 0xC3
#define PUDS_PARAM_TIMEOUT_RESPONSE 0xC4
#define PUDS_PARAM_SESSION_INFO 0xC5
#define PUDS_PARAM_API_VERSION 0xC6
```

```
#define PUDS_PARAM_RECEIVE_EVENT 0xC7
#define PUDS_PARAM_MAPPING_ADD 0xC8
#define PUDS_PARAM_MAPPING_REMOVE 0xC9
#define PUDS_PARAM_BLOCK_SIZE 0xE1
#define PUDS_PARAM_SEPERATION_TIME 0xE2
#define PUDS_PARAM_DEBUG 0xE3
#define PUDS_PARAM_CHANNEL_CONDITION 0xE4
#define PUDS_PARAM_WFT_MAX 0xE5
#define PUDS_PARAM_CAN_DATA_PADDING 0xE8
#define PUDS_PARAM_PADDING_VALUE 0xED
```

Pascal OO

```
TPUDSParameter = (
    PUDS_PARAM_SERVER_ADDRESS = $C1,
    PUDS_PARAM_SERVER_FILTER = $C2,
    PUDS_PARAM_TIMEOUT_REQUEST = $C3,
    PUDS_PARAM_TIMEOUT_RESPONSE = $C4,
    PUDS_PARAM_SESSION_INFO = $C5,
    PUDS_PARAM_API_VERSION = $C6,
    PUDS_PARAM_RECEIVE_EVENT = $C7,
    PUDS_PARAM_MAPPING_ADD = $C8,
    PUDS_PARAM_MAPPING_REMOVE = $C9,
    PUDS_PARAM_BLOCK_SIZE = $E1,
    PUDS_PARAM_SEPERATION_TIME = $E2,
    PUDS_PARAM_DEBUG = $E3,
    PUDS_PARAM_CHANNEL_CONDITION = $E4,
    PUDS_PARAM_WFT_MAX = $E5,
    PUDS_PARAM_CAN_DATA_PADDING = $E8
    PUDS_PARAM_PADDING_VALUE = $ED
);
```

C#

```
public enum TPUDSParameter : byte
{
    PUDS_PARAM_SERVER_ADDRESS = 0xC1,
    PUDS_PARAM_SERVER_FILTER = 0xC2,
    PUDS_PARAM_TIMEOUT_REQUEST = 0xC3,
    PUDS_PARAM_TIMEOUT_RESPONSE = 0xC4,
    PUDS_PARAM_SESSION_INFO = 0xC5,
    PUDS_PARAM_API_VERSION = 0xC6,
    PUDS_PARAM_RECEIVE_EVENT = 0xC7,
    PUDS_PARAM_MAPPING_ADD = 0xC8,
    PUDS_PARAM_MAPPING_REMOVE = 0xC9,
    PUDS_PARAM_BLOCK_SIZE = 0xE1,
    PUDS_PARAM_SEPERATION_TIME = 0xE2,
    PUDS_PARAM_DEBUG = 0xE3,
    PUDS_PARAM_CHANNEL_CONDITION = 0xE4,
    PUDS_PARAM_WFT_MAX = 0xE5,
    PUDS_PARAM_CAN_DATA_PADDING = 0xE8,
    PUDS_PARAM_PADDING_VALUE = 0xED,
}
```

C++ / CLR

```
public enum TPUDSParameter : Byte
{
      PUDS_PARAM_SERVER_ADDRESS = 0xC1,
      PUDS_PARAM_SERVER_FILTER = 0xC2,
```

```
      PUDS_PARAM_TIMEOUT_REQUEST = 0xC3,
      PUDS_PARAM_TIMEOUT_RESPONSE = 0xC4,
      PUDS_PARAM_SESSION_INFO = 0xC5,
      PUDS_PARAM_API_VERSION = 0xC6,
      PUDS_PARAM_RECEIVE_EVENT = 0xC7,
      PUDS_PARAM_MAPPING_ADD = 0xC8,
      PUDS_PARAM_MAPPING_REMOVE = 0xC9,
      PUDS_PARAM_BLOCK_SIZE = 0xE1,
      PUDS_PARAM_SEPERATION_TIME = 0xE2,
      PUDS_PARAM_DEBUG = 0xE3,
      PUDS_PARAM_CHANNEL_CONDITION = 0xE4,
      PUDS_PARAM_WFT_MAX = 0xE5,
      PUDS_PARAM_CAN_DATA_PADDING = 0xE8,
};
```

Visual Basic

```
Public Enum TPUDSParameter As Byte
    PUDS_PARAM_SERVER_ADDRESS = &HC1
    PUDS_PARAM_SERVER_FILTER = &HC2
    PUDS_PARAM_TIMEOUT_REQUEST = &HC3
    PUDS_PARAM_TIMEOUT_RESPONSE = &HC4
    PUDS_PARAM_SESSION_INFO = &HC5
    PUDS_PARAM_API_VERSION = &HC6
    PUDS_PARAM_RECEIVE_EVENT = &HC7
    PUDS_PARAM_MAPPING_ADD = &HC8
    PUDS_PARAM_MAPPING_REMOVE = &HC9
    PUDS_PARAM_BLOCK_SIZE = &HE1
    PUDS_PARAM_SEPERATION_TIME = &HE2
    PUDS_PARAM_DEBUG = &HE3
    PUDS_PARAM_CHANNEL_CONDITION = &HE4
    PUDS_PARAM_WFT_MAX = &HE5
    PUDS_PARAM_CAN_DATA_PADDING = &HE8
    PUDS_PARAM_PADDING_VALUE = &HED
End Enum
```

Values

Name	Value	Data type	Description
PUDS_PARAM_SERVER_ADDRESS	0xC1 (193)	WORD (2 Bytes)	Physical address of the server (or ECU)
PUDS_PARAM_SERVER_FILTER	0xC2 (194)	WORD (2 Bytes)	Filter for functional addresses
PUDS_PARAM_TIMEOUT_REQUEST	0xC3 (195)	DWORD (4 Bytes)	Maximum time allowed by the client to transmit a request
PUDS_PARAM_TIMEOUT_RESPONSE	0xC4 (196)	DWORD (4 Bytes)	Maximum time allowed by the client to receive a response
PUDS_PARAM_SESSION_INFO	0xC5 (197)	TPUDSSessionInfo	UDS current diagnostic session information
PUDS_PARAM_API_VERSION	0xC6 (198)	String	API version of the PCAN-UDS API
PUDS_PARAM_RECEIVE_EVENT	0xC7 (199)	String	Define UDS receive-event handler, require a pointer to an event HANDLE
PUDS_PARAM_MAPPING_ADD	0xC8 (200)	TPUDSMessage	Defines a new ISO-TP mapping
PUDS_PARAM_MAPPING_REMOVE	0xC9 (201)	TPUDSMessage	Removes an ISO-TP mapping
PUDS_PARAM_BLOCK_SIZE	0xE1 (225)	Byte	ISO-TP "BlockSize" (BS) parameter
PUDS_PARAM_SEPERATION_TIME	0xE2 (226)	Byte	ISO-TP "SeparationTime" (STmin) parameter
PUDS_PARAM_DEBUG	0xE3 (227)	Byte	Debug mode
PUDS_PARAM_CHANNEL_CONDITION	0xE4 (228)	Byte	PCAN-UDS channel condition
PUDS_PARAM_WFT_MAX	0xE5 (229)	Integer	ISO-TP "N_WFTmax" parameter
PUDS_PARAM_CAN_DATA_PADDING	0xE8 (232)	Byte	ISO-TP CAN frame data handling mode

Name	Value	Data type	Description
PUDS_PARAM_PADDING_VALUE	0xED (237)	Byte	Value used when CAN Data padding is enabled

Characteristics

PUDS_PARAM_SERVER_ADDRESS

Access: R W

Description: This value is used to define the address of the client/server. Defining this parameter is required as it allows configuration of the underlying ISO-TP API. By default the address is set to the standard address of external test equipment (0xF1). Although most UDS addresses are 1 byte data, the parameter requires a 2 bytes value in order to be compatible with ISO 14229-1:2006 addresses (see PUDS_SERVER_ADDR_MASK_ENHANCED_ISO_15765_3, 0x7FF). To define an ISO 14229-1:2006 address, add the flag PUDS_SERVER_ADDR_FLAG_ENHANCED_ISO_15765_3 (0x1000) to the address (using a bitwise OR operation, i.e.: (PUDS_SERVER_ADDR_FLAG_ENHANCED_ISO_15765_3 | 0x123).

Possible Values: 0x00 to 0xFF, or 0x1000 to 0x17FF for enhanced ISO 14229-1:2006 addresses.

Default Value: `PUDS_SERVER_ADDR_TEST_EQUIPMENT` (0xF1).

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_SERVER_Filter

Access: R W

Description: This parameter allows to add or remove addresses to the filter list. By default, the filter list denies every communication except the one with the server address. To add another address to the filter list (like a functional address the server should respond to), add the flag `PUDS_SERVER_FILTER_LISTEN` (0x8000) to the wanted address (using a bitwise OR operation). To remove a previously set address, use the flag `PUDS_SERVER_FILTER_IGNORE` (0x0000) with the same address.

Possible Values: 0x00 to 0xFF (or 0x7FF for enhanced ISO 14229-1:2006 addresses) in conjunction with the flags PUDS_SERVER_FILTER_LISTEN or PUDS_SERVER_FILTER_IGNORE. Example: to allow the address 0x11 use the value (PUDS_SERVER_FILTER_LISTEN | 0x11), to remove it use (PUDS_SERVER_FILTER_IGNORE | 0x11).

Default Value: none.

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_TIMEOUT_REQUEST

Access: R W

Description: This value defines the maximum waiting time to receive a confirmation of a transmission. It is used in UDS utility functions like UDS_WaitForService.

Possible Values: 0x00 (unlimited) to 0xFFFFFFFF.

Default Value: `PUDS_TIMEOUT_RESPONSE` (10000 ms).

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_TIMEOUT_RESPONSE

Access: R W

Description: This value defines the maximum waiting time to receive a response indication. Note that the exact timeout value is the sum of this parameter and the timeout defined in the active diagnostic session. It is used in UDS utility functions like UDS_WaitForService.

Possible Values: 0x00 (unlimited) to 0xFFFFFFFF.

Default Value: `PUDS_TIMEOUT_REQUEST` (10000 ms).

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_SESSION_INFO

Access: R W

Description: This parameter is used to read or override the current diagnostic session. The diagnostic session information is a value which is internally updated when the UDS service UDS_SvcDiagnosticSessionControl is invoked: it keeps track of the active session and its associated timeouts. If a non-default diagnostic session is active, a keep-alive mechanism (using physically addressed TesterPresent service requests with the “suppress positive response message” flag) is automatically activated according to UDS standard. If a user wants to deactivate this mechanism, he/she will have to get the current session information of the client and change the session type to PUDS_SVC_PARAM_DSC_DS (the default diagnostic session). Be careful that this change will only affect the client side and the ECU will not be aware of it, moreover the client will have to keep alive the active session by its own. The user can also define a different session with custom networking parameters, for instance setting back a non-default session and providing a NETADDRINFO parameter with functional addressing will send functionally addressed TesterPresent requests.

Possible Values: any TPUDSSessionInfo strucuture.

Default Value: automatically updated when the response of the service UDS_SvcDiagnosticSessionControl is processed (with the UDS_ProcessResponse function or higher-level functions: UDS_WaitForMultipleMessage, UDS_WaitForService and UDS_WaitForServiceFunctional).

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_API_Version

Access: R

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

Possible Values: The value is a null-terminated string indication the version number of the API implementation. The returned text has the following form: x,x,x,x for major, minor, release and build. It represents the binary version of the API, within two 32-bit integers, defined by four 16-bit integers. The length of this text value will have a maximum length of 24 bytes, 5 bytes for represent each 16-bit value, three separator characters (, or .) and the null-termination.

Default Value: NA.

PCAN-Device: NA. Any PCAN device can be used, including the PUDS_NONEBUS channel.

PUDS_PARAM_RECEIVE_EVENT

Access: R W

Description: This parameter is used to let the PCAN-UDS API notify an application when UDS messages are available to be read. In this form, message processing tasks of an application can react faster and make a more efficient use of the processor time.

Possible Values: This value has to be a handle for an event object returned by the Windows API function CreateEvent or the value 0 (IntPtr.Zero in a managed environment). When setting this parameter, the value of 0 resets the parameter in the PCAN-ISO-TP API. Reading a value of 0 indicates that no event handle is set. For more information about reading with events, please refer to the topic Using Events.

Note: .NET environment should use the prototype SetValue with a NumericBuffer, as in the following C# example:

```
ManualResetEvent hEvent = new ManualResetEvent(false);
uint myEventUint = (uint)hEvent.SafeWaitHandle.DangerousGetHandle().ToInt32();
TPUDSStatus status = UDSApi.SetValue(channel, TPUDSParameter.PUDS_PARAM_RECEIVE_EVENT,
                                     ref myEventUint, (uint)Marshal.SizeOf(IntPtr.Zero));
```

Default Value: Disabled (0).

PCAN-Device: All PCAN devices (excluding PCANTP_NONEBUS channel).

PUDS_PARAM_MAPPING_ADD

Access: W

Description: This parameter is used to configure a mapping to the underlying PCAN-ISO-TP API. This allows a user to map a custom Network Address Information to a CAN Identifier, thus allowing to communicate with non standard UDS requests or responses.

Possible Values: This value has to be a pointer to a valid TPUDSMsg structure where its network address information (NETADDRINFO) is fully defined. Furthermore, the data of the message shall contain the following information:

- If the mapping concerns an Unacknowledge Segmented Data Transfert (USDT), the length of the message must be 8 and DATA.RAW[0..3] will hold the mapped CAN ID and DATA.RAW[4..7] will hold the mapped CAN ID for the ISO-TP automatic responses. For instance, in OBD the tester present receives messages from ECU#1 with CAN ID 0x7E8 and responds with 0x7E0 (ex: DATA.RAW[0..7] = 00 00 07 E8 00 00 07 E0). With functionally addressed messages there are no internal ISO-TP responses, so the CAN ID response field should be set to -1 (0xFFFFFFFF).
- If the mapping concerns an Unacknowledge Unsegmented Data Transfert (UUDT), the length of the message must be 4 and DATA.RAW[0..3] will hold the mapped CAN ID. For instance, a tester present can receive UUDT with CAN ID 0x6E8 messages after requesting the readDataByPeriodicDataIdentifier from ECU#1(ex: DATA.RAW[0..3] = 00 00 06 E8).
- For a complete example, check §4.3.3 PCAN-UDS Example C:\Users\Christoph\Documents\Doku 2019\PCAN-UDS-API\Phase 2\PCAN-UDS-API_UserMan_eng.doc - _PCAN-UDS_example#_PCAN-UDS_example on page 343.

Note: If an UUDT mapping is added, PCAN-ISO-TP will automatically set its configuration to allow the reception of non-ISO-TP messages (i.e. standard CAN ID).

Default Value: Disabled (0).

PCAN-Device: All PCAN devices (excluding PCANTP_NONEBUS channel).

PUDS_PARAM_MAPPING_REMOVE

Access: W

Description: This parameter is used to remove a previously defined mapping from the underlying PCAN-ISO-TP API.

Possible Values: This value has to be a pointer to a valid TPUDSMsg structure where the network address information (NETADDRINFO) has at least the PROTOCOL information defined and the data of the message shall contain the following information the CAN ID of the mapping. That is DATA.RAW[0..3] will hold the previously mapped CAN ID (ex: DATA.RAW[0..3] = 00 00 06 E8).

Note that if the last UUDT mapping is added, PCAN-ISO-TP will automatically set its configuration to disable reception of non ISO-TP message (i.e. standard CAN ID).

Default Value: Disabled (0).

PCAN-Device: All PCAN devices (excluding PCANTP_NONEBUS channel).

PUDS_PARAM_BLOCK_SIZE

Access: R W

Description: This value is used to set the BlockSize (BS) parameter defined in the ISO-TP standard: it indicates to the sender the maximum number of consecutive frames that can be received without an intermediate FlowControl frame from the receiving network entity. A value of 0 indicates that no limit is set and the sending network layer entity shall send all remaining consecutive frames.

Possible Values: 0x00 (unlimited) to 0xFF.

Default Value: 10.

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_SEPARATION_TIME

Access: R W

Description: This value is used to set the SeparationTime (STmin) parameter defined in the ISO-TP standard: it indicates the minimum time the sender is to wait between the transmissions of two Consecutive Frames.

Possible Values: 0x00 (unlimited) to 0x7F (range from 0ms to 127ms). Note: as set in ISO-TP standard, values from 0xF1 to 0xF9 should define a range from 100µs to 900µs, but due to system time resolution limitation those value are defaulted to 1ms. Other values are ISO reserved.

Default Value: 10 ms.

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_DEBUG

Access: R W

Description: This parameter is used to control debug mode. If enabled, any received or transmitted CAN frames will be printed to the standard output.

Possible Values: PUDS_DEBUG_NONE disables debug mode and PUDS_DEBUG_CAN enables it.

Default Value: PUDS_DEBUG_NONE.

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

PUDS_PARAM_CHANNEL_CONDITION

Access: R

Description: This parameter is used to check and detect available PCAN hardware on a computer, even before trying to connect any of them. This is useful when an application wants the user to select which hardware should be using in a communication session.

Possible Values: This parameter can have one of these values: PUDS_CHANNEL_UNAVAILABLE, PUDS_CHANNEL_AVAILABLE and PUDS_CHANNEL_OCCUPIED.

Default Value: NA.

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

Note: It is not needed to have a PCAN channel initialized before asking for its condition.

PUDS_PARAM_WFT_MAX

Access: R W

Description: This parameter is used to set the maximum number of “FlowControl.Wait” frame transmission parameter (N_WFTmax) defined in the ISO-TP standard: it indicates how many FlowControl frames with the Wait status can be transmitted by a receiver in a row.

Possible Values: Any positive number.

Default Value: 0x10.

PCAN-Device: Any PCAN device can be used, including the PUDS_NONEBUS channel.

Note: also this parameter is set globally, channels will use the value set when they are initialized, so it is possible to define different values of N_WFTmax on separate channels. Consequently, once a channel is initialized, changing the WFTmax parameter will not affect that channel.

PUDS_PARAM_CAN_DATA_PADDING

Access: R W

Description: This parameter is used to define if the API should uses CAN data optimization or CAN data padding: the first case will optimize the CAN DLC to avoid sending unnecessary data, on the other hand with CAN data padding the API will always send CAN frames with a DLC of 8 and pads the data with zeroes.

Possible Values: PUDS_CAN_DATA_PADDING_NONE disables data padding (enabling CAN data optimization) and PUDS_CAN_DATA_PADDING_ON enables data padding.

Default Value: PUDS_CAN_DATA_PADDING_ON since ECUs that do not support CAN data optimization may not respond to UDS/CAN-TP messages.

PCAN-Device: All PCAN devices (excluding the PUDS_NONEBUS channel).

PUDS_PARAM_PADDING_VALUE

Access: R W

Description: This parameter is used to define the value for CAN data padding when it is enabled.

Possible values: Any value from 0x00 to 0xFF.

Default value: 0x55 (PUDS_CAN_DATA_PADDING_VALUE).

PCAN-Device: All PCAN devices (excluding PUDS_NONEBUS channel).

See also: Parameter Value Definitions on page 332.

3.5.7 TPUDSService

Represents a UDS on CAN service identifier.

Syntax

C++

```
#define PUDS_SI_DiagnosticSessionControl          0x10
#define PUDS_SI_ECUReset                          0x11
#define PUDS_SI_SecurityAccess                    0x27
#define PUDS_SI_CommunicationControl              0x28
#define PUDS_SI_TesterPresent                     0x3E
#define PUDS_SI_AccessTimingParameter             0x83
#define PUDS_SI_SecuredDataTransmission           0x84
#define PUDS_SI_ControlDTCSetting                 0x85
#define PUDS_SI_ResponseOnEvent                   0x86
#define PUDS_SI_LinkControl                       0x87
#define PUDS_SI_ReadDataByIdentifier              0x22
#define PUDS_SI_ReadMemoryByAddress               0x23
#define PUDS_SI_ReadScalingDataByIdentifier       0x24
#define PUDS_SI_ReadDataByPeriodicIdentifier      0x2A
#define PUDS_SI_DynamicallyDefineDataIdentifier   0x2C
#define PUDS_SI_WriteDataByIdentifier             0x2E
#define PUDS_SI_WriteMemoryByAddress              0x3D
#define PUDS_SI_ClearDiagnosticInformation        0x14
#define PUDS_SI_ReadDTCInformation                0x19
#define PUDS_SI_InputOutputControlByIdentifier    0x2F
#define PUDS_SI_RoutineControl                    0x31
#define PUDS_SI_RequestDownload                   0x34
#define PUDS_SI_RequestUpload                     0x35
#define PUDS_SI_TransferData                      0x36
#define PUDS_SI_RequestTransferExit               0x37
#define PUDS_NR_SI                                0x7F
```

Pascal OO

```
TPUDSService = (
    PUDS_SI_DiagnosticSessionControl = $10,
    PUDS_SI_ECUReset = $11,
    PUDS_SI_SecurityAccess = $27,
    PUDS_SI_CommunicationControl = $28,
    PUDS_SI_TesterPresent = $3E,
    PUDS_SI_AccessTimingParameter = $83,
    PUDS_SI_SecuredDataTransmission = $84,
    PUDS_SI_ControlDTCSetting = $85,
    PUDS_SI_ResponseOnEvent = $86,
    PUDS_SI_LinkControl = $87,
    PUDS_SI_ReadDataByIdentifier = $22,
    PUDS_SI_ReadMemoryByAddress = $23,
    PUDS_SI_ReadScalingDataByIdentifier = $24,
    PUDS_SI_ReadDataByPeriodicIdentifier = $2A,
    PUDS_SI_DynamicallyDefineDataIdentifier = $2C,
    PUDS_SI_WriteDataByIdentifier = $2E,
    PUDS_SI_WriteMemoryByAddress = $3D,
    PUDS_SI_ClearDiagnosticInformation = $14,
    PUDS_SI_ReadDTCInformation = $19,
    PUDS_SI_InputOutputControlByIdentifier = $2F,
    PUDS_SI_RoutineControl = $31,
    PUDS_SI_RequestDownload = $34,
    PUDS_SI_RequestUpload = $35,
    PUDS_SI_TransferData = $36,
    PUDS_SI_RequestTransferExit = $37,

    PUDS_NR_SI = $7f
);
```

C#

```
public enum TPUDSService : byte
{
    PUDS_SI_DiagnosticSessionControl = 0x10,
    PUDS_SI_ECUReset = 0x11,
    PUDS_SI_SecurityAccess = 0x27,
    PUDS_SI_CommunicationControl = 0x28,
    PUDS_SI_TesterPresent = 0x3E,
    PUDS_SI_AccessTimingParameter = 0x83,
    PUDS_SI_SecuredDataTransmission = 0x84,
    PUDS_SI_ControlDTCSetting = 0x85,
    PUDS_SI_ResponseOnEvent = 0x86,
    PUDS_SI_LinkControl = 0x87,
    PUDS_SI_ReadDataByIdentifier = 0x22,
    PUDS_SI_ReadMemoryByAddress = 0x23,
    PUDS_SI_ReadScalingDataByIdentifier = 0x24,
    PUDS_SI_ReadDataByPeriodicIdentifier = 0x2A,
    PUDS_SI_DynamicallyDefineDataIdentifier = 0x2C,
    PUDS_SI_WriteDataByIdentifier = 0x2E,
    PUDS_SI_WriteMemoryByAddress = 0x3D,
    PUDS_SI_ClearDiagnosticInformation = 0x14,
    PUDS_SI_ReadDTCInformation = 0x19,
    PUDS_SI_InputOutputControlByIdentifier = 0x2F,
    PUDS_SI_RoutineControl = 0x31,
    PUDS_SI_RequestDownload = 0x34,
    PUDS_SI_RequestUpload = 0x35,
    PUDS_SI_TransferData = 0x36,
    PUDS_SI_RequestTransferExit = 0x37,

    PUDS_NR_SI = 0x7f,
}
```

C++ / CLR

```
public enum TPUDSService : Byte
{
      PUDS_SI_DiagnosticSessionControl = 0x10,
      PUDS_SI_ECUReset = 0x11,
      PUDS_SI_SecurityAccess = 0x27,
      PUDS_SI_CommunicationControl = 0x28,
      PUDS_SI_TesterPresent = 0x3E,
      PUDS_SI_AccessTimingParameter = 0x83,
      PUDS_SI_SecuredDataTransmission = 0x84,
      PUDS_SI_ControlDTCSetting = 0x85,
      PUDS_SI_ResponseOnEvent = 0x86,
      PUDS_SI_LinkControl = 0x87,
      PUDS_SI_ReadDataByIdentifier = 0x22,
      PUDS_SI_ReadMemoryByAddress = 0x23,
      PUDS_SI_ReadScalingDataByIdentifier = 0x24,
      PUDS_SI_ReadDataByPeriodicIdentifier = 0x2A,
      PUDS_SI_DynamicallyDefineDataIdentifier = 0x2C,
      PUDS_SI_WriteDataByIdentifier = 0x2E,
      PUDS_SI_WriteMemoryByAddress = 0x3D,
      PUDS_SI_ClearDiagnosticInformation = 0x14,
      PUDS_SI_ReadDTCInformation = 0x19,
      PUDS_SI_InputOutputControlByIdentifier = 0x2F,
      PUDS_SI_RoutineControl = 0x31,
      PUDS_SI_RequestDownload = 0x34,
      PUDS_SI_RequestUpload = 0x35,
      PUDS_SI_TransferData = 0x36,
```

```
      PUDS_SI_RequestTransferExit = 0x37,

      PUDS_NR_SI = 0x7f,
};
```

Visual Basic

```
Public Enum TPUDSService As Byte
    PUDS_SI_DiagnosticSessionControl = &H10
    PUDS_SI_ECUReset = &H11
    PUDS_SI_SecurityAccess = &H27
    PUDS_SI_CommunicationControl = &H28
    PUDS_SI_TesterPresent = &H3E
    PUDS_SI_AccessTimingParameter = &H83
    PUDS_SI_SecuredDataTransmission = &H84
    PUDS_SI_ControlDTCSetting = &H85
    PUDS_SI_ResponseOnEvent = &H86
    PUDS_SI_LinkControl = &H87
    PUDS_SI_ReadDataByIdentifier = &H22
    PUDS_SI_ReadMemoryByAddress = &H23
    PUDS_SI_ReadScalingDataByIdentifier = &H24
    PUDS_SI_ReadDataByPeriodicIdentifier = &H2A
    PUDS_SI_DynamicallyDefineDataIdentifier = &H2C
    PUDS_SI_WriteDataByIdentifier = &H2E
    PUDS_SI_WriteMemoryByAddress = &H3D
    PUDS_SI_ClearDiagnosticInformation = &H14
    PUDS_SI_ReadDTCInformation = &H19
    PUDS_SI_InputOutputControlByIdentifier = &H2F
    PUDS_SI_RoutineControl = &H31
    PUDS_SI_RequestDownload = &H34
    PUDS_SI_RequestUpload = &H35
    PUDS_SI_TransferData = &H36
    PUDS_SI_RequestTransferExit = &H37

    PUDS_NR_SI = &H7F
End Enum
```

Values

Name	Value	Description
PUDS_SI_DiagnosticSessionControl	0x10 (16)	Identifier of the UDS Service DiagnosticSessionControl
PUDS_SI_ECUReset	0x11 (17)	Identifier of the UDS Service ECUReset
PUDS_SI_SecurityAccess	0x27 (39)	Identifier of the UDS Service SecurityAccess
PUDS_SI_CommunicationControl	0x28 (40)	Identifier of the UDS Service CommunicationControl
PUDS_SI_TesterPresent	0x3E (62)	Identifier of the UDS Service TesterPresent
PUDS_SI_AccessTimingParameter	0x83 (131)	Identifier of the UDS Service AccessTimingParameter
PUDS_SI_SecuredDataTransmission	0x84 (132)	Identifier of the UDS Service SecuredDataTransmission
PUDS_SI_ControlDTCSetting	0x85 (133)	Identifier of the UDS Service ControlDTCSetting
PUDS_SI_ResponseOnEvent	0x86 (134)	Identifier of the UDS Service ResponseOnEvent
PUDS_SI_LinkControl	0x87 (135)	Identifier of the UDS Service LinkControl
PUDS_SI_ReadDataByIdentifier	0x22 (34)	Identifier of the UDS Service ReadDataByIdentifier
PUDS_SI_ReadMemoryByAddress	0x23 (35)	Identifier of the UDS Service ReadMemoryByAddress
PUDS_SI_ReadScalingDataByIdentifier	0x24 (36)	Identifier of the UDS Service ReadScalingDataByIdentifier
PUDS_SI_ReadDataByPeriodicIdentifier	0x2A (42)	Identifier of the UDS Service ReadDataByPeriodicIdentifier
PUDS_SI_DynamicallyDefineDataIdentifier	0x2C (44)	Identifier of the UDS Service DynamicallyDefineDataIdentifier
PUDS_SI_WriteDataByIdentifier	0x2E (46)	Identifier of the UDS Service WriteDataByIdentifier

Name	Value	Description
PUDS_SI_WriteMemoryByAddress	0x3D (61)	Identifier of the UDS Service WriteMemoryByAddress
PUDS_SI_ClearDiagnosticInformation	0x14 (20)	Identifier of the UDS Service ClearDiagnosticInformation
PUDS_SI_ReadDTCInformation	0x19 (25)	Identifier of the UDS Service ReadDTCInformation
PUDS_SI_InputOutputControlByIdentifier	0x2F (47)	Identifier of the UDS Service InputOutputControlByIdentifier
PUDS_SI_RoutineControl	0x31 (49)	Identifier of the UDS Service RoutineControl
PUDS_SI_RequestDownload	0x34 (52)	Identifier of the UDS Service RequestDownload
PUDS_SI_RequestUpload	0x35 (53)	Identifier of the UDS Service RequestUpload
PUDS_SI_TransferData	0x36 (54)	Identifier of the UDS Service TransferData
PUDS_SI_RequestTransferExit	0x37 (55)	Identifier of the UDS Service RequestTransferExit
PUDS_NR_SI	0x7F (127)	UDS Negative response code identifier

3.5.8 TPUDSAddress

Represents the legislated addresses used with OBD (ISO-15765-4) communication.

Syntax

C++

```
#define PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT            0xF1
#define PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL            0x33
#define PUDS_ISO_15765_4_ADDR_ECU_1                     0x01
#define PUDS_ISO_15765_4_ADDR_ECU_2                     0x02
#define PUDS_ISO_15765_4_ADDR_ECU_3                     0x03
#define PUDS_ISO_15765_4_ADDR_ECU_4                     0x04
#define PUDS_ISO_15765_4_ADDR_ECU_5                     0x05
#define PUDS_ISO_15765_4_ADDR_ECU_6                     0x06
#define PUDS_ISO_15765_4_ADDR_ECU_7                     0x07
#define PUDS_ISO_15765_4_ADDR_ECU_8                     0x08
```

Pascal OO

```
TPUDSAddress = (
    PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT = $F1,
    PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL = $33,
    PUDS_ISO_15765_4_ADDR_ECU_1 = $01,
    PUDS_ISO_15765_4_ADDR_ECU_2 = $02,
    PUDS_ISO_15765_4_ADDR_ECU_3 = $03,
    PUDS_ISO_15765_4_ADDR_ECU_4 = $04,
    PUDS_ISO_15765_4_ADDR_ECU_5 = $05,
    PUDS_ISO_15765_4_ADDR_ECU_6 = $06,
    PUDS_ISO_15765_4_ADDR_ECU_7 = $07,
    PUDS_ISO_15765_4_ADDR_ECU_8 = $08
);
```

C#

```
public enum TPUDSAddress : byte
{
    PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT = 0xF1,
    PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL = 0x33,
    PUDS_ISO_15765_4_ADDR_ECU_1 = 0x01,
    PUDS_ISO_15765_4_ADDR_ECU_2 = 0x02,
    PUDS_ISO_15765_4_ADDR_ECU_3 = 0x03,
    PUDS_ISO_15765_4_ADDR_ECU_4 = 0x04,
    PUDS_ISO_15765_4_ADDR_ECU_5 = 0x05,
```

```
    PUDS_ISO_15765_4_ADDR_ECU_6 = 0x06,
    PUDS_ISO_15765_4_ADDR_ECU_7 = 0x07,
    PUDS_ISO_15765_4_ADDR_ECU_8 = 0x08,
}
```

C++ / CLR

```
public enum TPUDSAddress : Byte
{
      PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT = 0xF1,
      PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL = 0x33,
      PUDS_ISO_15765_4_ADDR_ECU_1 = 0x01,
      PUDS_ISO_15765_4_ADDR_ECU_2 = 0x02,
      PUDS_ISO_15765_4_ADDR_ECU_3 = 0x03,
      PUDS_ISO_15765_4_ADDR_ECU_4 = 0x04,
      PUDS_ISO_15765_4_ADDR_ECU_5 = 0x05,
      PUDS_ISO_15765_4_ADDR_ECU_6 = 0x06,
      PUDS_ISO_15765_4_ADDR_ECU_7 = 0x07,
      PUDS_ISO_15765_4_ADDR_ECU_8 = 0x08,
};
```

Visual Basic

```
Public Enum TPUDSAddress As Byte
    PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT = &HF1
    PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL = &H33
    PUDS_ISO_15765_4_ADDR_ECU_1 = &H1
    PUDS_ISO_15765_4_ADDR_ECU_2 = &H2
    PUDS_ISO_15765_4_ADDR_ECU_3 = &H3
    PUDS_ISO_15765_4_ADDR_ECU_4 = &H4
    PUDS_ISO_15765_4_ADDR_ECU_5 = &H5
    PUDS_ISO_15765_4_ADDR_ECU_6 = &H6
    PUDS_ISO_15765_4_ADDR_ECU_7 = &H7
    PUDS_ISO_15765_4_ADDR_ECU_8 = &H8
End Enum
```

Values

Name	Value	Description
PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT	0xF1 (241)	Legislated physical address for external equipment
PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL	0x33 (51)	Functional address for legislated OBD system
PUDS_ISO_15765_4_ADDR_ECU_1	0x01 (1)	Legislated-OBD ECU #1
PUDS_ISO_15765_4_ADDR_ECU_2	0x02 (2)	Legislated-OBD ECU #2
PUDS_ISO_15765_4_ADDR_ECU_3	0x03 (3)	Legislated-OBD ECU #3
PUDS_ISO_15765_4_ADDR_ECU_4	0x04 (4)	Legislated-OBD ECU #4
PUDS_ISO_15765_4_ADDR_ECU_5	0x05 (5)	Legislated-OBD ECU #5
PUDS_ISO_15765_4_ADDR_ECU_6	0x06 (6)	Legislated-OBD ECU #6
PUDS_ISO_15765_4_ADDR_ECU_7	0x07 (7)	Legislated-OBD ECU #7
PUDS_ISO_15765_4_ADDR_ECU_8	0x08 (8)	Legislated-OBD ECU #8

3.5.9 TPUDSCanId

Represents the legislated CAN Identifiers used with OBD (ISO-15765-4) communication. Each of these CAN IDs has a specific network addressing associated to.

Syntax

C++

```
#define PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST          0x7DF
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1          0x7E0
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1         0x7E8
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2          0x7E1
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2         0x7E9
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3          0x7E2
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3         0x7EA
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4          0x7E3
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4         0x7EB
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5          0x7E4
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5         0x7EC
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6          0x7E5
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6         0x7ED
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7          0x7E6
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7         0x7EE
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8          0x7E7
#define PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8         0x7EF
```

Pascal OO

```
TPUDSCanId = (
    PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST = $7DF,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1 = $7E0,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1 = $7E8,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2 = $7E1,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2 = $7E9,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3 = $7E2,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3 = $7EA,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4 = $7E3,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4 = $7EB,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5 = $7E4,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5 = $7EC,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6 = $7E5,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6 = $7ED,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7 = $7E6,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7 = $7EE,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8 = $7E7,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8 = $7EF
);
```

C#

```
public enum TPUDSCanId : uint
{
    PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST = 0x7DF,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1 = 0x7E0,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1 = 0x7E8,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2 = 0x7E1,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2 = 0x7E9,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3 = 0x7E2,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3 = 0x7EA,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4 = 0x7E3,
```

```
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4 = 0x7EB,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5 = 0x7E4,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5 = 0x7EC,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6 = 0x7E5,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6 = 0x7ED,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7 = 0x7E6,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7 = 0x7EE,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8 = 0x7E7,
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8 = 0x7EF,
}
```

C++ / CLR

```
public enum TPUDSCanId : UInt32
{
      PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST = 0x7DF,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1 = 0x7E0,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1 = 0x7E8,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2 = 0x7E1,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2 = 0x7E9,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3 = 0x7E2,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3 = 0x7EA,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4 = 0x7E3,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4 = 0x7EB,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5 = 0x7E4,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5 = 0x7EC,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6 = 0x7E5,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6 = 0x7ED,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7 = 0x7E6,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7 = 0x7EE,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8 = 0x7E7,
      PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8 = 0x7EF,
};
```

Visual Basic

```
Public Enum TPUDSCanId As UInt32
    PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST = &H7DF
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1 = &H7E0
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1 = &H7E8
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2 = &H7E1
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2 = &H7E9
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3 = &H7E2
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3 = &H7EA
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4 = &H7E3
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4 = &H7EB
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5 = &H7E4
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5 = &H7EC
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6 = &H7E5
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6 = &H7ED
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7 = &H7E6
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7 = &H7EE
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8 = &H7E7
    PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8 = &H7EF
End Enum
```

Values

Name	Value	Description
PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST	0x7DF	CAN identifier for functionally addressed request messsages sent by external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1	0x7E0	Physical request CAN ID from external test equipment to ECU #1
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1	0x7E8	Physical response CAN ID from ECU #1 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2	0x7E1	Physical request CAN ID from external test equipment to ECU #2
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2	0x7E9	Physical response CAN ID from ECU #2 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3	0x7E2	Physical request CAN ID from external test equipment to ECU #3
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3	0x7EA	Physical response CAN ID from ECU #3 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4	0x7E3	Physical request CAN ID from external test equipment to ECU #4
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4	0x7EB	Physical response CAN ID from ECU #4 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5	0x7E4	Physical request CAN ID from external test equipment to ECU #5
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5	0x7EC	Physical response CAN ID from ECU #5 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6	0x7E5	Physical request CAN ID from external test equipment to ECU #6
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6	0x7ED	Physical response CAN ID from ECU #6 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7	0x7E6	Physical request CAN ID from external test equipment to ECU #7
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7	0x7EE	Physical response CAN ID from ECU #7 to external test equipment
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8	0x7E7	Physical request CAN ID from external test equipment to ECU #8
PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8	0x7EF	Physical response CAN ID from ECU #8 to external test equipment

3.5.10 TPUDSProtocol

Represents the supported network layer protocol for UDS communication.

Syntax

C++

```
#define TPUDSProtocol BYTE

#define PUDS_PROTOCOL_NONE                      0x00
#define PUDS_PROTOCOL_ISO_15765_2_11B           0x01
#define PUDS_PROTOCOL_ISO_15765_2_11B_REMOTE    0x02
#define PUDS_PROTOCOL_ISO_15765_2_29B           0x03
#define PUDS_PROTOCOL_ISO_15765_2_29B_REMOTE    0x04
#define PUDS_PROTOCOL_ISO_15765_3_29B           0x05
#define PUDS_PROTOCOL_ISO_15765_2_29B_NORMAL    0x06
#define PUDS_PROTOCOL_ISO_15765_2_11B_EXTENDED  0x07
#define PUDS_PROTOCOL_ISO_15765_2_29B_EXTENDED  0x08
```

Pascal OO

```
TPUDSProtocol = (
    PUDS_PROTOCOL_NONE = $00,
    PUDS_PROTOCOL_ISO_15765_2_11B = $01,
    PUDS_PROTOCOL_ISO_15765_2_11B_REMOTE = $02,
    PUDS_PROTOCOL_ISO_15765_2_29B = $03,
    PUDS_PROTOCOL_ISO_15765_2_29B_REMOTE = $04,
    PUDS_PROTOCOL_ISO_15765_3_29B = $05,
    PUDS_PROTOCOL_ISO_15765_2_29B_NORMAL = $06,
    PUDS_PROTOCOL_ISO_15765_2_11B_EXTENDED = $07,
    PUDS_PROTOCOL_ISO_15765_2_29B_EXTENDED = $08
);
```

C#

```
public enum TPUDSProtocol : byte
{
    PUDS_PROTOCOL_NONE = 0x00,
    PUDS_PROTOCOL_ISO_15765_2_11B = 0x01,
    PUDS_PROTOCOL_ISO_15765_2_11B_REMOTE = 0x02,
    PUDS_PROTOCOL_ISO_15765_2_29B = 0x03,
    PUDS_PROTOCOL_ISO_15765_2_29B_REMOTE = 0x04,
    PUDS_PROTOCOL_ISO_15765_3_29B = 0x05,
    PUDS_PROTOCOL_ISO_15765_2_29B_NORMAL = 0x06,
    PUDS_PROTOCOL_ISO_15765_2_11B_EXTENDED = 0x07,
    PUDS_PROTOCOL_ISO_15765_2_29B_EXTENDED = 0x08,
}
```

C++ / CLR

```
public enum TPUDSProtocol : Byte
{
      PUDS_PROTOCOL_NONE = 0x00,
      PUDS_PROTOCOL_ISO_15765_2_11B = 0x01,
      PUDS_PROTOCOL_ISO_15765_2_11B_REMOTE = 0x02,
      PUDS_PROTOCOL_ISO_15765_2_29B = 0x03,
      PUDS_PROTOCOL_ISO_15765_2_29B_REMOTE = 0x04,
      PUDS_PROTOCOL_ISO_15765_3_29B = 0x05,
      PUDS_PROTOCOL_ISO_15765_2_29B_NORMAL = 0x06,
      PUDS_PROTOCOL_ISO_15765_2_11B_EXTENDED = 0x07,
      PUDS_PROTOCOL_ISO_15765_2_29B_EXTENDED = 0x08,
};
```

Visual Basic

```
Public Enum TPUDSProtocol As Byte
    PUDS_PROTOCOL_NONE = &H0
    PUDS_PROTOCOL_ISO_15765_2_11B = &H1
    PUDS_PROTOCOL_ISO_15765_2_11B_REMOTE = &H2
    PUDS_PROTOCOL_ISO_15765_2_29B = &H3
    PUDS_PROTOCOL_ISO_15765_2_29B_REMOTE = &H4
    PUDS_PROTOCOL_ISO_15765_3_29B = &H5
    PUDS_PROTOCOL_ISO_15765_2_29B_NORMAL = &H6
    PUDS_PROTOCOL_ISO_15765_2_11B_EXTENDED = &H7
    PUDS_PROTOCOL_ISO_15765_2_29B_EXTENDED = &H8
End Enum
```

Values

Name	Value	Description
PUDS_PROTOCOL_NONE	0	Network layer configuration for Unacknowlegde Unsegmented Data Transfer
PUDS_PROTOCOL_ISO_15765_2_11B	1	Network layer configuration for the ISO-TP API: 11 BIT CAN ID, NORMAL addressing and diagnostic message type
PUDS_PROTOCOL_ISO_15765_2_11B_REMOTE	2	Network layer configuration for the ISO-TP API: 11 BIT CAN ID, MIXED addressing and remote diagnostic message type
PUDS_PROTOCOL_ISO_15765_2_29B	3	Network layer configuration for the ISO-TP API: 29 BIT CAN ID, FIXED NORMAL addressing and diagnostic message type
PUDS_PROTOCOL_ISO_15765_2_29B_REMOTE	4	Network layer configuration for the ISO-TP API: 29 BIT CAN ID, MIXED addressing and remote diagnostic message type
PUDS_PROTOCOL_ISO_15765_3_29B	5	Network layer configuration for the ISO-TP API: Enhanced diagnostics 29 bit CAN Identifiers

Name	Value	Description
PUDS_PROTOCOL_ISO_15765_2_29B_NORMAL	6	Network layer configuration for the ISO-TP API: 29 BIT CAN ID, NORMAL addressing and diagnostic message type. Note that specific CAN ID mappings must be configured via the CAN-ISO-TP API to transmit and receive such messages
PUDS_PROTOCOL_ISO_15765_2_11B_EXTENDED	7	Network layer configuration for the ISO-TP API: 29 BIT CAN ID, MIXED addressing and remote diagnostic message type. Note that specific CAN ID mappings must be configured via the CAN-ISO-TP API to transmit and receive such messages
PUDS_PROTOCOL_ISO_15765_3_29B_EXTENDED	8	Network layer configuration for the ISO-TP API: Enhanced diagnostics 29 bit CAN Identifiers type. Note that specific CAN ID mappings must be configured via the CAN-ISO-TP API to transmit and receive such messages

See also: UDS and ISO-TP Network Addressing Information on page 339, UUDT Read/Write example p. 344.

3.5.11 TPUDSAddressingType

Represents the format addressing type of UDS on CAN messages (using ISO-TP API).

Syntax

C++

```
#define TPUDSAddressingType BYTE

#define PUDS_ADDRESSING_PHYSICAL 0x01
#define PUDS_ADDRESSING_FUNCTIONAL 0x02
```

Pascal OO

```
{$Z1}
TPUDSAddressingType = (
  PUDS_ADDRESSING_PHYSICAL = $01,
  PUDS_ADDRESSING_FUNCTIONAL = $02,
);
```

C#

```
public enum TPUDSAddressingType : byte
{
  PUDS_ADDRESSING_PHYSICAL = 0x01,
  PUDS_ADDRESSING_FUNCTIONAL = 0x02,
}
```

C++ / CLR

```
public enum class TPUDSAddressingType : Byte
{
  PUDS_ADDRESSING_UNKNOWN = 0x00,
  PUDS_ADDRESSING_PHYSICAL = 0x01,
  PUDS_ADDRESSING_FUNCTIONAL = 0x02,
};
```

Visual Basic

```
Public Enum TPUDSAddressingType As Byte
  PUDS_ADDRESSING_PHYSICAL = &H01
  PUDS_ADDRESSING_FUNCTIONAL = &H02
End Enum
```

Values

Name	Value	Description
PUDS_ADDRESSING_PHYSICAL	1	Physical addressing (used for communication between 2 nodes)
PUDS_ADDRESSING_FUNCTIONAL	2	Functional addressing (used to broadcast messages on the CAN bus)

Remarks: When using functionally addressed messages (`PUDS_ADDRESSING_FUNCTIONAL`), keep in mind that the UDS message must fit in a single CAN frame:

- the length of the DATA plus the ISO-TP header must be equal or less than 8 bytes,
- the ISO-TP header varies from 1 to 2 bytes depending on the network addressing format (MIXED and EXTENDED addressing will use 2 bytes)

See also: UDS and ISO-TP Network Addressing Information on page 339.

3.5.12 TPUDSMessageType

Represents the type of UDS messages.

Syntax

C++

```
#define TPUDSMessageType BYTE

#define PUDS_MESSAGE_TYPE_REQUEST 0x00
#define PUDS_MESSAGE_TYPE_CONFIRM 0x01
#define PUDS_MESSAGE_TYPE_INDICATION 0x02
#define PUDS_MESSAGE_TYPE_INDICATION_TX 0x03
#define PUDS_MESSAGE_TYPE_CONFIRM_UUDT 0x04
```

Pascal OO

```
{$Z1}
TPUDSMessageType = (
  PUDS_MESSAGE_TYPE_REQUEST = $00,
  PUDS_MESSAGE_TYPE_CONFIRM = $01,
  PUDS_MESSAGE_TYPE_INDICATION = $02,
  PUDS_MESSAGE_TYPE_INDICATION_TX = $03,
  PUDS_MESSAGE_TYPE_CONFIRM_UUDT = $04,
);
```

C#

```
public enum TPUDSMessageType : byte
{
  PUDS_MESSAGE_TYPE_REQUEST = 0x00,
```

```
  PUDS_MESSAGE_TYPE_CONFIRM = 0x01,
  PUDS_MESSAGE_TYPE_INDICATION = 0x02,
  PUDS_MESSAGE_TYPE_INDICATION_TX = 0x03,
  PUDS_MESSAGE_TYPE_CONFIRM_UUDT = 0x04,
}
```

C++ / CLR

```
public enum class TPUDSMessageType : Byte
{
  PUDS_MESSAGE_TYPE_REQUEST = 0x00,
  PUDS_MESSAGE_TYPE_CONFIRM = 0x01,
  PUDS_MESSAGE_TYPE_INDICATION = 0x02,
  PUDS_MESSAGE_TYPE_INDICATION_TX = 0x03,
  PUDS_MESSAGE_TYPE_CONFIRM_UUDT = 0x04,
};
```

Visual Basic

```
Public Enum TPUDSMessageType As Byte
  PUDS_MESSAGE_TYPE_REQUEST = 0x00,
  PUDS_MESSAGE_TYPE_CONFIRM = &H01,
  PUDS_MESSAGE_TYPE_INDICATION = &H02
  PUDS_MESSAGE_TYPE_INDICATION_TX = &H03
  PUDS_MESSAGE_TYPE_CONFIRM_UUDT = &H04
End Enum
```

Values

Name	Value	Description
PUDS_MESSAGE_TYPE_REQUEST	0	UDS Request message.
PUDS_MESSAGE_TYPE_CONFIRM	1	UDS Request or Response confirmation message.
PUDS_MESSAGE_TYPE_INDICATION	2	Incoming UDS message.
PUDS_MESSAGE_TYPE_INDICATION_TX	3	UDS message transmission started.
PUDS_MESSAGE_TYPE_CONFIRM_UUDT	4	Unacknowledge Unsegmented Data Transfert (UUDT) response.

3.5.13 TPUDSSvcParamDSC

Represents the subfunction parameter for UDS service DiagnosticSessionControl.

Syntax

C++

```
#define PUDS_SV_PARAM_DSC_DS              0x01
#define PUDS_SV_PARAM_DSC _ECUPS    0x02
#define PUDS_SV_PARAM_DSC_ECUEDS    0x03
#define PUDS_SV_PARAM_DSC _SSDS           0x04
```

Pascal OO

```
{$Z1}
TPUDSSvcParamDSC = (
  PUDS_SVC_PARAM_DSC_DS = $01,
  PUDS_SVC_PARAM_DSC_ECUPS = $02,
  PUDS_SVC_PARAM_DSC_ECUEDS = $03,
  PUDS_SVC_PARAM_DSC_SSDS = $04
);
```

C#

```
public enum TPUDSSvcParamDSC: byte
{
  PUDS_SVC_PARAM_DSC_DS = 0x01,
  PUDS_SVC_PARAM_DSC_ECUPS = 0x02,
  PUDS_SVC_PARAM_DSC_ECUEDS = 0x03,
  PUDS_SVC_PARAM_DSC_SSDS = 0x04,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamDSC: Byte
{
      PUDS_SVC_PARAM_DSC_DS = 0x01,
      PUDS_SVC_PARAM_DSC_ECUPS = 0x02,
      PUDS_SVC_PARAM_DSC_ECUEDS = 0x03,
      PUDS_SVC_PARAM_DSC_SSDS = 0x04
};
```

Visual Basic Syntax

```
Public Enum TPUDSSvcParamDSC As Byte
    PUDS_SVC_PARAM_DSC_DS = &H1
    PUDS_SVC_PARAM_DSC_ECUPS = &H2
    PUDS_SVC_PARAM_DSC_ECUEDS = &H3
    PUDS_SVC_PARAM_DSC_SSDS = &H4
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_DSC_DS	1	Default Session
PUDS_SVC_PARAM_DSC_ECUPS	2	ECU Programming Session
PUDS_SVC_PARAM_DSC_ECUEDS	3	ECU Extended Diagnostic Session
PUDS_SVC_PARAM_DSC_SSDS	4	Safety System Diagnostic Session

See also: PUDS_SvcDiagnosticSessionControl (**class-method:** SvcDiagnosticSessionControl).

3.5.14 TPUDSSvcParamER

Represents the subfunction parameter for UDS service ECUReset.

Syntax

C++

```
#define PUDS_SVC_PARAM_ER_HR              0x01
#define PUDS_SVC_PARAM_ER_KOFFONR         0x02
#define PUDS_SVC_PARAM_ER_SR              0x03
#define PUDS_SVC_PARAM_ER_ERPSD           0x04
#define PUDS_SVC_PARAM_ER_DRPSD           0x05
```

Pascal OO

```
TPUDSSvcParamER = (
    PUDS_SVC_PARAM_ER_HR = $01,
    PUDS_SVC_PARAM_ER_KOFFONR = $02,
    PUDS_SVC_PARAM_ER_SR = $03,
    PUDS_SVC_PARAM_ER_ERPSD = $04,
    PUDS_SVC_PARAM_ER_DRPSD = $05
);
```

C#

```
public enum TPUDSSvcParamER : byte
{
    PUDS_SVC_PARAM_ER_HR = 0x01,
    PUDS_SVC_PARAM_ER_KOFFONR = 0x02,
    PUDS_SVC_PARAM_ER_SR = 0x03,
    PUDS_SVC_PARAM_ER_ERPSD = 0x04,
    PUDS_SVC_PARAM_ER_DRPSD = 0x05,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamER : Byte
{
      PUDS_SVC_PARAM_ER_HR = 0x01,
      PUDS_SVC_PARAM_ER_KOFFONR = 0x02,
      PUDS_SVC_PARAM_ER_SR = 0x03,
      PUDS_SVC_PARAM_ER_ERPSD = 0x04,
      PUDS_SVC_PARAM_ER_DRPSD = 0x05,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamER As Byte
    PUDS_SVC_PARAM_ER_HR = &H1
    PUDS_SVC_PARAM_ER_KOFFONR = &H2
    PUDS_SVC_PARAM_ER_SR = &H3
    PUDS_SVC_PARAM_ER_ERPSD = &H4
    PUDS_SVC_PARAM_ER_DRPSD = &H5
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_ER_HR	1	Hard Reset
PUDS_SVC_PARAM_ER_KOFFONR	2	Key Off on Reset
PUDS_SVC_PARAM_ER_SR	3	Soft Reset
PUDS_SVC_PARAM_ER_ERPSD	4	Enable Rapid Power Shutdown

Name	Value	Description
PUDS_SVC_PARAM_ER_DRPSD	5	Disable Rapid Power Shutdown

See also: PUDS_SvcECUReset (**class-method:** SvcECUReset).

3.5.15 TPUDSSvcParamCC

Represents the subfunction parameter for UDS service ComunicationControl.

Syntax

C++

```
#define PUDS_SVC_PARAM_CC_ERXTX     0x00
#define PUDS_SVC_PARAM_CC_ERXDTX    0x01
#define PUDS_SVC_PARAM_CC_DRXETX    0x02
#define PUDS_SVC_PARAM_CC_DRXTX     0x03
```

Pascal OO

```
TPUDSSvcParamCC = (
    PUDS_SVC_PARAM_CC_ERXTX = $00,
    PUDS_SVC_PARAM_CC_ERXDTX = $01,
    PUDS_SVC_PARAM_CC_DRXETX = $02,
    PUDS_SVC_PARAM_CC_DRXTX = $03
);
```

C#

```
public enum TPUDSSvcParamCC : byte
{
    PUDS_SVC_PARAM_CC_ERXTX = 0x00,
    PUDS_SVC_PARAM_CC_ERXDTX = 0x01,
    PUDS_SVC_PARAM_CC_DRXETX = 0x02,
    PUDS_SVC_PARAM_CC_DRXTX = 0x03,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamCC : Byte
{
      PUDS_SVC_PARAM_CC_ERXTX = 0x00,
      PUDS_SVC_PARAM_CC_ERXDTX = 0x01,
      PUDS_SVC_PARAM_CC_DRXETX = 0x02,
      PUDS_SVC_PARAM_CC_DRXTX = 0x03,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamCC As Byte
    PUDS_SVC_PARAM_CC_ERXTX = &H0
    PUDS_SVC_PARAM_CC_ERXDTX = &H1
    PUDS_SVC_PARAM_CC_DRXETX = &H2
    PUDS_SVC_PARAM_CC_DRXTX = &H3
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_CC_ERXTX	0	Enable Rx and Tx
PUDS_SVC_PARAM_CC_ERXDTX	1	Enable Rx and Disable Tx
PUDS_SVC_PARAM_CC_DRXETX	2	Disable Rx and Enable Tx
PUDS_SVC_PARAM_CC_DRXTX	3	Disable Rx and Tx

See also: PUDS_SvcCommunicationControl (**class-method:** SvcCommunicationControl).

3.5.16 TPUDSSvcParamTP

Represents the subfunction parameter for UDS service TesterPresent.

Syntax

C++

```
#define PUDS_SVC_PARAM_TP_ZSUBF                 0x00
```

Pascal OO

```
TPUDSSvcParamTP = (
    PUDS_SVC_PARAM_TP_ZSUBF = $00
);
```

C#

```
public enum TPUDSSvcParamTP : byte
{
    PUDS_SVC_PARAM_TP_ZSUBF = 0x00,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamTP : Byte
{
      PUDS_SVC_PARAM_TP_ZSUBF = 0x00,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamTP As Byte
    PUDS_SVC_PARAM_TP_ZSUBF = &H0
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_TP_ZSUBF	0	Zero subfunction

See also: PUDS_SvcTesterPresent (**class-method:** SvcTesterPresent).

3.5.17 TPUDSSvcParamCDTCS

Represents the subfunction parameter for UDS service ControlDTCSetting.

Syntax

C++

```
#define PUDS_SVC_PARAM_CDTCS_ON          0x01
#define PUDS_SVC_PARAM_CDTCS_OFF         0x02
```

Pascal OO

```
TPUDSSvcParamCDTCS = (
    PUDS_SVC_PARAM_CDTCS_ON = $01,
    PUDS_SVC_PARAM_CDTCS_OFF = $02
);
```

C#

```
public enum TPUDSSvcParamCDTCS : byte
{
    PUDS_SVC_PARAM_CDTCS_ON = 0x01,
    PUDS_SVC_PARAM_CDTCS_OFF = 0x02,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamCDTCS : Byte
{
      PUDS_SVC_PARAM_CDTCS_ON = 0x01,
      PUDS_SVC_PARAM_CDTCS_OFF = 0x02,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamCDTCS As Byte
    PUDS_SVC_PARAM_CDTCS_ON = &H1
    PUDS_SVC_PARAM_CDTCS_OFF = &H2
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_CDTCS_ON	1	The server(s) shall resume the setting of diagnostic trouble codes
PUDS_SVC_PARAM_CDTCS_OFF	2	The server(s) shall stop the setting of diagnostic trouble codes

See also: PUDS_SvcControlDTCSetting (**class-method:** SvcControlDTCSetting).

3.5.18 TPUDSSvcParamROE

Represents the subfunction parameter for UDS service ControlDTCSetting.

Syntax

C++

```
#define PUDS_SVC_PARAM_ROE_STPROE        0x00
#define PUDS_SVC_PARAM_ROE_ONDTCS        0x01
#define PUDS_SVC_PARAM_ROE_OTI           0x02
#define PUDS_SVC_PARAM_ROE_OCODID        0x03
```

```
#define PUDS_SVC_PARAM_ROE_RAE              0x04
#define PUDS_SVC_PARAM_ROE_STRTROE          0x05
#define PUDS_SVC_PARAM_ROE_CLRROE           0x06
#define PUDS_SVC_PARAM_ROE_OCOV             0x07
```

Pascal OO

```
TPUDSSvcParamROE = (
    PUDS_SVC_PARAM_ROE_STPROE = $00,
    PUDS_SVC_PARAM_ROE_ONDTCS = $01,
    PUDS_SVC_PARAM_ROE_OTI = $02,
    PUDS_SVC_PARAM_ROE_OCODID = $03,
    PUDS_SVC_PARAM_ROE_RAE = $04,
    PUDS_SVC_PARAM_ROE_STRTROE = $05,
    PUDS_SVC_PARAM_ROE_CLRROE = $06,
    PUDS_SVC_PARAM_ROE_OCOV = $07
);
```

C#

```
public enum TPUDSSvcParamROE : byte
{
    PUDS_SVC_PARAM_ROE_STPROE = 0x00,
    PUDS_SVC_PARAM_ROE_ONDTCS = 0x01,
    PUDS_SVC_PARAM_ROE_OTI = 0x02,
    PUDS_SVC_PARAM_ROE_OCODID = 0x03,
    PUDS_SVC_PARAM_ROE_RAE = 0x04,
    PUDS_SVC_PARAM_ROE_STRTROE = 0x05,
    PUDS_SVC_PARAM_ROE_CLRROE = 0x06,
    PUDS_SVC_PARAM_ROE_OCOV = 0x07,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamROE : Byte
{
      PUDS_SVC_PARAM_ROE_STPROE = 0x00,
      PUDS_SVC_PARAM_ROE_ONDTCS = 0x01,
      PUDS_SVC_PARAM_ROE_OTI = 0x02,
      PUDS_SVC_PARAM_ROE_OCODID = 0x03,
      PUDS_SVC_PARAM_ROE_RAE = 0x04,
      PUDS_SVC_PARAM_ROE_STRTROE = 0x05,
      PUDS_SVC_PARAM_ROE_CLRROE = 0x06,
      PUDS_SVC_PARAM_ROE_OCOV = 0x07,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamROE As Byte
    PUDS_SVC_PARAM_ROE_STPROE = &H0
    PUDS_SVC_PARAM_ROE_ONDTCS = &H1
    PUDS_SVC_PARAM_ROE_OTI = &H2
    PUDS_SVC_PARAM_ROE_OCODID = &H3
    PUDS_SVC_PARAM_ROE_RAE = &H4
    PUDS_SVC_PARAM_ROE_STRTROE = &H5
    PUDS_SVC_PARAM_ROE_CLRROE = &H6
    PUDS_SVC_PARAM_ROE_OCOV = &H7
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_ROE_STPROE	0	Stop Response On Event
PUDS_SVC_PARAM_ROE_ONDTCS	1	On DTC Status Change
PUDS_SVC_PARAM_ROE_OTI	2	On Timer Interrupt
PUDS_SVC_PARAM_ROE_OCODID	3	On Change Of Data Identifier
PUDS_SVC_PARAM_ROE_RAE	4	Report Activated Events
PUDS_SVC_PARAM_ROE_STRTROE	5	Start Response On Event
PUDS_SVC_PARAM_ROE_CLRROE	6	Clear Response On Event
PUDS_SVC_PARAM_ROE_OCOV	7	On Comparison Of Values

See also: PUDS_SvcResponseOnEvent (**class-method:** SvcResponseOnEvent).

3.5.19 TPUDSSvcParamROERecommendedServiceID

Represents the recommended service to use with the UDS service ResponseOnEvent.

Syntax

C++

```
#define PUDS_SVC_PARAM_ROE_STRT_SI_RDBI   PUDS_SI_ReadDataByIdentifier
#define PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI  PUDS_SI_ReadDTCInformation
#define PUDS_SVC_PARAM_ROE_STRT_SI_RC     PUDS_SI_RoutineControl
#define PUDS_SVC_PARAM_ROE_STRT_SI_IOCBI  PUDS_SI_InputOutputControlByIdentifier
```

Pascal OO

```
TPUDSSvcParamROERecommendedServiceID = (
    PUDS_SVC_PARAM_ROE_STRT_SI_RDBI = 34,
    PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI = 25,
    PUDS_SVC_PARAM_ROE_STRT_SI_RC = 49,
    PUDS_SVC_PARAM_ROE_STRT_SI_IOCBI = 47
);
```

C#

```
public enum TPUDSSvcParamROERecommendedServiceID : byte
{
    PUDS_SVC_PARAM_ROE_STRT_SI_RDBI = TPUDSService.PUDS_SI_ReadDataByIdentifier,
    PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI = TPUDSService.PUDS_SI_ReadDTCInformation,
    PUDS_SVC_PARAM_ROE_STRT_SI_RC = TPUDSService.PUDS_SI_RoutineControl,
    PUDS_SVC_PARAM_ROE_STRT_SI_IOCBI =
TPUDSService.PUDS_SI_InputOutputControlByIdentifier,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamROERecommendedServiceID : Byte
{
      PUDS_SVC_PARAM_ROE_STRT_SI_RDBI = PUDS_SI_ReadDataByIdentifier,
      PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI = PUDS_SI_ReadDTCInformation,
      PUDS_SVC_PARAM_ROE_STRT_SI_RC = PUDS_SI_RoutineControl,
      PUDS_SVC_PARAM_ROE_STRT_SI_IOCBI = PUDS_SI_InputOutputControlByIdentifier,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamROERecommendedServiceID As Byte
    PUDS_SVC_PARAM_ROE_STRT_SI_RDBI = TPUDSService.PUDS_SI_ReadDataByIdentifier
    PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI = TPUDSService.PUDS_SI_ReadDTCInformation
    PUDS_SVC_PARAM_ROE_STRT_SI_RC = TPUDSService.PUDS_SI_RoutineControl
    PUDS_SVC_PARAM_ROE_STRT_SI_IOCBI =
TPUDSService.PUDS_SI_InputOutputControlByIdentifier
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_ROE_STRT_SI_RDBI	PUDS_SI_ReadDataByIdentifier	UDS service ReadDataByIdentifier
PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI	PUDS_SI_ReadDTCInformation	UDS service ReadDTCInformation
PUDS_SVC_PARAM_ROE_STRT_SI_RC	PUDS_SI_RoutineControl	UDS service RoutineControl
PUDS_SVC_PARAM_ROE_STRT_SI_IOCBI	PUDS_SI_InputOutputControlByIdentifier	UDS service InputOutputControlByIdentifier

See also: PUDS_SvcResponseOnEvent (**class-method:** SvcResponseOnEvent).

3.5.20 TPUDSSvcParamLC

Represents the subfunction parameter for UDS service LinkControl.

Syntax

C++

```
#define PUDS_SVC_PARAM_LC_VBTWFBR                    0x01
#define PUDS_SVC_PARAM_LC_VBTWSBR                    0x02
#define PUDS_SVC_PARAM_LC_TB                         0x03
```

Pascal OO

```
TPUDSSvcParamLC = (
    PUDS_SVC_PARAM_LC_VBTWFBR = $01,
    PUDS_SVC_PARAM_LC_VBTWSBR = $02,
    PUDS_SVC_PARAM_LC_TB = $03
);
```

C#

```
public enum TPUDSSvcParamLC : byte
{
    PUDS_SVC_PARAM_LC_VBTWFBR = 0x01,
    PUDS_SVC_PARAM_LC_VBTWSBR = 0x02,
    PUDS_SVC_PARAM_LC_TB = 0x03,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamLC : Byte
{
      PUDS_SVC_PARAM_LC_VBTWFBR = 0x01,
      PUDS_SVC_PARAM_LC_VBTWSBR = 0x02,
      PUDS_SVC_PARAM_LC_TB = 0x03,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamLC As Byte
    PUDS_SVC_PARAM_LC_VBTWFBR = &H1
    PUDS_SVC_PARAM_LC_VBTWSBR = &H2
    PUDS_SVC_PARAM_LC_TB = &H3
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_LC_VBTWFBR	1	Verify Baudrate Transition With Fixed Baudrate
PUDS_SVC_PARAM_LC_VBTWSBR	2	Verify Baudrate Transition With Specific Baudrate
PUDS_SVC_PARAM_LC_TB	3	Transition Baudrate

See also: PUDS_SvcLinkControl (**class-method:** SvcLinkControl).

3.5.21 TPUDSSvcParamLCBaudrateIdentifier

Represents the standard Baudrate Identifiers for use with the UDS service LinkControl.

Syntax

C++

```
#define PUDS_SVC_PARAM_LC_BAUDRATE_PC_9600      0x01
#define PUDS_SVC_PARAM_LC_BAUDRATE_PC_19200     0x02
#define PUDS_SVC_PARAM_LC_BAUDRATE_PC_38400     0x03
#define PUDS_SVC_PARAM_LC_BAUDRATE_PC_57600     0x04
#define PUDS_SVC_PARAM_LC_BAUDRATE_PC_115200    0x05
#define PUDS_SVC_PARAM_LC_BAUDRATE_CAN_125K     0x10
#define PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K     0x11
#define PUDS_SVC_PARAM_LC_BAUDRATE_CAN_500K     0x12
#define PUDS_SVC_PARAM_LC_BAUDRATE_CAN_1M       0x13
```

Pascal OO

```
TPUDSSvcParamLCBaudrateIdentifier = (
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_9600    =     $01,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_19200   =     $02,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_38400   =     $03,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_57600   =     $04,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_115200 =      $05,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_125K   =     $10,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K   =     $11,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_500K   =     $12,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_1M     =     $13
);
```

C#

```
public enum TPUDSSvcParamLCBaudrateIdentifier : byte
{
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_9600    =     0x01,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_19200   =     0x02,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_38400   =     0x03,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_57600   =     0x04,
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_115200 =      0x05,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_125K   =     0x10,
```

```
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K   =     0x11,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_500K   =     0x12,
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_1M     =     0x13,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamLCBaudrateIdentifier : Byte
{
      PUDS_SVC_PARAM_LC_BAUDRATE_PC_9600  =     0x01,
      PUDS_SVC_PARAM_LC_BAUDRATE_PC_19200 =     0x02,
      PUDS_SVC_PARAM_LC_BAUDRATE_PC_38400 =     0x03,
      PUDS_SVC_PARAM_LC_BAUDRATE_PC_57600 =     0x04,
      PUDS_SVC_PARAM_LC_BAUDRATE_PC_115200 =    0x05,
      PUDS_SVC_PARAM_LC_BAUDRATE_CAN_125K =     0x10,
      PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K =     0x11,
      PUDS_SVC_PARAM_LC_BAUDRATE_CAN_500K =     0x12,
      PUDS_SVC_PARAM_LC_BAUDRATE_CAN_1M   =     0x13,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamLCBaudrateIdentifier As Byte
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_9600 = &H1
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_19200 = &H2
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_38400 = &H3
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_57600 = &H4
    PUDS_SVC_PARAM_LC_BAUDRATE_PC_115200 = &H5
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_125K = &H10
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K = &H11
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_500K = &H12
    PUDS_SVC_PARAM_LC_BAUDRATE_CAN_1M = &H13
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_LC_BAUDRATE_PC_9600	1	standard PC baud rate of 9.6 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_PC_19200	2	standard PC baud rate of 19.2 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_PC_38400	3	standard PC baud rate of 38.4 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_PC_57600	4	standard PC baud rate of 57.6 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_PC_115200	5	standard PC baud rate of 115.2 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_CAN_125K	0x10 (16)	standard CAN baud rate of 125 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K	0x11 (17)	standard CAN baud rate of 250 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_CAN_500K	0x12 (18)	standard CAN baud rate of 500 KBaud
PUDS_SVC_PARAM_LC_BAUDRATE_CAN_1M	0x13 (19)	standard CAN baud rate of 1 MBaud

See also: PUDS_SvcLinkControl (**class-method:** SvcLinkControl).

3.5.22 TPUDSSvcParamDI

Represents the Data Identifiers defined in UDS standard ISO-14229-1.

Syntax

C++

```
#define PUDS_SVC_PARAM_DI_BSIDID          0xF180
#define PUDS_SVC_PARAM_DI_ASIDID          0xF181
#define PUDS_SVC_PARAM_DI_ADIDID          0xF182
#define PUDS_SVC_PARAM_DI_BSFPDID         0xF183
#define PUDS_SVC_PARAM_DI_ASFPDID         0xF184
#define PUDS_SVC_PARAM_DI_ADFPDID         0xF185
#define PUDS_SVC_PARAM_DI_ADSDID          0xF186
#define PUDS_SVC_PARAM_DI_VMSPNDID        0xF187
#define PUDS_SVC_PARAM_DI_VMECUSNDID      0xF188
#define PUDS_SVC_PARAM_DI_VMECUSVNDID     0xF189
#define PUDS_SVC_PARAM_DI_SSIDDID         0xF18A
#define PUDS_SVC_PARAM_DI_ECUMDDID        0xF18B
#define PUDS_SVC_PARAM_DI_ECUSNDID        0xF18C
#define PUDS_SVC_PARAM_DI_SFUDID          0xF18D
#define PUDS_SVC_PARAM_DI_VMKAPNDID       0xF18E
#define PUDS_SVC_PARAM_DI_VINDID          0xF190
#define PUDS_SVC_PARAM_DI_VMECUHNDID      0xF191
#define PUDS_SVC_PARAM_DI_SSECUHWNDID     0xF192
#define PUDS_SVC_PARAM_DI_SSECUHWVNDID    0xF193
#define PUDS_SVC_PARAM_DI_SSECUSWNDID     0xF194
#define PUDS_SVC_PARAM_DI_SSECUSWVNDID    0xF195
#define PUDS_SVC_PARAM_DI_EROTANDID       0xF196
#define PUDS_SVC_PARAM_DI_SNOETDID        0xF197
#define PUDS_SVC_PARAM_DI_RSCOTSNDID      0xF198
#define PUDS_SVC_PARAM_DI_PDDID           0xF199
#define PUDS_SVC_PARAM_DI_CRSCOCESNDID    0xF19A
#define PUDS_SVC_PARAM_DI_CDDID           0xF19B
#define PUDS_SVC_PARAM_DI_CESWNDID        0xF19C
#define PUDS_SVC_PARAM_DI_EIDDID          0xF19D
#define PUDS_SVC_PARAM_DI_ODXFDID         0xF19E
#define PUDS_SVC_PARAM_DI_EDID            0xF19F
```

Pascal OO

```
TPUDSSvcParamDI= (
    PUDS_SVC_PARAM_DI_BSIDID = $F180,
    PUDS_SVC_PARAM_DI_ASIDID = $F181,
    PUDS_SVC_PARAM_DI_ADIDID = $F182,
    PUDS_SVC_PARAM_DI_BSFPDID = $F183,
    PUDS_SVC_PARAM_DI_ASFPDID = $F184,
    PUDS_SVC_PARAM_DI_ADFPDID = $F185,
    PUDS_SVC_PARAM_DI_ADSDID = $F186,
    PUDS_SVC_PARAM_DI_VMSPNDID = $F187,
    PUDS_SVC_PARAM_DI_VMECUSNDID = $F188,
    PUDS_SVC_PARAM_DI_VMECUSVNDID = $F189,
    PUDS_SVC_PARAM_DI_SSIDDID = $F18A,
    PUDS_SVC_PARAM_DI_ECUMDDID = $F18B,
    PUDS_SVC_PARAM_DI_ECUSNDID = $F18C,
    PUDS_SVC_PARAM_DI_SFUDID = $F18D,
    PUDS_SVC_PARAM_DI_VMKAPNDID = $F18E,
    PUDS_SVC_PARAM_DI_VINDID = $F190,
    PUDS_SVC_PARAM_DI_VMECUHNDID = $F191,
    PUDS_SVC_PARAM_DI_SSECUHWNDID = $F192,
    PUDS_SVC_PARAM_DI_SSECUHWVNDID = $F193,
    PUDS_SVC_PARAM_DI_SSECUSWNDID = $F194,
    PUDS_SVC_PARAM_DI_SSECUSWVNDID = $F195,
    PUDS_SVC_PARAM_DI_EROTANDID = $F196,
```

```
    PUDS_SVC_PARAM_DI_SNOETDID = $F197,
    PUDS_SVC_PARAM_DI_RSCOTSNDID = $F198,
    PUDS_SVC_PARAM_DI_PDDID = $F199,
    PUDS_SVC_PARAM_DI_CRSCOCESNDID = $F19A,
    PUDS_SVC_PARAM_DI_CDDID = $F19B,
    PUDS_SVC_PARAM_DI_CESWNDID = $F19C,
    PUDS_SVC_PARAM_DI_EIDDID = $F19D,
    PUDS_SVC_PARAM_DI_ODXFDID = $F19E,
    PUDS_SVC_PARAM_DI_EDID = $F19F
);
```

C#

```
public enum TPUDSSvcParamDI : ushort
{
    PUDS_SVC_PARAM_DI_BSIDID = 0xF180,
    PUDS_SVC_PARAM_DI_ASIDID = 0xF181,
    PUDS_SVC_PARAM_DI_ADIDID = 0xF182,
    PUDS_SVC_PARAM_DI_BSFPDID = 0xF183,
    PUDS_SVC_PARAM_DI_ASFPDID = 0xF184,
    PUDS_SVC_PARAM_DI_ADFPDID = 0xF185,
    PUDS_SVC_PARAM_DI_ADSDID = 0xF186,
    PUDS_SVC_PARAM_DI_VMSPNDID = 0xF187,
    PUDS_SVC_PARAM_DI_VMECUSNDID = 0xF188,
    PUDS_SVC_PARAM_DI_VMECUSVNDID = 0xF189,
    PUDS_SVC_PARAM_DI_SSIDDID = 0xF18A,
    PUDS_SVC_PARAM_DI_ECUMDDID = 0xF18B,
    PUDS_SVC_PARAM_DI_ECUSNDID = 0xF18C,
    PUDS_SVC_PARAM_DI_SFUDID = 0xF18D,
    PUDS_SVC_PARAM_DI_VMKAPNDID = 0xF18E,
    PUDS_SVC_PARAM_DI_VINDID = 0xF190,
    PUDS_SVC_PARAM_DI_VMECUHNDID = 0xF191,
    PUDS_SVC_PARAM_DI_SSECUHWNDID = 0xF192,
    PUDS_SVC_PARAM_DI_SSECUHWVNDID = 0xF193,
    PUDS_SVC_PARAM_DI_SSECUSWNDID = 0xF194,
    PUDS_SVC_PARAM_DI_SSECUSWVNDID = 0xF195,
    PUDS_SVC_PARAM_DI_EROTANDID = 0xF196,
    PUDS_SVC_PARAM_DI_SNOETDID = 0xF197,
    PUDS_SVC_PARAM_DI_RSCOTSNDID = 0xF198,
    PUDS_SVC_PARAM_DI_PDDID = 0xF199,
    PUDS_SVC_PARAM_DI_CRSCOCESNDID = 0xF19A,
    PUDS_SVC_PARAM_DI_CDDID = 0xF19B,
    PUDS_SVC_PARAM_DI_CESWNDID = 0xF19C,
    PUDS_SVC_PARAM_DI_EIDDID = 0xF19D,
    PUDS_SVC_PARAM_DI_ODXFDID = 0xF19E,
    PUDS_SVC_PARAM_DI_EDID = 0xF19F,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamDI : unsigned short
{
      PUDS_SVC_PARAM_DI_BSIDID = 0xF180,
      PUDS_SVC_PARAM_DI_ASIDID = 0xF181,
      PUDS_SVC_PARAM_DI_ADIDID = 0xF182,
      PUDS_SVC_PARAM_DI_BSFPDID = 0xF183,
      PUDS_SVC_PARAM_DI_ASFPDID = 0xF184,
      PUDS_SVC_PARAM_DI_ADFPDID = 0xF185,
      PUDS_SVC_PARAM_DI_ADSDID = 0xF186,
      PUDS_SVC_PARAM_DI_VMSPNDID = 0xF187,
      PUDS_SVC_PARAM_DI_VMECUSNDID = 0xF188,
```

```
      PUDS_SVC_PARAM_DI_VMECUSVNDID = 0xF189,
      PUDS_SVC_PARAM_DI_SSIDDID = 0xF18A,
      PUDS_SVC_PARAM_DI_ECUMDDID = 0xF18B,
      PUDS_SVC_PARAM_DI_ECUSNDID = 0xF18C,
      PUDS_SVC_PARAM_DI_SFUDID = 0xF18D,
      PUDS_SVC_PARAM_DI_VMKAPNDID = 0xF18E,
      PUDS_SVC_PARAM_DI_VINDID = 0xF190,
      PUDS_SVC_PARAM_DI_VMECUHNDID = 0xF191,
      PUDS_SVC_PARAM_DI_SSECUHWNDID = 0xF192,
      PUDS_SVC_PARAM_DI_SSECUHWVNDID = 0xF193,
      PUDS_SVC_PARAM_DI_SSECUSWNDID = 0xF194,
      PUDS_SVC_PARAM_DI_SSECUSWVNDID = 0xF195,
      PUDS_SVC_PARAM_DI_EROTANDID = 0xF196,
      PUDS_SVC_PARAM_DI_SNOETDID = 0xF197,
      PUDS_SVC_PARAM_DI_RSCOTSNDID = 0xF198,
      PUDS_SVC_PARAM_DI_PDDID = 0xF199,
      PUDS_SVC_PARAM_DI_CRSCOCESNDID = 0xF19A,
      PUDS_SVC_PARAM_DI_CDDID = 0xF19B,
      PUDS_SVC_PARAM_DI_CESWNDID = 0xF19C,
      PUDS_SVC_PARAM_DI_EIDDID = 0xF19D,
      PUDS_SVC_PARAM_DI_ODXFDID = 0xF19E,
      PUDS_SVC_PARAM_DI_EDID = 0xF19F,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamDI As UShort
    PUDS_SVC_PARAM_DI_BSIDID = &HF180
    PUDS_SVC_PARAM_DI_ASIDID = &HF181
    PUDS_SVC_PARAM_DI_ADIDID = &HF182
    PUDS_SVC_PARAM_DI_BSFPDID = &HF183
    PUDS_SVC_PARAM_DI_ASFPDID = &HF184
    PUDS_SVC_PARAM_DI_ADFPDID = &HF185
    PUDS_SVC_PARAM_DI_ADSDID = &HF186
    PUDS_SVC_PARAM_DI_VMSPNDID = &HF187
    PUDS_SVC_PARAM_DI_VMECUSNDID = &HF188
    PUDS_SVC_PARAM_DI_VMECUSVNDID = &HF189
    PUDS_SVC_PARAM_DI_SSIDDID = &HF18A
    PUDS_SVC_PARAM_DI_ECUMDDID = &HF18B
    PUDS_SVC_PARAM_DI_ECUSNDID = &HF18C
    PUDS_SVC_PARAM_DI_SFUDID = &HF18D
    PUDS_SVC_PARAM_DI_VMKAPNDID = &HF18E
    PUDS_SVC_PARAM_DI_VINDID = &HF190
    PUDS_SVC_PARAM_DI_VMECUHNDID = &HF191
    PUDS_SVC_PARAM_DI_SSECUHWNDID = &HF192
    PUDS_SVC_PARAM_DI_SSECUHWVNDID = &HF193
    PUDS_SVC_PARAM_DI_SSECUSWNDID = &HF194
    PUDS_SVC_PARAM_DI_SSECUSWVNDID = &HF195
    PUDS_SVC_PARAM_DI_EROTANDID = &HF196
    PUDS_SVC_PARAM_DI_SNOETDID = &HF197
    PUDS_SVC_PARAM_DI_RSCOTSNDID = &HF198
    PUDS_SVC_PARAM_DI_PDDID = &HF199
    PUDS_SVC_PARAM_DI_CRSCOCESNDID = &HF19A
    PUDS_SVC_PARAM_DI_CDDID = &HF19B
    PUDS_SVC_PARAM_DI_CESWNDID = &HF19C
    PUDS_SVC_PARAM_DI_EIDDID = &HF19D
    PUDS_SVC_PARAM_DI_ODXFDID = &HF19E
    PUDS_SVC_PARAM_DI_EDID = &HF19F
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_DI_BSIDID	0xF180	Boot Software Identification Data Identifier
PUDS_SVC_PARAM_DI_ASIDID	0xF181	Application Software Identification Data Identifier
PUDS_SVC_PARAM_DI_ADIDID	0xF182	Application Data Identification Data Identifier
PUDS_SVC_PARAM_DI_BSFPDID	0xF183	Boot Software Identification Data Identifier
PUDS_SVC_PARAM_DI_ASFPDID	0xF184	Application Software Fingerprint Data Identifier
PUDS_SVC_PARAM_DI_ADFPDID	0xF185	Application Data Fingerprint Data Identifier
PUDS_SVC_PARAM_DI_ADSDID	0xF186	Active Diagnostic Session Data Identifier
PUDS_SVC_PARAM_DI_VMSPNDID	0xF187	Vehicle Manufacturer Spare Part Number Data Identifier
PUDS_SVC_PARAM_DI_VMECUSNDID	0xF188	Vehicle Manufacturer ECU Software Number Data Identifier
PUDS_SVC_PARAM_DI_VMECUSVNDID	0xF189	Vehicle Manufacturer ECU Software Version Number Data Identifier
PUDS_SVC_PARAM_DI_SSIDDID	0xF18A	System Supplier Identifier Data Identifier
PUDS_SVC_PARAM_DI_ECUMDDID	0xF18B	ECU Manufacturing Date Data Identifier
PUDS_SVC_PARAM_DI_ECUSNDID	0xF18C	ECU Serial Number Data Identifier
PUDS_SVC_PARAM_DI_SFUDID	0xF18D	Supported Functional Units Data Identifier
PUDS_SVC_PARAM_DI_VMKAPNDID	0xF18E	Vehicle Manufacturer Kit Assembly Part Number Data Identifier
PUDS_SVC_PARAM_DI_VINDID	0xF190	VIN Data Identifier
PUDS_SVC_PARAM_DI_VMECUHNDID	0xF191	Vehicle Manufacturer ECU Hardware Number Data Identifier
PUDS_SVC_PARAM_DI_SSECUHWNDID	0xF192	System Supplier ECU Hardware Number Data Identifier
PUDS_SVC_PARAM_DI_SSECUHWVNDID	0xF193	System Supplier ECU Hardware Version Number Data Identifier
PUDS_SVC_PARAM_DI_SSECUSWNDID	0xF194	System Supplier ECU Software Number Data Identifier
PUDS_SVC_PARAM_DI_SSECUSWVNDID	0xF195	System Supplier ECU Software Version Number Data Identifier
PUDS_SVC_PARAM_DI_EROTANDID	0xF196	Exhaust Regulation Or Type Approval Number Data Identifier
PUDS_SVC_PARAM_DI_SNOETDID	0xF197	System Name Or Engine Type Data Identifier
PUDS_SVC_PARAM_DI_RSCOTSNDID	0xF198	Repair Shop Code Or Tester Serial Number Data Identifier
PUDS_SVC_PARAM_DI_PDDID	0xF199	Programming Date Data Identifier
PUDS_SVC_PARAM_DI_CRSCOCESNDID	0xF19A	Calibration Repair Shop Code Or Calibration Equipment Serial Number Data Identifier
PUDS_SVC_PARAM_DI_CDDID	0xF19B	calibrationDate Data Identifier
PUDS_SVC_PARAM_DI_CESWNDID	0xF19C	Calibration Equipment Software Number Data Identifier
PUDS_SVC_PARAM_DI_EIDDID	0xF19D	ECU Installation Date Data Identifier
PUDS_SVC_PARAM_DI_ODXFDID	0xF19E	ODX File Data Identifier
PUDS_SVC_PARAM_DI_EDID	0xF19F	Entity Data Identifier

See also: PUDS_SvcReadDataByIdentifier (**class-method:** SvcReadDataByIdentifier).

3.5.23 TPUDSSvcParamRDBPI

Represents the subfunction parameter for UDS service ReadDataByPeriodicIdentifier.

Syntax

C++

```
#define PUDS_SVC_PARAM_RDBPI_SASR          0x01
#define PUDS_SVC_PARAM_RDBPI_SAMR          0x02
#define PUDS_SVC_PARAM_RDBPI_SAFR          0x03
#define PUDS_SVC_PARAM_RDBPI_SS            0x04
```

Pascal OO

```
TPUDSSvcParamRDBPI = (
    PUDS_SVC_PARAM_RDBPI_SASR = $01,
    PUDS_SVC_PARAM_RDBPI_SAMR = $02,
    PUDS_SVC_PARAM_RDBPI_SAFR = $03,
    PUDS_SVC_PARAM_RDBPI_SS = $04
);
```

C#

```
public enum TPUDSSvcParamRDBPI : byte
{
    PUDS_SVC_PARAM_RDBPI_SASR = 0x01,
    PUDS_SVC_PARAM_RDBPI_SAMR = 0x02,
    PUDS_SVC_PARAM_RDBPI_SAFR = 0x03,
    PUDS_SVC_PARAM_RDBPI_SS = 0x04,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamRDBPI : Byte
{

      PUDS_SVC_PARAM_RDBPI_SASR = 0x01,
      PUDS_SVC_PARAM_RDBPI_SAMR = 0x02,
      PUDS_SVC_PARAM_RDBPI_SAFR = 0x03,
      PUDS_SVC_PARAM_RDBPI_SS = 0x04,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamRDBPI As Byte
    PUDS_SVC_PARAM_RDBPI_SASR = &H1
    PUDS_SVC_PARAM_RDBPI_SAMR = &H2
    PUDS_SVC_PARAM_RDBPI_SAFR = &H3
    PUDS_SVC_PARAM_RDBPI_SS = &H4
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_RDBPI_SASR	1	Send At Slow Rate
PUDS_SVC_PARAM_RDBPI_SAMR	2	Send At Medium Rate
PUDS_SVC_PARAM_RDBPI_SAFR	3	Send At Fast Rate
PUDS_SVC_PARAM_RDBPI_SS	4	Stop Sending

See also: PUDS_SvcReadDataByIdentifier (**class-method:** SvcReadDataByIdentifier).

3.5.24 TPUDSSvcParamDDDI

Represents the subfunction parameter for UDS service DynamicallyDefineDataIdentifier.

Syntax

C++

```
#define PUDS_SVC_PARAM_DDDI_DBID          0x01
#define PUDS_SVC_PARAM_DDDI_DBMA          0x02
```

```
#define PUDS_SVC_PARAM_DDDI_CDDDI          0x03
```

Pascal OO

```
TPUDSSvcParamDDDI = (
    PUDS_SVC_PARAM_DDDI_DBID = $01,
    PUDS_SVC_PARAM_DDDI_DBMA = $02,
    PUDS_SVC_PARAM_DDDI_CDDDI = $03
);
```

C#

```
public enum TPUDSSvcParamDDDI : byte
{
    PUDS_SVC_PARAM_DDDI_DBID = 0x01,
    PUDS_SVC_PARAM_DDDI_DBMA = 0x02,
    PUDS_SVC_PARAM_DDDI_CDDDI = 0x03,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamDDDI : Byte
{
      PUDS_SVC_PARAM_DDDI_DBID = 0x01,
      PUDS_SVC_PARAM_DDDI_DBMA = 0x02,
      PUDS_SVC_PARAM_DDDI_CDDDI = 0x03,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamDDDI As Byte
    PUDS_SVC_PARAM_DDDI_DBID = &H1
    PUDS_SVC_PARAM_DDDI_DBMA = &H2
    PUDS_SVC_PARAM_DDDI_CDDDI = &H3
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_DDDI_DBID	1	Define By Identifier
PUDS_SVC_PARAM_DDDI_DBMA	2	Define By Memory Address
PUDS_SVC_PARAM_DDDI_CDDDI	3	Clear Dynamically Defined Data Identifier

See also: PUDS_SvcDynamicallyDefineDataIdentifierDBID (**class-method:** SvcDynamicallyDefineDataIdentifierDBID), PUDS_SvcDynamicallyDefineDataIdentifierDBMA (**class-method:** SvcDynamicallyDefineDataIdentifierCDDDI), PUDS_SvcDynamicallyDefineDataIdentifierDBID (**class-method:** SvcDynamicallyDefineDataIdentifierCDDDI).

3.5.25 TPUDSSvcParamRDTCI

Represents the subfunction parameter for UDS service ReadDTCInformation.

Syntax

C++

```
#define PUDS_SVC_PARAM_RDTCI_RNODTCBSM          0x01
#define PUDS_SVC_PARAM_RDTCI_RDTCBSM            0x02
#define PUDS_SVC_PARAM_RDTCI_RDTCSSI            0x03
#define PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC         0x04
#define PUDS_SVC_PARAM_RDTCI_RDTCSSBRN          0x05
#define PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN         0x06
#define PUDS_SVC_PARAM_RDTCI_RNODTCBSMR         0x07
#define PUDS_SVC_PARAM_RDTCI_RDTCBSMR           0x08
#define PUDS_SVC_PARAM_RDTCI_RSIODTC            0x09
#define PUDS_SVC_PARAM_RDTCI_RSUPDTC            0x0A
#define PUDS_SVC_PARAM_RDTCI_RFTFDTC            0x0B
#define PUDS_SVC_PARAM_RDTCI_RFCDTC             0x0C
#define PUDS_SVC_PARAM_RDTCI_RMRTFDTC           0x0D
#define PUDS_SVC_PARAM_RDTCI_RMRCDTC            0x0E
#define PUDS_SVC_PARAM_RDTCI_RMMDTCBSM          0x0F
#define PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN         0x10
#define PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM        0x11
#define PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM       0x12
#define PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM         0x13
```

Pascal OO

```
TPUDSSvcParamRDTCI = (
    PUDS_SVC_PARAM_RDTCI_RNODTCBSM = $01,
    PUDS_SVC_PARAM_RDTCI_RDTCBSM = $02,
    PUDS_SVC_PARAM_RDTCI_RDTCSSI = $03,
    PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC = $04,
    PUDS_SVC_PARAM_RDTCI_RDTCSSBRN = $05,
    PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN = $06,
    PUDS_SVC_PARAM_RDTCI_RNODTCBSMR = $07,
    PUDS_SVC_PARAM_RDTCI_RDTCBSMR = $08,
    PUDS_SVC_PARAM_RDTCI_RSIODTC = $09,
    PUDS_SVC_PARAM_RDTCI_RSUPDTC = $0A,
    PUDS_SVC_PARAM_RDTCI_RFTFDTC = $0B,
    PUDS_SVC_PARAM_RDTCI_RFCDTC = $0C,
    PUDS_SVC_PARAM_RDTCI_RMRTFDTC = $0D,
    PUDS_SVC_PARAM_RDTCI_RMRCDTC = $0E,
    PUDS_SVC_PARAM_RDTCI_RMMDTCBSM = $0F,
    PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN = $10,
    PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM = $11,
    PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM = $12,
    PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM = $13
);
```

C#

```
public enum TPUDSSvcParamRDTCI : byte
{
    PUDS_SVC_PARAM_RDTCI_RNODTCBSM = 0x01,
    PUDS_SVC_PARAM_RDTCI_RDTCBSM = 0x02,
    PUDS_SVC_PARAM_RDTCI_RDTCSSI = 0x03,
    PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC = 0x04,
    PUDS_SVC_PARAM_RDTCI_RDTCSSBRN = 0x05,
    PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN = 0x06,
    PUDS_SVC_PARAM_RDTCI_RNODTCBSMR = 0x07,
    PUDS_SVC_PARAM_RDTCI_RDTCBSMR = 0x08,
    PUDS_SVC_PARAM_RDTCI_RSIODTC = 0x09,
```

```
    PUDS_SVC_PARAM_RDTCI_RSUPDTC = 0x0A,
    PUDS_SVC_PARAM_RDTCI_RFTFDTC = 0x0B,
    PUDS_SVC_PARAM_RDTCI_RFCDTC = 0x0C,
    PUDS_SVC_PARAM_RDTCI_RMRTFDTC = 0x0D,
    PUDS_SVC_PARAM_RDTCI_RMRCDTC = 0x0E,
    PUDS_SVC_PARAM_RDTCI_RMMDTCBSM = 0x0F,
    PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN = 0x10,
    PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM = 0x11,
    PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM = 0x12,
    PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM = 0x13,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamRDTCI : Byte
{
      PUDS_SVC_PARAM_RDTCI_RNODTCBSM = 0x01,
      PUDS_SVC_PARAM_RDTCI_RDTCBSM = 0x02,
      PUDS_SVC_PARAM_RDTCI_RDTCSSI = 0x03,
      PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC = 0x04,
      PUDS_SVC_PARAM_RDTCI_RDTCSSBRN = 0x05,
      PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN = 0x06,
      PUDS_SVC_PARAM_RDTCI_RNODTCBSMR = 0x07,
      PUDS_SVC_PARAM_RDTCI_RDTCBSMR = 0x08,
      PUDS_SVC_PARAM_RDTCI_RSIODTC = 0x09,
      PUDS_SVC_PARAM_RDTCI_RSUPDTC = 0x0A,
      PUDS_SVC_PARAM_RDTCI_RFTFDTC = 0x0B,
      PUDS_SVC_PARAM_RDTCI_RFCDTC = 0x0C,
      PUDS_SVC_PARAM_RDTCI_RMRTFDTC = 0x0D,
      PUDS_SVC_PARAM_RDTCI_RMRCDTC = 0x0E,
      PUDS_SVC_PARAM_RDTCI_RMMDTCBSM = 0x0F,
      PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN = 0x10,
      PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM = 0x11,
      PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM = 0x12,
      PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM = 0x13,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamRDTCI As Byte
    PUDS_SVC_PARAM_RDTCI_RNODTCBSM = &H1
    PUDS_SVC_PARAM_RDTCI_RDTCBSM = &H2
    PUDS_SVC_PARAM_RDTCI_RDTCSSI = &H3
    PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC = &H4
    PUDS_SVC_PARAM_RDTCI_RDTCSSBRN = &H5
    PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN = &H6
    PUDS_SVC_PARAM_RDTCI_RNODTCBSMR = &H7
    PUDS_SVC_PARAM_RDTCI_RDTCBSMR = &H8
    PUDS_SVC_PARAM_RDTCI_RSIODTC = &H9
    PUDS_SVC_PARAM_RDTCI_RSUPDTC = &HA
    PUDS_SVC_PARAM_RDTCI_RFTFDTC = &HB
    PUDS_SVC_PARAM_RDTCI_RFCDTC = &HC
    PUDS_SVC_PARAM_RDTCI_RMRTFDTC = &HD
    PUDS_SVC_PARAM_RDTCI_RMRCDTC = &HE
    PUDS_SVC_PARAM_RDTCI_RMMDTCBSM = &HF
    PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN = &H10
    PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM = &H11
    PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM = &H12
    PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM = &H13
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_DI_BSIDID	0xF180	Boot Software Identification Data Identifier
PUDS_SVC_PARAM_DI_ASIDID	0xF181	Application Software Identification Data Identifier
PUDS_SVC_PARAM_DI_ADIDID	0xF182	Application Data Identification Data Identifier
PUDS_SVC_PARAM_DI_BSFPDID	0xF183	Boot Software Identification Data Identifier
PUDS_SVC_PARAM_DI_ASFPDID	0xF184	Application Software Fingerprint Data Identifier
PUDS_SVC_PARAM_DI_ADFPDID	0xF185	Application Data Fingerprint Data Identifier
PUDS_SVC_PARAM_DI_ADSDID	0xF186	Active Diagnostic Session Data Identifier
PUDS_SVC_PARAM_DI_VMSPNDID	0xF187	Vehicle Manufacturer Spare Part Number Data Identifier
PUDS_SVC_PARAM_RDTCI_RNODTCBSM	1	report Number Of DTC By Status Mask
PUDS_SVC_PARAM_RDTCI_RDTCBSM	2	report DTC By Status Mask
PUDS_SVC_PARAM_RDTCI_RDTCSSI	3	report DTC Snapshot Identification
PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC	4	report DTC Snapshot Record By DTC Number
PUDS_SVC_PARAM_RDTCI_RDTCSSBRN	5	report DTC Snapshot Record By Record Number
PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN	6	report DTC Extended Data Record By DTC Number
PUDS_SVC_PARAM_RDTCI_RNODTCBSMR	7	report Number Of DTC By Severity Mask Record
PUDS_SVC_PARAM_RDTCI_RDTCBSMR	8	report DTC By Severity Mask Record
PUDS_SVC_PARAM_RDTCI_RSIODTC	9	report Severity Information Of DTC
PUDS_SVC_PARAM_RDTCI_RSUPDTC	0x0A (10)	report Supported DTC
PUDS_SVC_PARAM_RDTCI_RFTFDTC	0x0B (11)	report First Test Failed DTC
PUDS_SVC_PARAM_RDTCI_RFCDTC	0x0C (12)	report First Confirmed DTC
PUDS_SVC_PARAM_RDTCI_RMRTFDTC	0x0D (13)	report Most Recent Test Failed DTC
PUDS_SVC_PARAM_RDTCI_RMRCDTC	0x0E (14)	report Most Recent Confirmed DTC
PUDS_SVC_PARAM_RDTCI_RMMDTCBSM	0x0F (15)	report Mirror Memory DTC By Status Mask
PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN	0x10 (16)	report Mirror Memory DTC Extended Data Record By DTC Number
PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM	0x11 (16)	report Number Of Mirror MemoryDTC By Status Mask
PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM	0x12 (17)	report Number Of Emissions Related OBD DTC By Status Mask
PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM	0x13 (18)	report Emissions Related OBD DTC By Status Mask

See also: PUDS_SvcReadDTCInformation (**class-method:** SvcReadDTCInformation).

3.5.26 TPUDSSvcParamRDTCI_DTCSVM

Represents the DTC severity mask's flags (DTCSVM) used with the UDS service SvcReadDTCInformation.

Syntax

C++

```
#define PUDS_SVC_PARAM_RDTCI_DTCSVM_NSA         0x00
#define PUDS_SVC_PARAM_RDTCI_DTCSVM_MO          0x20
#define PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKANH      0x40
#define PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKI        0x80
```

Pascal OO

```
TPUDSSvcParamRDTCI_DTCSVM = (
    PUDS_SVC_PARAM_RDTCI_DTCSVM_NSA = $00,
    PUDS_SVC_PARAM_RDTCI_DTCSVM_MO = $20,
    PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKANH = $40,
    PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKI = $80
```

```
);
```

C#

```
[Flags]
public enum TPUDSSvcParamRDTCI_DTCSVM : byte
{
    PUDS_SVC_PARAM_RDTCI_DTCSVM_NSA = 0x00,
    PUDS_SVC_PARAM_RDTCI_DTCSVM_MO = 0x20,
    PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKANH = 0x40,
    PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKI = 0x80,
}
```

C++ / CLR

```
[Flags]
enum struct TPUDSSvcParamRDTCI_DTCSVM : Byte
{
      PUDS_SVC_PARAM_RDTCI_DTCSVM_NSA = 0x00,
      PUDS_SVC_PARAM_RDTCI_DTCSVM_MO = 0x20,
      PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKANH = 0x40,
      PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKI = 0x80,
};
```

Visual Basic

```
<Flags()> _
Public Enum TPUDSSvcParamRDTCI_DTCSVM As Byte
    PUDS_SVC_PARAM_RDTCI_DTCSVM_NSA = &H0
    PUDS_SVC_PARAM_RDTCI_DTCSVM_MO = &H20
    PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKANH = &H40
    PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKI = &H80
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_RDTCI_DTCSVM_NSA	0	DTC severity bit definitions : no SeverityAvailable
PUDS_SVC_PARAM_RDTCI_DTCSVM_MO	0x20 (32)	DTC severity bit definitions : maintenance Only
PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKANH	0x40 (64)	DTC severity bit definitions : check At Next Halt
PUDS_SVC_PARAM_RDTCI_DTCSVM_CHKI	0x80 (128)	DTC severity bit definitions : check Immediately

See also: PUDS_SvcReadDTCInformation (**class-method:** SvcReadDTCInformation).

3.5.27 TPUDSSvcParamIOCBI

Represents the InputOutput Control Parameter for UDS service InputOutputControlByIdentifier.

Syntax

C++

```
#define PUDS_SVC_PARAM_IOCBI_RCTECU             0x00
#define PUDS_SVC_PARAM_IOCBI_RTD                0x01
#define PUDS_SVC_PARAM_IOCBI_FCS                0x02
#define PUDS_SVC_PARAM_IOCBI_STA                0x03
```

Pascal OO

```
TPUDSSvcParamIOCBI = (
    PUDS_SVC_PARAM_IOCBI_RCTECU = $00,
    PUDS_SVC_PARAM_IOCBI_RTD = $01,
    PUDS_SVC_PARAM_IOCBI_FCS = $02,
    PUDS_SVC_PARAM_IOCBI_STA = $03
);
```

C#

```
public enum TPUDSSvcParamIOCBI : byte
{
    PUDS_SVC_PARAM_IOCBI_RCTECU = 0x00,
    PUDS_SVC_PARAM_IOCBI_RTD = 0x01,
    PUDS_SVC_PARAM_IOCBI_FCS = 0x02,
    PUDS_SVC_PARAM_IOCBI_STA = 0x03,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamIOCBI : Byte
{
      PUDS_SVC_PARAM_IOCBI_RCTECU = 0x00,
      PUDS_SVC_PARAM_IOCBI_RTD = 0x01,
      PUDS_SVC_PARAM_IOCBI_FCS = 0x02,
      PUDS_SVC_PARAM_IOCBI_STA = 0x03,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamIOCBI As Byte
    PUDS_SVC_PARAM_IOCBI_RCTECU = &H0
    PUDS_SVC_PARAM_IOCBI_RTD = &H1
    PUDS_SVC_PARAM_IOCBI_FCS = &H2
    PUDS_SVC_PARAM_IOCBI_STA = &H3
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_IOCBI_RCTECU	0	Return Control To ECU
PUDS_SVC_PARAM_IOCBI_RTD	1	Reset To Default
PUDS_SVC_PARAM_IOCBI_FCS	2	Freeze Current State
PUDS_SVC_PARAM_IOCBI_STA	3	Short Term Adjustment

See also: PUDS_ SvcInputOutputControlByIdentifier (**class-method:** SvcInputOutputControlByIdentifier).

3.5.28 TPUDSSvcParamRC

Represents the subfunction parameter for UDS service RoutineControl.

Syntax

C++

```
#define PUDS_SVC_PARAM_RC_STR              0x01
#define PUDS_SVC_PARAM_RC_STPR             0x02
#define PUDS_SVC_PARAM_RC_RRR              0x03
```

Pascal OO

```
TPUDSSvcParamRC = (
    PUDS_SVC_PARAM_RC_STR = $01,
    PUDS_SVC_PARAM_RC_STPR = $02,
    PUDS_SVC_PARAM_RC_RRR = $03
);
```

C#

```
public enum TPUDSSvcParamRC : byte
{
    PUDS_SVC_PARAM_RC_STR = 0x01,
    PUDS_SVC_PARAM_RC_STPR = 0x02,
    PUDS_SVC_PARAM_RC_RRR = 0x03,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamRC : Byte
{
      PUDS_SVC_PARAM_RC_STR = 0x01,
      PUDS_SVC_PARAM_RC_STPR = 0x02,
      PUDS_SVC_PARAM_RC_RRR = 0x03,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamRC As Byte
    PUDS_SVC_PARAM_RC_STR = &H1
    PUDS_SVC_PARAM_RC_STPR = &H2
    PUDS_SVC_PARAM_RC_RRR = &H3
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_RC_STR	1	Start Routine
PUDS_SVC_PARAM_RC_STPR	2	Stop Routine
PUDS_SVC_PARAM_RC_RRR	3	Request Routine Results

See also: PUDS_SvcRoutineControl (**class-method:** SvcRoutineControl).

3.5.29 TPUDSSvcParamRC_RID

Represents the routine identifier used with the UDS service RoutineControl.

Syntax

C++

```
#define PUDS_SVC_PARAM_RC_RID_DLRI_       0xE200
#define PUDS_SVC_PARAM_RC_RID_EM_         0xFF00
#define PUDS_SVC_PARAM_RC_RID_CPD_        0xFF01
#define PUDS_SVC_PARAM_RC_RID_EMMDTC_     0xFF02
```

Pascal OO

```
TPUDSSvcParamRC_RID= (
```

```
    PUDS_SVC_PARAM_RC_RID_DLRI_ = $E200,
    PUDS_SVC_PARAM_RC_RID_EM_ = $FF00,
    PUDS_SVC_PARAM_RC_RID_CPD_ = $FF01,
    PUDS_SVC_PARAM_RC_RID_EMMDTC_ = $FF02
);
```

C#

```
public enum TPUDSSvcParamRC_RID : ushort
{
    PUDS_SVC_PARAM_RC_RID_DLRI_ = 0xE200,
    PUDS_SVC_PARAM_RC_RID_EM_ = 0xFF00,
    PUDS_SVC_PARAM_RC_RID_CPD_ = 0xFF01,
    PUDS_SVC_PARAM_RC_RID_EMMDTC_ = 0xFF02,
}
```

C++ / CLR

```
enum struct TPUDSSvcParamRC_RID : unsigned short
{
      PUDS_SVC_PARAM_RC_RID_DLRI_ = 0xE200,
      PUDS_SVC_PARAM_RC_RID_EM_ = 0xFF00,
      PUDS_SVC_PARAM_RC_RID_CPD_ = 0xFF01,
      PUDS_SVC_PARAM_RC_RID_EMMDTC_ = 0xFF02,
};
```

Visual Basic

```
Public Enum TPUDSSvcParamRC_RID As UShort
    PUDS_SVC_PARAM_RC_RID_DLRI_ = &HE200
    PUDS_SVC_PARAM_RC_RID_EM_ = &HFF00
    PUDS_SVC_PARAM_RC_RID_CPD_ = &HFF01
    PUDS_SVC_PARAM_RC_RID_EMMDTC_ = &HFF02
End Enum
```

Values

Name	Value	Description
PUDS_SVC_PARAM_RC_RID_DLRI_	0xE200 (57856)	Deploy Loop Routine ID
PUDS_SVC_PARAM_RC_RID_EM_	0xFF00 (65280)	Erase Memory
PUDS_SVC_PARAM_RC_RID_CPD_	0xFF01 (65281)	Check Programming Dependencies
PUDS_SVC_PARAM_RC_RID_EMMDTC_	0xFF02 (65282)	Erase Mirror Memory DTCs

See also: PUDS_SvcRoutineControl (**class-method:** SvcRoutineControl).

3.6 Methods

The methods defined for the classes UDSApi and TUDSApi are divided in 4 groups of functionality. Note that these methods are static and can be called in the name of the class, without instantiation.

Connection

	Function	Description
	Initialize	Initializes a PUDS channel
	Uninitialize	Uninitializes a PUDS channel

Configuration

	Function	Description
	SetValue	Sets a configuration or information value within a PUDS Channel

Information

	Function	Description
	GetValue	Retrieves information from a PUDS Channel
	GetStatus	Retrieves the current BUS status of a PUDS Channel

Communication

	Function	Description
	Read	Reads a UDS message from the receive queue of a PUDS Channel
	Write	Writes to transmit queue a UDS message using a connected PUDS Channel
	Reset	Resets the receive and transmit queues of a PUDS Channel
	WaitForSingleMessage	Waits for a confirmation or response message based on a UDS Message Request
	WaitForMultipleMessage	Waits for multiple reponses based on a UDS Message Request
	WaitForService	Waits for the confirmation of the transmission of a UDS service request and the reception of its response
	WaitForServiceFunctional	Waits for the confirmation of the transmission of a functional UDS service request and the reception of its responses
	ProcessResponse	Processes a UDS response message to update internal UDS features
	SvcDiagnosticSessionControl	Writes to the transmit queue a request for UDS service DiagnosticSessionControl
	SvcECUReset	Writes to the transmit queue a request for UDS service ECUReset
	SvcSecurityAccess	Writes to the transmit queue a request for UDS service SecurityAccess
	SvcCommunicationControl	Writes to the transmit queue a request for UDS service CommunicationControl
	SvcTesterPresent	Writes to the transmit queue a request for UDS service TesterPresent
	SvcSecuredDataTransmission	Writes to the transmit queue a request for UDS service SecuredDataTransmission
	SvcControlDTCSetting	Writes to the transmit queue a request for UDS service ControlDTCSetting
	SvcResponseOnEvent	Writes to the transmit queue a request for UDS service ResponseOnEvent
	SvcLinkControl	Writes to the transmit queue a request for UDS service LinkControl
	SvcReadDataByIdentifier	Writes to the transmit queue a request for UDS service ReadDataByIdentifier
	SvcReadMemoryByAddress	Writes to the transmit queue a request for UDS service ReadMemoryByAddress
	SvcReadScalingDataByIdentifier	Writes to the transmit queue a request for UDS service ReadScalingDataByIdentifier
	SvcReadDataByPeriodicIdentifier	Writes to the transmit queue a request for UDS service ReadDataByPeriodicIdentifier
	SvcDynamicallyDefineDataIdentifierDBID	Writes to the transmit queue a request for UDS service DynamicallyDefineDataIdentifier
	SvcDynamicallyDefineDataIdentifierDBMA	Writes to the transmit queue a request for UDS service DynamicallyDefineDataIdentifier
	SvcDynamicallyDefineDataIdentifierCDDDI	Writes to the transmit queue a request for UDS service DynamicallyDefineDataIdentifier
	SvcWriteDataByIdentifier	Writes to the transmit queue a request for UDS service WriteDataByIdentifier
	SvcWriteMemoryByAddress	Writes to the transmit queue a request for UDS service WriteMemoryByAddress

	Function	Description
	SvcClearDiagnosticInformation	Writes to the transmit queue a request for UDS service ClearDiagnosticInformation
	SvcReadDTCInformation	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcReadDTCInformationRDTCSSBDTC	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcReadDTCInformationRDTCSSBRN	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcReadDTCInformationReportExtended	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcReadDTCInformationReportSeverity	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcReadDTCInformationRSIODTC	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcReadDTCInformationNoParam	Writes to the transmit queue a request for UDS service ReadDTCInformation
	SvcInputOutputControlByIdentifier	Writes to the transmit queue a request for UDS service InputOutputControlByIdentifier
	SvcRoutineControl	Writes to the transmit queue a request for UDS service RoutineControl
	SvcRequestDownload	Writes to the transmit queue a request for UDS service RequestDownload
	SvcRequestUpload	Writes to the transmit queue a request for UDS service RequestUpload
	SvcTransferData	Writes to the transmit queue a request for UDS service TransferData
	SvcRequestTransferExit	Writes to the transmit queue a request for UDS service RequestTransferExit

3.6.1 Initialize

Initializes a PUDS Channel.

Overloads

	Function	Description
	Initialize(TPUDSCANHandle, TPUDSBaudrate)	Initializes a Plug-And-Play PUDS Channel
	Initialize(TPUDSCANHandle, TPUDSBaudrate, TPUDSHWType, UInt32, UInt16)	Initializes a Non-Plug-And-Play PUDS Channel

3.6.2 Initialize(TPUDSCANHandle, TPUDSBaudrate)

Initializes a PUDS Channel which represents a Plug & Play PCAN-Device.

Syntax

Pascal OO

```
class function Initialize(
  CanChannel: TPUDSCANHandle;
  Baudrate: TPUDSBaudrate
  ): TPUDSStatus; overload;
```

C#

```
public static TPUDSStatus Initialize(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel,
        TPUDSBaudrate Baudrate);
```

C++ / CLR

```
static TPUDSStatus Initialize(
  [MarshalAs(UnmanagedType::U2)]
  TPUDSCANHandle CanChannel,
  [MarshalAs(UnmanagedType::U2)]
  TPUDSBaudrate Baudrate);
```

Visual Basic

```
Public Shared Function Initialize( _
  <MarshalAs(UnmanagedType.U2)> _
  ByVal CanChannel As TPUDSCANHandle, _
  <MarshalAs(UnmanagedType.U2)> _
  ByVal Baudrate As TPUDSBaudrate) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CANChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Baudrate	The speed for the communication (see TPUDSBaudrate on page 25)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_ALREADY_INITIALIZED	Indicates that the desired PUDS Channel is already in use
PUDS_ERROR_CAN_ERROR	This error flag states that the error is composed of a more precise PCAN-Basic error

Remarks: As indicated by its name, the Initialize method initiates a PUDS 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 PUDS_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PUDS Channel means:

- to reserve the Channel for the calling application/process
- to allocate channel resources, like receive and transmit queues
- to forward initialization to PCAN-ISO-TP API and PCAN-Basic API, hence registering/connecting the Hardware denoted by the channel handle
- to set-up the default values of the different parameters (See GetValue) and configure default standard ISO-TP mappings:
  - 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)

The Initialization process will fail if an application tries to initialize a PUDS-Channel that has already been initialized within the same process.

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

Example

The following example shows the initialize and uninitialize processes for a Plug-And-Play channel (channel 2 of a PCAN-PCI hardware).

C#:

```
TPUDSStatus result;

// The Plug & Play Channel (PCAN-PCI) is initialized
result = UDSApi.Initialize(UDSApi.PUDS_PCIBUS2, TPUDSBaudrate.PUDS_BAUD_500K);
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released
UDSApi.Uninitialize(UDSApi.PUDS_NONEBUS);
```

C++/CLR:

```
TPUDSStatus result;

// The Plug & Play Channel (PCAN-PCI) is initialized
result = UDSApi::Initialize(UDSApi::PUDS_PCIBUS2, PUDS_BAUD_500K);
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released
UDSApi::Uninitialize(UDSApi::PUDS_NONEBUS);
```

Visual Basic:

```
Dim result As TPUDSStatus

' The Plug & Play Channel (PCAN-PCI) is initialized
result = UDSApi.Initialize(UDSApi.PUDS_PCIBUS2, TPUDSBaudrate.PUDS_BAUD_500K)
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
End If

' All initialized channels are released
UDSApi.Uninitialize(UDSApi.PUDS_NONEBUS)
```

Pascal OO:

```
var
  result: TPUDSStatus;
```

```
begin
    // The Plug & Play Channel (PCAN-PCI) is initialized
    result := TUdsApi.Initialize(TUdsApi.PUDS_PCIBUS2, PUDS_BAUD_500K);
    if (result <> PUDS_ERROR_OK) then
      MessageBox(0, 'Initialization failed', 'Error', MB_OK)
    else
      MessageBox(0, 'PCAN-PCI (Ch-2) was initialized', 'Success', MB_OK);

    // All initialized channels are released
    TUdsApi.Uninitialize(TUdsApi.PUDS_NONEBUS);
end;
```

See also: Uninitialize on page 82, GetValue on page 94, Understanding PCAN-UDS on page 8.

Plain function version: UDS_Initialize.

3.6.3 Initialize(TPUDSCANHandle, TPUDSBaudrate, TPUDSHWType, UInt32, UInt16)

Initialize a PUDS Channel which represents a Non-Plug & Play PCAN-Device.

Syntax

Pascal OO

```
class function Initialize(
  CanChannel: TPUDSCANHandle;
  Baudrate: TPUDSBaudrate;
  HwType: TPUDSHWType;
  IOPort: LongWord;
  Interrupt: Word
  ): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "CANTP_Initialize")]
public static extern TPUDSStatus Initialize(
  [MarshalAs(UnmanagedType.U2)]
  TPUDSCANHandle CanChannel,
  [MarshalAs(UnmanagedType.U2)]
  TPUDSBaudrate Baudrate,
  [MarshalAs(UnmanagedType.U1)]
  TPUDSHWType HwType,
  UInt32 IOPort,
  UInt16 Interrupt);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "CANTP_Initialize")]
static TPUDSStatus Initialize(
  [MarshalAs(UnmanagedType::U2)]
  TPUDSCANHandle CanChannel,
  [MarshalAs(UnmanagedType::U2)]
```

```
  TPUDSBaudrate Baudrate,
  [MarshalAs(UnmanagedType::U1)]
  TPUDSHWType HwType,
  UInt32 IOPort,
  UInt16 Interrupt);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="CANTP_Initialize")> _
Public Shared Function Initialize( _
  <MarshalAs(UnmanagedType.U2)> _
  ByVal CanChannel As TPUDSCANHandle, _
  <MarshalAs(UnmanagedType.U2)> _
  ByVal Baudrate As TPUDSBaudrate, _
  <MarshalAs(UnmanagedType.U1)> _
  ByVal HwType As TPUDSHWType, _
  ByVal IOPort As UInt32, _
  ByVal Interrupt As UInt16) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Baudrate	The speed for the communication (see TPUDSBaudrate on page 25)
HwType	The type of hardware (see TPUDSHWType on page 28)
IOPort	The I/O address for the parallel port.
Interrupt	Interrupt number of the parallel port.

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_ALREADY_INITIALIZED	Indicates that the desired PUDS Channel is already in use
PUDS_ERROR_CAN_ERROR	This error flag states that the error is composed of a more precise PCAN-Basic error

Remarks: As indicated by its name, the Initialize method initiates a PUDS 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 PUDS_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PUDS Channel means:

- to reserve the Channel for the calling application/process
- to allocate channel resources, like receive and transmit queues
- to forward initialization to PCAN-ISO-TP API and PCAN-Basic API, hence registering/connecting the Hardware denoted by the channel handle
- to set-up the default values of the different parameters (See GetValue) and configure default standard ISO-TP mappings:

- 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)

The Initialization process will fail if an application tries to initialize a PUDS-Channel that has already been initialized within the same process.

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

Example

The following example shows the initialize and uninitializes processes for a Non-Plug-And-Play channel (channel 1 of the PCAN-DNG).

C#

```
TPUDSStatus result;

// The Non-Plug & Play Channel (PCAN-DNG) is initialized
result = UDSApi.Initialize(UDSApi.PUDS_DNGBUS1, TPUDSBaudrate.PUDS_BAUD_500K,
TPUDSHWType.PUDS_TYPE_DNG_SJA, 0x378, 7);
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released
UDSApi.Uninitialize(UDSApi.PUDS_NONEBUS);
```

C++/CLR:

```
TPUDSStatus result;

// The Non-Plug & Play Channel (PCAN-DNG) is initialized
result = UDSApi::Initialize(UDSApi::PUDS_DNGBUS1, PUDS_BAUD_500K,
PUDS_TYPE_DNG_SJA, 0x378, 7);
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");
// All initialized channels are released
UDSApi::Uninitialize(UDSApi::PUDS_NONEBUS);
```

Visual Basic:

```
Dim result As TPUDSStatus

' The Non-Plug & Play Channel (PCAN-DNG) is initialized
result = UDSApi.Initialize(UDSApi.PUDS_DNGBUS1, TPUDSBaudrate.PUDS_BAUD_500K,
TPUDSHWType.PUDS_TYPE_DNG_SJA, &H378, 7)
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
```

```
End If

' All initialized channels are released
UDSApi.Uninitialize(UDSApi.PUDS_NONEBUS)
```

Pascal OO:

```
var
  result: TPUDSStatus;

begin
  // The Non-Plug & Play Channel (PCAN-DNG) is initialized
  result := TUDSApi.Initialize(TUdsApi.PUDS_DNGBUS1, PUDS_BAUD_500K, PUDS_TYPE_DNG_SJA, $378, 7);
  if (result <> PUDS_ERROR_OK) then
    MessageBox(0, 'Initialization failed', 'Error', MB_OK)
  else
    MessageBox(0, 'PCAN-PCI (Ch-2) was initialized', 'Error', MB_OK);

  // All initialized channels are released
  TUdsApi.Uninitialize(TUdsApi.PUDS_NONEBUS);
end;
```

See also: Uninitialize below, GetValue on page 94, Understanding PCAN-UDS on page 8.

Plain function version: CANTP_Uninitialize.

3.6.4 Uninitialize

Uninitialize a PUDS Channel.

Syntax

Pascal OO

```
class function Uninitialize(
  CanChannel: TUDSCANHandle
  ): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Uninitialize")]
public static TPUDSStatus Uninitialize(
        [MarshalAs(PUnmanagedType.U2)]
        TPUDSCANHandle CanChannel);
```

C++ / CLR

```
static TPUDSStatus Uninitialize (
  [MarshalAs(UnmanagedType::U2)]
  TPUDSCANHandle CanChannel);
```

Visual Basic

```
Public Shared Function Initialize( _
```

```
    <MarshalAs(UnmanagedType.U2)> _
    ByVal CanChannel As TPUDSCANHandle) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

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

Remarks: A PUDS Channel can be released using one of these possibilities.

- Single-Release: Given a handle of a PUDS 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 PUDS_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

The following example shows the initialize and uninitializes processes for a Plug-And-Play channel (channel 2 of a PCAN-PCI hardware).

C#:

```
TPUDSStatus result;

// The Plug & Play Channel (PCAN-PCI) is initialized
result = UDSApi.Initialize(UDSApi.PUDS_PCIBUS2,
TPUDSBaudrate.PUDS_BAUD_500K);
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// Release channel
UDSApi.Uninitialize(UDSApi.PUDS_PCIBUS2);
```

C++/CLR:

```
TPUDSStatus result;

// The Plug & Play Channel (PCAN-PCI) is initialized
result = UDSApi::Initialize(UDSApi::PUDS_PCIBUS2, PUDS_BAUD_500K);
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");

// Release channel
UDSApi::Uninitialize(UDSApi::PUDS_PCIBUS2);
```

Visual Basic:

```
Dim result As TPUDSStatus

' The Plug & Play Channel (PCAN-PCI) is initialized
result = UDSApi.Initialize(UDSApi.PUDS_PCIBUS2,
TPUDSBaudrate.PUDS_BAUD_500K)
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
End If

' Release channel
UDSApi.Uninitialize(UDSApi.PUDS_PCIBUS2)
```

Pascal OO:

```
var
  result: TPUDSStatus;

begin
  // The Plug & Play Channel (PCAN-PCI) is initialized
  result := TUdsApi.Initialize(TUdsApi.PUDS_PCIBUS2, PUDS_BAUD_500K);
  if (result <> PUDS_ERROR_OK) then
    MessageBox(0, 'Initialization failed', 'Error', MB_OK)
  else
    MessageBox(0, 'PCAN-PCI (Ch-2) was initialized', 'Error', MB_OK);

  // Release channel
  TUdsApi.Uninitialize(TUdsApi.PUDS _PCIBUS2);
end;
```

See also: Initialize on page 76.

Plain function version: CANTP_Uninitialize.

3.6.5 SetValue

Set a configuration or information value within a PUDS Channel.

Overloads

	Function	Description
	SetValue(TPUDSCANHandle, TPUDSParameter, UInt32, UInt32);	Sets a configuration or information numeric value within a PUDS Channel
	SetValue(TPUDSCANHandle, TPUDSParameter, String, UInt32);	Sets a configuration or information string value within a PUDS Channel
	SetValue(TPUDSCANHandle, TPUDSParameter, Byte[], UInt32);	Sets a configuration or information with an array of bytes within a PUDS Channel
	SetValue(TPUDSCANHandle, TPUDSParameter, IntPrt, UInt32);	Sets a configuration or information within a PUDS Channel

3.6.6 SetValue (TPUDSCANHandle, TPUDSParameter, UInt32, Uint32)

Set a configuration or information numeric value within a PUDS Channel.

Syntax

Pascal OO

```
class function SetValue(
        CanChannel: TPUDSCANHandle;
        Parameter: TPUDSParameter;
        NumericBuffer: PLongWord;
        BufferLength: LongWord
        ): TPUDSStatus; overload;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
public static extern TPUDSStatus SetValue(
   [MarshalAs(UnmanagedType.U2)]
   TPUDSCANHandle CanChannel,
   [MarshalAs(UnmanagedType.U1)]
   TPUDSParameter Parameter,
   ref UInt32 NumericBuffer,
   UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
static TPUDSStatus SetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPUDSParameter Parameter,
        UInt32% NumericBuffer,
```

```
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        ByRef NumericBuffer As UInt32, _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
NumericBuffer	The buffer containing the numeric value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with an integer buffer

Remarks: Use the method SetValue to set configuration information or environment values of a PUDS Channel. Note that any calls with non UDS parameters (ie. TPUDSParameter) will be forwarded to PCAN-ISO-TP API and PCAN-Basic API.

More information about the parameters and values that can be set can be found in Parameter Value Definitions. Since most of the UDS parameters require a numeric value (byte or integer) this is the most common and useful override.

Example

The following example shows the use of the method SetValue on the channel PUDS_PCIBUS2 to enable debug mode.

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

C#

```
TPUDSStatus result;
UInt32 iBuffer = 0;

// Enable CAN DEBUG mode
iBuffer = UDSApi.PUDS_DEBUG_CAN;
result = UDSApi.SetValue(UDSApi.PUDS_PCIBUS2, TPUDSParameter.PUDS_PARAM_DEBUG, ref
iBuffer, sizeof(UInt32));
```

```
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Failed to set value");
else
    MessageBox.Show("Value changed successfully ");
```

C++/CLR:

```
TPUDSStatus result;
UInt32 iBuffer = 0;

// Enable CAN DEBUG mode
iBuffer = UDSApi::PUDS_DEBUG_CAN;
result = UDSApi::SetValue(UDSApi::PUDS_PCIBUS2, PUDS_PARAM_DEBUG, iBuffer,
sizeof(UInt32));
if (result != PUDS _ERROR_OK)
      MessageBox::Show("Failed to set value");
else
      MessageBox::Show("Value changed successfully ");
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim iBuffer As UInt32 = 0

' Enable CAN DEBUG mode
iBuffer = UDSApi.PUDS_DEBUG_CAN
result=UDSApi.SetValue(UDSApi.PUDS_PCIBUS2,
TPUDSParameter.PUDS_PARAM_DEBUG, iBuffer, Convert.ToUInt32(Len(iBuffer)))
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Failed to set value")
Else
    MessageBox.Show("Value changed successfully ")
End If
```

Pascal OO:

```
var
  result: TPUDSStatus;
  iBuffer: UINT;

begin
  // Enable CAN DEBUG mode
  iBuffer := TUdsApi.PUDS_DEBUG_CAN;
  result := TUdsApi.SetValue(TUdsApi.PUDS_PCIBUS2, PUDS_PARAM_DEBUG, PLongWord(@iBuffer),
sizeof(iBuffer));
  if (result <> PUDS_ERROR_OK) then
    MessageBox(0, 'Failed to set value', 'Error', MB_OK)
  else
    MessageBox(0, 'Value changed successfully ', 'Error', MB_OK);
end;
```

See also: TPUDSParameter on page 31, Parameter Value Definitions on page 332, GetValue on page 94.

Plain function version: UDS_GetValue.

3.6.7 SetValue (TPUDSCANHandle, TPUDSParameter, StringBuffer, Uint32)

Set a configuration or information string value within a PUDS Channel.

Syntax

Pascal OO

```
class function SetValue(
        CanChannel: TPUDSCANHandle;
        Parameter: TPUDSParameter;
        StringBuffer: PAnsiChar;
        BufferLength: LongWord
        ): TPUDSStatus; overload;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
public static extern TPUDSStatus SetValue(
   [MarshalAs(UnmanagedType.U2)]
   TPUDSCANHandle CanChannel,
   [MarshalAs(UnmanagedType.U1)]
   TPUDSParameter Parameter,
   [MarshalAs(UnmanagedType.LPStr, SizeParamIndex = 3)]
   string StringBuffer,
   UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
static TPUDSStatus SetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPUDSParameter Parameter,
        [MarshalAs(UnmanagedType::LPStr, SizeParamIndex = 3)]
        String^ StringBuffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        <MarshalAs(UnmanagedType.LPStr, SizeParamIndex:=3)> _
        ByVal StringBuffer As String, _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
StringBuffer	The buffer containing the string value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passedto the method are invalid. Check the value of 'Parameter' and assert it is compatible with an integer buffer

Remarks: This overrides is only defined for users who wishes to configure PCAN-Basic API through the UDS API.

See also: GetValue on page 94, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_SetValue.

3.6.8 SetValue (TPUDSANHandle, TPUDSParameter, Byte[], Uint32)

Set a configuration or information value as a byte array within a PUDS Channel.

Syntax

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
public static extern TPUDSStatus SetValue(
  [MarshalAs(UnmanagedType.U2)]
  TPUDSCANHandle CanChannel,
  [MarshalAs(UnmanagedType.U1)]
  TPUDSParameter Parameter,
  [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
        Byte[] Buffer,
  UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
static TPUDSStatus SetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPUDSParameter Parameter,
        [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 3)]
        array<Byte>^ Buffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        ByVal Buffer As Byte(), _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
Buffer	The buffer containing the array value to be set
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passedto the method are invalid. Check the value of 'Parameter' and assert it is compatible with an integer buffer

Remarks: Use the method SetValue to set configuration information or environment values of a PUDS Channel. Note that any calls with non UDS parameters (ie. TPUDSParameter) will be forwarded to PCAN-ISO-TP API and PCAN-Basic API.

Note: That any calls with non UDS parameters (ie. TPUDSParameter) will be forwarded to PCAN-ISO-TP API and PCAN-Basic API.

More information about the parameters and values that can be set can be found in Parameter Value Definition.

Example

The following example shows the use of the method SetValue on the channel PUDS_USBBUS1 to change the current UDS Session Information.

Note: this only affects the API client side ONLY, no communication with any ECUs is made. If a user wants to disable the automatic transmission of TesterPresent requests that keeps alive a non-default diagnostic session, he/she should set the `SESSION_TYPE` to the default diagnostic session (`TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_DS`).

It is assumed that the channel was already initialized and the user retrieved the current session information with the GetValue API function.

C#

```
TPUDSStatus result;

// Define unlimited blocksize
result = UDSApi.SetValue(UDSApi.PUDS_USBBUS1,
TPUDSParameter.PUDS_PARAM_BLOCK_SIZE, new byte[] { 0 }, 1);
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Failed to set value");
else
    MessageBox.Show("Value changed successfully ");
```

C++/CLR:

```
TPUDSStatus result;

// Define unlimited blocksize
result = UDSApi::SetValue(UDSApi::PUDS_USBBUS1, PUDS_PARAM_BLOCK_SIZE,
gcnew array<Byte> { 0 }, 1);
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Failed to set value");
else
      MessageBox::Show("Value changed successfully ");
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim bufferArray(2) As Byte

' Define unlimited blocksize
bufferArray(0) = 0
result = UDSApi.SetValue(UDSApi.PUDS_USBBUS1, TPUDSParameter.PUDS_PARAM_BLOCK_SIZE,
bufferArray, Convert.ToUInt32(bufferArray.Length))
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Failed to set value")
Else
    MessageBox.Show("Value changed successfully ")
End If
```

See also: GetValue on page 94, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_SetValue.

3.6.9 SetValue(TPUDSCANHandle, TPUDSParameter, IntPtr, UInt32)

Set a configuration or information value as a byte array within a PUDS Channel.

Syntax

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
public static extern TPUDSStatus SetValue(
  [MarshalAs(UnmanagedType.U1)]
  TPUDSCANHandle CanChannel,
  [MarshalAs(UnmanagedType.U1)]
  TPUDSParameter Parameter,
  IntPtr Buffer,
```

```
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SetValue")]
static TPUDSStatus SetValue(
        [MarshalAs(UnmanagedType::U1)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPUDSParameter Parameter,
        IntPtr Buffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        ByVal Buffer As IntPtr, _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
Buffer	The buffer containing the array value to be set
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passedto the method are invalid. Check the value of 'Parameter' and assert it is compatible with an integer buffer

Remarks

Use the method SetValue to set configuration information or environment values of a PUDS Channel.

Note: That any calls with non UDS parameters (ie. TPUDSParameter) will be forwarded to PCAN-ISO-TP API and PCAN-Basic API.

More information about the parameters and values that can be set can be found in Parameter Value Definitions.

Example

The following example shows the use of the method SetValue on the channel PUDS_USBBUS1 to change the current UDS Session Information.

Note: this only affects the API client side ONLY, no communication with any ECUs is made. If a user wants to disable the automatic transmission of TesterPresent requests that keeps alive a non-default diagnostic session, he/she should set the `SESSION_TYPE` to the default diagnostic session (`TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_DS`).

It is assumed that the channel was already initialized and the user retrieved the current session information with the GetValue API function.

C#:

```
TPCANTPHandle CanChannel = UDSApi.PUDS_USBBUS1;
TPUDSStatus result;
TPUDSSessionInfo sessionInfo;
int sessionSize;
IntPtr sessionPtr;

sessionInfo = new TPUDSSessionInfo();
// the current session information is retrieved (see getValue example)
…
// change the session type
sessionInfo.SESSION_TYPE = (byte)UDSApi.TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_DS;
// Get pointer from structure
sessionSize = Marshal.SizeOf(sessionInfo);
sessionPtr = Marshal.AllocHGlobal(sessionSize);
Marshal.StructureToPtr(sessionInfo, sessionPtr, false);
// Set Session information
result = UDSApi.SetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SESSION_INFO,
sessionPtr, (uint) sessionSize);
// free ressource
Marshal.FreeHGlobal(sessionPtr);
```

C++/CLR:

```
TPUDSCANHandle CanChannel = UDSApi::PUDS_USBBUS1;
TPUDSStatus result;
TPUDSSessionInfo^ sessionInfo;
int sessionSize;
IntPtr sessionPtr;

sessionInfo = gcnew TPUDSSessionInfo();
// the current session information is retrieved (see getValue example)
…
// change the session type
sessionInfo->SESSION_TYPE = (unsigned char)
UDSApi::TPUDSSvcParamDSC::PUDS_SVC_PARAM_DSC_DS;
// Get pointer from structure
sessionSize = Marshal::SizeOf(sessionInfo);
sessionPtr = Marshal::AllocHGlobal(sessionSize);
Marshal::StructureToPtr(sessionInfo, sessionPtr, false);
// Get Session information
result = UDSApi::SetValue(CanChannel, PUDS_PARAM_SESSION_INFO, sessionPtr,
sessionSize);
// free ressource
Marshal::FreeHGlobal(sessionPtr);
```

Visual Basic:

```
Dim CanChannel As TPCANTPHandle = UDSApi.PUDS_USBBUS1
Dim result As TPUDSStatus
Dim sessionInfo As TPUDSSessionInfo
Dim sessionSize as Integer
Dim sessionPtr as IntPtr

sessionInfo = new TPUDSSessionInfo
' the current session information is retrieved (see getValue example)
…
' change the session type
sessionInfo.SESSION_TYPE = UDSApi.TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_DS
' Get pointer from structure
sessionSize = Marshal.SizeOf(sessionInfo)
sessionPtr = Marshal.AllocHGlobal(sessionSize)
Marshal.StructureToPtr(sessionInfo, sessionPtr, false)
' Set Session information
result = UDSApi.SetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SESSION_INFO,
sessionPtr, Convert.ToUInt32(sessionSize))
' free ressource
Marshal.FreeHGlobal(sessionPtr)
```

See also: GetValue below, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_SetValue.

3.6.10 GetValue

Retrieve information from a PUDS Channel.

Overloads

	Function	Description
	GetValue(TPUDSCANHandle, TPUDSParameter, UInt32, UInt32);	Retrieves information from a PUDS Channel in numeric form
	GetValue(TPUDSCANHandle, TPUDSParameter, String, UInt32);	Retrieves information from a PUD Channel in text form
	GetValue(TPUDSCANHandle, TPUDSParameter, Byte[], UInt32)	Retrieves information from a PUDS Channel in byte array form
	GetValue(TPUDSCANHandle, TPUDSParameter, IntPtr, UInt32)	Retrieves information from a PUDS Channel in pointer form

3.6.11 GetValue (TPUDSCANHandle, TPUDSParameter, StringBuffer, Uint32)

Retrieve information from a PUDS Channel in text form.

Syntax

Pascal OO

```
class function GetValue(
        CanChannel: TPUDSCANHandle;
        Parameter: TPUDSParameter;
        StringBuffer: PAnsiChar;
        BufferLength: LongWord
```

```
): TPUDSStatus; overload;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
public static extern TPUDSStatus GetValue(
[MarshalAs(UnmanagedType.U2)]
TPUDSCANHandle CanChannel,
[MarshalAs(UnmanagedType.U1)]
TPUDSParameter Parameter,
StringBuilder StringBuffer,
UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
static TPUDSStatus GetValue(
[MarshalAs(UnmanagedType::U2)]
TPUDSCANHandle CanChannel,
[MarshalAs(UnmanagedType::U1)]
TPUDSParameter Parameter,
StringBuilder^ StringBuffer,
UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UD_GetValue")> _
Public Shared Function GetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        ByVal StringBuffer As StringBuilder, _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
StringBuffer	The buffer to return the required string value
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application.
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with a string buffer

Example

The following example shows the use of the method GetValue to retrieve the version of the UDS API. Depending on the result, a message will be shown to the user.

C#

```
TPUDSStatus result;
StringBuilder BufferString;

// Get API version
BufferString = new StringBuilder(255);
result = UDSApi.GetValue(UDSApi.PUDS_NONEBUS,
TPUDSParameter.PUDS_PARAM_API_VERSION, BufferString, 255);
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Failed to get value");
else
    MessageBox.Show(BufferString.ToString());
```

C++ / CLR

```
TPUDSStatus result;
StringBuilder^ BufferString;

// Get API version
BufferString = gcnew StringBuilder(255);
result = UDApi::GetValue(UDSApi::PUDS_NONEBUS,
      PUDS_PARAM_API_VERSION, BufferString, 255);
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Failed to get value");
else
      MessageBox::Show(BufferString->ToString());
```

Visual Basic

```
Dim result As TPUDSStatus
Dim BufferString As StringBuilder

' Get API version
BufferString = New StringBuilder(255)
result = UDSApi.GetValue(UDSApi.PUDS_NONEBUS, _
    TPUDSParameter.PUDS_PARAM_API_VERSION, BufferString, 255)
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(BufferString.ToString())
End If
```

Pascal OO

```
var
  result: TPUDSStatus;
  BufferString: array [0..256] of Char;

begin
  // Get API version
  result := TUdsApi.GetValue(TUdsApi.PUDS_NONEBUS, PUDS_PARAM_API_VERSION, BufferString, 255);
  if (result <> PUDS_ERROR_OK) then
    MessageBox(0, 'Failed to get value', 'Error', MB_OK)
  else
    MessageBox(0, BufferString, 'Success', MB_OK);
end;
```

See also: SetValue on page 85, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_GetValue.

3.6.12 GetValue (TPUDSCANHandle, TPUDSParameter, Uint32, Uint32)

Retrieve information from a PUDS Channel in numeric form.

Syntax

Pascal OO

```
class function GetValue(
        CanChannel: TPUDSCANHandle;
        Parameter: TPUDSParameter;
        NumericBuffer: PLongWord;
        BufferLength: LongWord
): TPUDSStatus; overload;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
public static extern TPUDSStatus GetValue(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType.U1)]
        TPUDSParameter Parameter,
        out UInt32 NumericBuffer,
        UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
static TPUDSStatus GetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPUDSParameter Parameter,
        UInt32% NumericBuffer,
```

```
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_GetValue")> _
Public Shared Function GetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        ByRef NumericBuffer As UInt32, _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
NumericBuffer	The buffer to return the required numeric value
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application.
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with a string buffer

Example

The following example shows the use of the method GetValue on the channel PUDS_USBBUS1 to retrieve the ISO-TP separation time value (STmin). Depending on the result, a message will be shown to the user.

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

C#

```
TPCANTPHandle CanChannel = UDSApi.PUDS_USBBUS1;
TPUDSStatus result;
UInt32 iBuffer = 0;

// Get the value of the ISO-TP Separation Time (STmin) parameter
result = UDSApi.GetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SEPERATION_TIME,
out iBuffer, sizeof(UInt32));
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Failed to get value");
Else
    MessageBox.Show(iBuffer.ToString());
```

C++ / CLR

```
TPUDCANHandle CanChannel = UDSApi::PUDS_USBBUS1;
TPUDSStatus result;
UInt32 iBuffer = 0;

// Get the value of the ISO-TP Separation Time (STmin) parameter
result = CanTpApi::GetValue(CanChannel, PUDS_PARAM_SEPERATION_TIME, iBuffer,
sizeof(UInt32));
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Failed to get value");
else
      MessageBox::Show(iBuffer.ToString());
```

Visual Basic

```
Dim CanChannel As TPCANTPHandle = UDSApi.PUDS_USBBUS1
Dim result As TPUDSStatus
Dim iBuffer As UInt32 = 0

' Get the value of the ISO-TP Separation Time (STmin) parameter
result = UDSApi.GetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SEPERATION_TIME, _
    iBuffer, Convert.ToUInt32(Len(iBuffer)))
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(iBuffer.ToString())
End If
```

Pascal OO

```
var
  CanChannel: TPUDSCANHandle;
  result: TPUDSStatus;
  iBuffer: UINT;

begin
  CanChannel := TUdsApi.PUDS_USBBUS1;
  // Get the value of the ISO-TP Separation Time (STmin) parameter
  result := TUdsApi.GetValue(CanChannel, PUDS_PARAM_SEPERATION_TIME, PLongWord(@iBuffer),
sizeof(iBuffer));
  if (result <> PUDS_ERROR_OK) then
    MessageBox(0, 'Failed to get value', 'Error', MB_OK)
  else
    MessageBox(0, PAnsiChar(AnsiString(Format('STmin = %d', [Integer(iBuffer)]))), 'Success', MB_OK);
end;
```

See also: SetValue on page 85, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_GetValue.

3.6.13 GetValue (TPUDSCANHandle, TPUDSParameter, Byte[], UInt32)

Retrieve information from a PUDS Channel in a byte array.

Syntax

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
public static extern TPUDSStatus GetValue(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType.U1)]
        TPUDSParameter Parameter,
        [MarshalAs(UnmanagedType.LPArray)]
        [Out] Byte[] Buffer,
        UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
static TPUDSStatus GetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPUDSParameter Parameter,
        [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 3)]
        array<Byte>^ Buffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_GetValue")> _
Public Shared Function GetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPUDSParameter, _
        ByVal Buffer As Byte(), _
        ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
Buffer	The buffer containing the array value to retrieve
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPCANTPStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be used because it was not found in the list of reserved channels of the calling application.
POBDII_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with a string buffer

Example

The following example shows the use of the method GetValue on the channel PUDS_USBBUS1 to retrieve the ISO-TP separation time value (STmin). Depending on the result, a message will be shown to the user.

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

C#

```
TPUDSCANHandle CanChannel = CanTpApi.PUDS_USBBUS1;
TPUDSStatus result;
uint bufferLength = 2;
byte[] bufferArray = new byte[bufferLength];

// Get the value of the ISO-TP Separation Time (STmin) parameter
result = CanTpApi.GetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SEPERATION_TIME,
bufferArray, sizeof(byte) * bufferLength);
if (result != TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show("Failed to get value");
else
    MessageBox.Show(bufferArray[0].ToString());
```

C++ / CLR

```
TPUDSCANHandle CanChannel = CanTpApi::PUDS_USBBUS1;
TPUDSStatus result;
UInt32 bufferLength = 2;
array<Byte>^ bufferArray = gcnew array<Byte>(bufferLength);

// Get the value of the ISO-TP Separation Time (STmin) parameter
result = CanTpApi::GetValue(CanChannel, PUDS_PARAM_SEPERATION_TIME,
      bufferArray, sizeof(Byte) * bufferLength);
if (result != PUDS_ERROR_OK)
      MessageBox::Show("Failed to get value");
else
      MessageBox::Show(bufferArray->ToString());
```

Visual Basic

```
Dim CanChannel As TPCANTPHandle = UDSApi.PUDS_USBBUS1
Dim result As TPUDSStatus
Dim bufferLength As UInt32 = 2
Dim bufferArray(bufferLength) As Byte

' Get the value of the ISO-TP Separation Time (STmin) parameter
result = UDSApi.GetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SEPERATION_TIME, _
    bufferArray, Convert.ToUInt32(bufferLength)))
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    MessageBox.Show("Failed to get value")
Else
```

```
    MessageBox.Show(bufferArray(0).ToString())
End If
```

Pascal OO

```
var
  CanChannel: TPUDSCANHandle;
  result: TPUDSStatus;
  bufferArray: array [0..1] of Byte;

begin
  CanChannel := TUdsApi.PUDS_USBBUS1;

  // Get the value of the ISO-TP Separation Time (STmin) parameter
  result := TUdsApi.GetValue(CanChannel, PUDS_PARAM_SEPERATION_TIME, PLongWord(@bufferArray),
Length(bufferArray));
  if (result <> PUDS_ERROR_OK) then
    MessageBox(0, 'Failed to get value', 'Error', MB_OK)
  else
    MessageBox(0, PAnsiChar(AnsiString(Format('STmin = %d', [Integer(bufferArray[0])]))), 'Success', MB_OK);
end;
```

See also: SetValue on page 85, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_GetValue.

3.6.14 GetValue(TPUDSCANHandle, TPUDSParameter, IntPtr, UInt32)

Retrieve information from a PUDS Channel through a pointer.

Syntax

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
public static extern TPUDSStatus GetValue(
    [MarshalAs(UnmanagedType.U2)]
    TPUDSCANHandle CanChannel,
    [MarshalAs(UnmanagedType.U1)]
    TPUDSParameter Parameter,
    IntPtr Buffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetValue")]
static TPUDSStatus GetValue(
      [MarshalAs(UnmanagedType::U2)]
      TPUDSCANHandle CanChannel,
      [MarshalAs(UnmanagedType::U1)]
      TPUDSParameter Parameter,
      IntPtr Buffer,
      UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_GetValue")> _
Public Shared Function GetValue( _
    <MarshalAs(UnmanagedType.U2)> _
    ByVal CanChannel As TPUDSCANHandle, _
    <MarshalAs(UnmanagedType.U1)> _
    ByVal Parameter As TPUDSParameter, _
    ByVal Buffer As IntPtr, _
    ByVal BufferLength As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
Buffer	The buffer containing the array value to retrieve
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPCANTPStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be used because it was not found in the list of reserved channels of the calling application.
POBDII_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with a string buffer

Example

The following example shows the use of the method GetValue on the channel PUDS_USBBUS1 to retrieve the UDS Session Information. A console message will be written with the information retrieved.

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

C#:

```
TPCANTPHandle CanChannel = UDSApi.PUDS_USBBUS1;
TPUDSStatus result;
TPUDSSessionInfo sessionInfo;
int sessionSize;
IntPtr sessionPtr;

// define the session N_AI to retrieve
sessionInfo = new TPUDSSessionInfo();
sessionInfo.NETADDRINFO.SA = (byte)
TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
sessionInfo.NETADDRINFO.TA = (byte) TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
sessionInfo.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
sessionInfo.NETADDRINFO.RA = (byte) 0x00;
sessionInfo.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;
// Get pointer from structure
sessionSize = Marshal.SizeOf(sessionInfo);
sessionPtr = Marshal.AllocHGlobal(sessionSize);
Marshal.StructureToPtr(sessionInfo, sessionPtr, false);
// Get Session information
```

```
result = UDSApi.GetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SESSION_INFO,
sessionPtr, (uint) sessionSize);
// Get structure from pointer (note: if status is TPUDSStatus.PUDS_NOT_INITIALIZED,
default TIMEOUT values are still returned)
sessionInfo = (TPUDSSessionInfo)Marshal.PtrToStructure(sessionPtr,
typeof(TPUDSSessionInfo));
Console.WriteLine("Current session information = {0}", sessionInfo.SESSION_TYPE);
Console.WriteLine("  TIMEOUT_P2CAN_SERVER_MAX = {0}",
sessionInfo.TIMEOUT_P2CAN_SERVER_MAX);
// free ressource
Marshal.FreeHGlobal(sessionPtr);
```

C++/CLR:

```
TPUDSCANHandle CanChannel = UDSApi::PUDS_USBBUS1;
TPUDSStatus result;
TPUDSSessionInfo^ sessionInfo;
int sessionSize;
IntPtr sessionPtr;

// define the session N_AI to retrieve
sessionInfo = gcnew TPUDSSessionInfo();
sessionInfo->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
sessionInfo->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
sessionInfo->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
sessionInfo->NETADDRINFO.RA = 0x00;
sessionInfo->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;
// Get pointer from structure
sessionSize = Marshal::SizeOf(sessionInfo);
sessionPtr = Marshal::AllocHGlobal(sessionSize);
Marshal::StructureToPtr(sessionInfo, sessionPtr, false);
// Get Session information
result = UDSApi::GetValue(CanChannel, PUDS_PARAM_SESSION_INFO, sessionPtr,
sessionSize);
// Get structure from pointer (note: if status is TPUDSStatus.PUDS_NOT_INITIALIZED,
default TIMEOUT values are still returned)
sessionInfo = (TPUDSSessionInfo^)Marshal::PtrToStructure(sessionPtr,
TPUDSSessionInfo::typeid);
Console::WriteLine("Current session information = {0}", sessionInfo->SESSION_TYPE);
Console::WriteLine("  TIMEOUT_P2CAN_SERVER_MAX = {0}", sessionInfo-
>TIMEOUT_P2CAN_SERVER_MAX);
// free ressource
Marshal::FreeHGlobal(sessionPtr);
```

Visual Basic:

```
Dim CanChannel As TPCANTPHandle = UDSApi.PUDS_USBBUS1
Dim result As TPUDSStatus
Dim sessionInfo As TPUDSSessionInfo
Dim sessionSize as Integer
Dim sessionPtr as IntPtr

' Define the session N_AI to retrieve
sessionInfo = new TPUDSSessionInfo
sessionInfo.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
sessionInfo.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
sessionInfo.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
sessionInfo.NETADDRINFO.RA = &H0
sessionInfo.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B
' Get pointer from structure
sessionSize = Marshal.SizeOf(sessionInfo)
```

```
sessionPtr = Marshal.AllocHGlobal(sessionSize)
Marshal.StructureToPtr(sessionInfo, sessionPtr, false)
' Get Session information
result = UDSApi.GetValue(CanChannel, TPUDSParameter.PUDS_PARAM_SESSION_INFO,
sessionPtr, Convert.ToUInt32(sessionSize))
' Get structure from pointer (note: if status is TPUDSStatus.PUDS_NOT_INITIALIZED,
default TIMEOUT values are still returned)
sessionInfo = Marshal.PtrToStructure(sessionPtr, GetType(TPUDSSessionInfo))
Console.WriteLine("Current session information = {0}", sessionInfo.SESSION_TYPE)
Console.WriteLine("  TIMEOUT_P2CAN_SERVER_MAX = {0}",
sessionInfo.TIMEOUT_P2CAN_SERVER_MAX)
' free ressource
Marshal.FreeHGlobal(sessionPtr)
```

See also: SetValue on page 85, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_GetValue.

3.6.15 GetStatus

Gets the current BUS status of a PUDS Channel.

Syntax

Pascal OO

```
class function GetStatus(
        CanChannel: TPUDSCANHandle
): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetStatus")]
public static extern TPUDSStatus GetStatus(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_GetStatus")]
static TPUDSStatus GetStatus(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_GetStatus")> _
Public Shared Function GetStatus( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_OK	Indicates that the status of the given PUDS Channel is OK
PUDS_ERROR_BUSLIGHT	Indicates a bus error within the given PUDS Channel. The hardware is in bus-light status
PUDS_ERROR_BUSHEAVY:	Indicates a bus error within the given PUDS Channel. The hardware is in bus-heavy status
PUDS_ERROR_BUSOFF:	Indicates a bus error within the given PUDS Channel. The hardware is in bus-off status
PUDS_ERROR_NOT_INITIALIZED:	Indicates that the given PUDS channel cannot be uninitialized because it 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 uninitialize 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 PUDS_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
TPUDSStatus result;

// Check the status of the PCI Channel
result = CanTpApi.GetStatus(CanTpApi.PUDS_PCIBUS1);
switch (result)
{
    case TPUDSStatus.PUDS_ERROR_BUSLIGHT:
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...");
        break;
    case TPUDSStatus.PUDS_ERROR_BUSHEAVY:
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...");
        break;
    case TPUDSStatus.PUDS_ERROR_BUSOFF:
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...");
        break;
    case TPUDSStatus.PUDS_ERROR_OK:
        MessageBox.Show("PCAN-PCI (Ch-1): Status is OK");
        break;
    default:
        // An error occurred
        MessageBox.Show("Failed to retrieve status");
        break;
}
```

C++ / CLR

```
TPUDSStatus result;

// Check the status of the PCI Channel
```

```
result = CanTpApi::GetStatus(CanTpApi::PUDS_PCIBUS1);
switch (result)
{
case PUDS_ERROR_BUSLIGHT:
      MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...");
      break;
case PUDS_ERROR_BUSHEAVY:
      MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...");
      break;
case PUDS_ERROR_BUSOFF:
      MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...");
      break;
case PUDS_ERROR_OK:
      MessageBox::Show("PCAN-PCI (Ch-1): Status is OK");
      break;
default:
      // An error occurred);
      MessageBox::Show("Failed to retrieve status");
      break;
}
```

Visual Basic

```
Dim result As TPUDSStatus

' Check the status of the PCI Channel
result = CanTpApi.GetStatus(CanTpApi.PUDS_PCIBUS1)
Select Case (result)
    Case TPUDSStatus.PUDS_ERROR_BUSLIGHT
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...")
    Case TPUDSStatus.PUDS_ERROR_BUSHEAVY
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...")
    Case TPUDSStatus.PUDS_ERROR_BUSOFF
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...")
    Case TPUDSStatus.PUDS_ERROR_OK
        MessageBox.Show("PCAN-PCI (Ch-1): Status is OK")
    Case Else
        ' An error occurred
        MessageBox.Show("Failed to retrieve status")
End Select
```

Pascal OO

```
var
  result: TPUDSStatus;

begin

  // Check the status of the PCI Channel
  result := TCanTpApi.GetStatus(TCanTpApi.PUDS_PCIBUS1);
  Case (result) of
    PUDS_ERROR_BUSLIGHT:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...', 'Error', MB_OK);
    PUDS_ERROR_BUSHEAVY:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...', 'Error', MB_OK);
    PUDS_ERROR_BUSOFF:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-OFF status...', 'Error', MB_OK);
    PUDS_ERROR_OK:
```

```
        MessageBox(0, 'PCAN-PCI (Ch-1): Status is OK', 'Error', MB_OK);
    else
        // An error occurred);
        MessageBox(0, 'Failed to retrieve status', 'Error', MB_OK);
    end;
end;
```

See also: TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Plain function version: UDS_GetStatus.

3.6.16 Read

Reads a CAN UDS message from the receive queue of a PUDS Channel.

Syntax

Pascal OO

```
class function Read(
        CanChannel: TPUDSCANHandle;
        var MessageBuffer: TPUDSPMsg
): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Read")]
public static extern TPUDSStatus Read(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel,
        out TPUDSMsg MessageBuffer);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Read")]
static TPUDSStatus Read(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        TPUDSMsg %MessageBuffer);
```

Visual basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_Read")> _
Public Shared Function Read( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        ByRef MessageBuffer As TPUDSMsg) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Message Buffer	A TPUDSMsg buffer to store the CAN UDS message

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NO_MESSAGE	Indicates that the receive queue of the Channel is empty
PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application

Remarks: The message type (see TPUDSMessageType) of a CAN UDS message indicates if the message is a complete received UDS message, a transmission confirmation or an indication of a pending message. This value should be checked every time a message has been read successfully, along with the RESULT value as it contains the network status of the message.

Note: That higher level functions like WaitForSingleMessage or WaitForService should be preferred in cases where a client just has to read the response from a service request.

Example

The following example shows the use of the method Read on the channel PUDS_USBBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
TPUDSStatus result;
TPUDSMsg msg;
bool bStop = false;

do
{
    // Read the first message in the queue
    result = UDSApi.Read(UDSApi.PUDS_USBBUS1, out msg);
    if (result == TPUDSStatus.PUDS_ERROR_OK)
    {
        // Process the received message
        MessageBox.Show("A message was received");
        ProcessMessage(msg);
    }
    else
    {
        // An error occurred
        MessageBox.Show("An error ocured");
        // Here can be decided if the loop has to be terminated
        bStop = HandleReadError(result);
    }
} while (!bStop);
```

C++ / CLR

```
TPUDSStatus result;
TPUDSMsg msg;
bool bStop = false;

do
{
	// Read the first message in the queue
	result = UDSApi::Read(UDSApi::PUDS_USBBUS1, msg);
	if (result == PUDS_ERROR_OK)
	{
		// Process the received message
		MessageBox::Show("A message was received");
		//ProcessMessage(msg);
	}
	else
	{
		// An error occurred
		MessageBox::Show("An error ocured");
		// Here can be decided if the loop has to be terminated
		//bStop = HandleReadError(result);
	}
} while (!bStop);
```

Visual Basic

```
Dim result As TPUDSStatus
Dim msg As TPUDSMsg
Dim bStop As Boolean = False

Do
    ' Read the first message in the queue
    msg = New TPUDSMsg()
    result = UDSApi.Read(UDSApi.PUDS_USBBUS1, msg)
    If result = TPUDSStatus.PUDS_ERROR_OK Then
        ' Process the received message
        MessageBox.Show("A message was received")
        ProcessMessage(msg)
    Else
        ' An error occurred
        MessageBox.Show("An error ocured")
        ' Here can be decided if the loop has to be terminated
        bStop = HandleReadError(result)
    End If
Loop While bStop = False
```

Pascal OO

```
var
  result: TPUDSStatus;
  msg: TPUDSMsg;
  bStop: Boolean;

begin
  bStop := False;
  repeat
    // Read the first message in the queue
    result := TUdsApi.Read(TUdsApi.PUDS_USBBUS1, msg);
```

```
    if (result = PUDS_ERROR_OK) then
    begin
      // Process the received message
      MessageBox(0, 'A message was received', 'Error', MB_OK);
      ProcessMessage(msg);
    end
    else
    begin
      // An error occurred
      MessageBox(0, 'An error ocured', 'Error', MB_OK);
      // Here can be decided if the loop has to be terminated
      bStop = HandleReadError(result);
    end;

  until (bStop = true);
end;
```

See also: Write below.

Plain function version: UDS_Read.

3.6.17 Write

Transmits a CAN UDS message.

Syntax

Pascal OO

```
class function Write(
        CanChannel: TPUDSCANHandle;
        var MessageBuffer: TPUDSMsg
): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Write")]
public static extern TPUDSStatus Write(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel,
        ref TPUDSMsg MessageBuffer);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Write")]
static TPUDSStatus Write(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel,
        TPUDSMsg %MessageBuffer);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_Write")> _
Public Shared Function Write( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle, _
        ByRef MessageBuffer As TPUDSMsg) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Message Buffer	A TPUDSMsg buffer containing the CAN UDS message to be sent

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application or that a required CAN ID mapping was not found
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The Write function do not actually send the UDS message, the transmission is asynchronous. Should a message fail to be transmitted, it will be added to the reception queue with a specific network error code in the RESULT value of the TPUDSMsg.

Note: To transmit a standard UDS service request, it is recommended to use the corresponding API Service method starting with Svc (like `SvcDiagnosticSessionControl`).

Example

The following example shows the use of the method Write on the channel PUDS_USBBUS1. It adds to the transmit queue a UDS request from source 0xF1 to ECU #1 and then waits until a confirmation message is received. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel was already initialized and mapping were configured, the content of DATA is not initialized in the example.

C#

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();

// prepare an 11bit CAN ID, physically addressed UDS  message containing 4095 Bytes
of data
request.DATA = new byte[4095];
// [...] fill data
request.LEN = (ushort)request.DATA.Length;
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST;

request.NETADDRINFO.SA = (byte) TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte) TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
```

```
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDSApi.Write(UDSApi.PUDS_USBBUS1, ref request);
if (result == TPUDSStatus.PUDS_ERROR_OK)
{
    // Loop until the transmission confirmation is received
    do
    {
        result = UDSApi.Read(UDSApi.PUDS_USBBUS1, out request);
        MessageBox.Show(String.Format("Read = {0}, type={1}, result={2}", result,
request.MSGTYPE, request.RESULT));
    } while (result == TPUDSStatus.PUDS_ERROR_NO_MESSAGE);
}
else
{
    // An error occurred
    MessageBox.Show("Error occured: " + result.ToString());
}
```

C++ / CLR

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();

// prepare an 11bit CAN ID, physically addressed UDS message containing 4095 Bytes
of data
request->DATA = gcnew array<Byte>(4095);
// [...] fill data
request->LEN = (unsigned short)request->DATA->Length;
request->MSGTYPE = PUDS_MESSAGE_TYPE_REQUEST;

request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDSApi::Write(UDSApi::PUDS_USBBUS1, *request);
if (result == PUDS_ERROR_OK)
{
      // Loop until the transmission confirmation is received
      do
      {
            result = UDSApi::Read(UDSApi::PUDS_USBBUS1, *request);
            MessageBox::Show(String::Format("Read = {0}, type={1}, result={2}",
((int)result).ToString(), ((int)request->MSGTYPE).ToString(), ((int)request-
>RESULT).ToString()));
      } while (result == PUDS_ERROR_NO_MESSAGE);
}
else
{
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
}
```

Visual Basic

```
Dim request As TPUDSMsg = New TPUDSMsg()
Dim result As TPUDSStatus

' prepare an 11bit CAN ID, physically addressed UDS message containing 4095 bytes
of data
request.DATA = New Byte(4095) {}
' [...] fill data
request.LEN = request.DATA.Length
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST

request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' The message is sent using the PCAN-USB
result = UDSApi.Write(UDSApi.PUDS_USBBUS1, request)
If result = TPUDSStatus.PUDS_ERROR_OK Then
    ' Loop until the transmission confirmation is received
    Do
        result = UDSApi.Read(UDSApi.PUDS_USBBUS1, request)
        MessageBox.Show(String.Format("Read = {0}, type={1}, result={2}", result,
request.MSGTYPE, request.RESULT))
    Loop While result = TPUDSStatus.PUDS_ERROR_NO_MESSAGE
Else
    ' An error occurred
    MessageBox.Show("Error occured: " + result.ToString())
End If
```

Pascal OO

```
var
  request: TPUDSMsg;
  result: TPUDSStatus;

begin
  // prepare an 11bit CAN ID, physically addressed UDS message containing 4095 bytes of data
  // [...] fill data
  request.LEN := Length(request.DATA);
  request.MSGTYPE := PUDS_MESSAGE_TYPE_REQUEST;

  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // The message is sent using the PCAN-USB
  result := TUdsApi.Write(TUdsApi.PUDS_USBBUS1, request);
  if (result = PUDS_ERROR_OK) then
  begin
    // Loop until the transmission confirmation is received
    repeat
      result := TUdsApi.Read(TUdsApi.PUDS_USBBUS1, request);
```

```
        MessageBox(0, PAnsiChar(AnsiString(
          Format('Read = %d, type=%d, result=%d',
          [Integer(result), Integer(request.MSGTYPE), Integer(request.RESULT)]))), 'Error', MB_OK);
    until (result = PUDS_ERROR_NO_MESSAGE);
  end
  else
    // An error occurred
    MessageBox(0, PAnsiChar(AnsiString(Format('Error occured = %d', [Integer(result)]))), 'Error', MB_OK);
end;
```

See also: Read on page 108.

Plain function version: UDS_Read.

3.6.18 Reset

Resets the receive and transmit queues of a PCUDS Channel.

Syntax

Pascal OO

```
class function Reset(
        CanChannel: TPUDSCANHandle
): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Reset")]
public static extern TPUDSStatus Reset(
        [MarshalAs(UnmanagedType.U2)]
        TPUDSCANHandle CanChannel);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_Reset")]
static TPUDSStatus Reset(
        [MarshalAs(UnmanagedType::U2)]
        TPUDSCANHandle CanChannel);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_Reset")> _
Public Shared Function Reset( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPUDSCANHandle) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized 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.

Example

The following example shows the use of the method Reset on the channel PUDS_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
TPUDSStatus result;

// The PCI Channel is reset
result = UDSApi.Reset(UDSApi.PUDS_PCIBUS1);
if (result != TPUDSStatus.PUDS_ERROR_OK)
{
    // An error occurred
    MessageBox.Show("An error occured");
}
else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset");
```

C++ / CLR

```
TPUDSStatus result;

// The PCI Channel is reset
result = UDSApi::Reset(UDSApi::PUDS_PCIBUS1);
if (result != PUDS_ERROR_OK)
{
      // An error occurred
      MessageBox::Show("An error occured");
}
else
      MessageBox::Show("PCAN-PCI (Ch-1) was reset");
```

Visual Basic

```
Dim result As TPUDSStatus

' The PCI Channel is reset
result = UDSApi.Reset(UDSApi.PUDS_PCIBUS1)
If result <> TPUDSStatus.PUDS_ERROR_OK Then
    ' An error occurred
    MessageBox.Show("An error occured")
Else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset")
End If
```

Pascal OO

```
var
  result: TPUDSStatus;

begin
  // The PCI Channel is reset
  result := TUdsApi.Reset(TUdsApi.PUDS_PCIBUS1);
  if (result <> PUDS_ERROR_OK) then
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
  else
    MessageBox(0, 'PCAN-PCI (Ch-1) was reset', 'Error', MB_OK);
end;
```

See also: Uninitialize on page 82.

Plain function version: UDS_Reset.

3.6.19 WaitForSingleMessage

Waits for a UDS response or transmit confirmation based on a UDS request.

Syntax

Pascal OO

```
class function WaitForSingleMessage(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    var MessageRequest: TPUDSMsg;
    IsWaitForTransmit: Boolean;
    TimeInterval: LongWord;
    Timeout: LongWord): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForSingleMessage")]
public static extern TPUDSStatus WaitForSingleMessage(
    [MarshalAs(UnmanagedType.U1)]
    TPUDSCANHandle CanChannel,
    out TPUDSMsg MessageBuffer,
    ref TPUDSMsg MessageRequest,
    bool IsWaitForTransmit,
    UInt32 TimeInterval,
    UInt32 Timeout);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForSingleMessage")]
static TPUDSStatus WaitForSingleMessage(
      [MarshalAs(UnmanagedType::U1)]
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSMsg %MessageRequest,
      bool IsWaitForTransmit,
      UInt32 TimeInterval,
```

```
      UInt32 Timeout);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_WaitForSingleMessage")> _
Public Shared Function WaitForSingleMessage( _
    <MarshalAs(UnmanagedType.U1)> _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByRef MessageRequest As TPUDSMsg, _
    ByVal IsWaitForTransmit As Boolean, _
    ByVal TimeInterval As UInt32, _
    ByVal Timeout As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Message Buffer	A TPUDSMsg buffer cto store the UDS message
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously sent
IsWaitForTransmit	States whether the message to wait is a transmit confirmation or a UDS response
TimeInterval	Time to wait between polling for new UDS messages in milliseconds
Timeout	Maximum time to wait (in milliseconds) for a message indication corresponding to the MessageRequest. Note: a zero value means unlimited time

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel

Remarks: The Timeout parameter is ignored once a message indication matching the request is received (i.e. the first frame of the message). The function will then return once the message is fully received or a network error occurred.

To prevent unexpected locking, the user can abort the function by calling the function UDS_Reset (**class-method:** Reset).

Note: that the criteria to identify if a response matches the message request is based only on the network addressing information and the UDS service identifier: if a same service is requested multiple times with different parameters (like service ReadDataByIdentifier with different Data IDs), the user will have to ensure that the extra content matches the original request.

Example

The following example shows the use of the method WaitForSingleMessage on the channel PUDS_USBBUS1. It writes a UDS message on the CAN Bus and waits for the confirmation of the transmition. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg confirmation = new TPUDSMsg();

// prepare an 11bit CAN ID, physically addressed UDS  message containing 4095 Bytes
of data
request.DATA = new byte[4095];
// [...] fill data
request.LEN = (ushort)request.DATA.Length;
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST;
request.LEN = (ushort)request.DATA.Length;
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST;

request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDSApi.Write(UDSApi.PUDS_USBBUS1, ref request);
if (result == TPUDSStatus.PUDS_ERROR_OK)
{
   // wait for the transmit confirmation
   result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, out confirmation, ref
request, true, 10, 100);
   if (result == TPUDSStatus.PUDS_ERROR_OK)
       MessageBox.Show(String.Format("Message was transmitted."));
   else
       // An error occurred
       MessageBox.Show(String.Format("Error occured while waiting for transmit
confirmation: {0}", (int)result));
}
else
{
   // An error occurred
   MessageBox.Show(String.Format("Error occured: {0}", (int)result));
}
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ confirmation = gcnew TPUDSMsg();

// prepare an 11bit CAN ID, physically addressed UDS  message containing 4095 Bytes
of data
request->DATA = gcnew array<Byte>(4095);
// [...] fill data
request->LEN = (unsigned short)request->DATA->Length;
request->MSGTYPE = PUDS_MESSAGE_TYPE_REQUEST;

request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
```

```
result = UDSApi::Write(UDSApi::PUDS_USBBUS1, *request);
if (result == PUDS_ERROR_OK)
{
      // wait for the transmit confirmation
      result = UDSApi::WaitForSingleMessage(UDSApi::PUDS_USBBUS1, *confirmation,
*request, true, 10, 100);
      if (result == PUDS_ERROR_OK)
            MessageBox::Show(String::Format("Message was transmitted."));
      else
            // An error occurred
            MessageBox::Show(String::Format("Error occured while waiting for
transmit confirmation: {0}", (int)result));
}
else
{
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
}
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim confirmation As TPUDSMsg = New TPUDSMsg()

' prepare an 11bit CAN ID, physically addressed UDS  message containing 4095 Bytes
of data
request.DATA = New Byte(4095) {}
' [...] fill data
request.LEN = request.DATA.Length
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST

request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' The message is sent using the PCAN-USB
result = UDSApi.Write(UDSApi.PUDS_USBBUS1, request)
if (result = TPUDSStatus.PUDS_ERROR_OK) then
    ' wait for the transmit confirmation
    result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, confirmation,
request, true, 10, 100)
    if (result = TPUDSStatus.PUDS_ERROR_OK) then
        MessageBox.Show(String.Format("Message was transmitted."))
    else
        ' An error occurred
        MessageBox.Show(String.Format("Error occured while waiting for transmit
confirmation: {0}", result.ToString()))
    end if

Else
    ' An error occurred
    MessageBox.Show("Error occured: " + result.ToString())
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  confirmation: TPUDSMsg;
  result: TPUDSStatus;

begin
  // prepare an 11bit CAN ID, physically addressed UDS message containing 4095 bytes of data
  // […] fill data
  request.LEN := Length(request.DATA);
  request.MSGTYPE := PUDS_MESSAGE_TYPE_REQUEST;

  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B  ;

  // The message is sent using the PCAN-USB
  result := TUdsApi.Write(TUdsApi.PUDS_USBBUS1, request);
  if (result = PUDS_ERROR_OK) then
  begin
    // wait for the transmit confirmation
    result := TUdsApi.WaitForSingleMessage(TUdsApi.PUDS_USBBUS1, confirmation, request, true, 10, 100);
    if (result = PUDS_ERROR_OK) then
      MessageBox(0, 'Message transmitted', 'Success', MB_OK)
    else
      // An error occurred
      MessageBox(0, 'Error occured while waiting for transmit confirmation', 'Error', MB_OK);
  end
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: Write on page 111.

Plain function version: UDS_WaitForSingleMessage.

3.6.20 WaitForMultipleMessage

Waits for multiple UDS responses based on a UDS request (multiple responses can be obtained when a functional UDS request is transmitted).

Syntax

Pascal OO

```
class function WaitForMultipleMessage(
    CanChannel: TPUDSCANHandle;
    Buffer: PTPUDSMsg;
    MaxCount: LongWord;
    pCount: PLongWord;
```

```
    var MessageRequest: TPUDSMsg;
    TimeInterval: LongWord;
    Timeout: LongWord;
    TimeoutEnhanced: LongWord;
    WaitUntilTimeout: Boolean): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForMultipleMessage")]
public static extern TPUDSStatus WaitForMultipleMessage(
    [MarshalAs(UnmanagedType.U1)]
    TPUDSCANHandle CanChannel,
    [In, Out]
    TPUDSMsg[] Buffer,
    UInt32 MaxCount,
    out UInt32 pCount,
    ref TPUDSMsg MessageRequest,
    UInt32 TimeInterval,
    UInt32 Timeout,
    UInt32 TimeoutEnhanced,
    bool WaitUntilTimeout);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForMultipleMessage")]
static TPUDSStatus WaitForMultipleMessage(
      [MarshalAs(UnmanagedType::U1)]
      TPUDSCANHandle CanChannel,
      array<TPUDSMsg>^ Buffer,
      UInt32 MaxCount,
      UInt32 %pCount,
      TPUDSMsg %MessageRequest,
      UInt32 TimeInterval,
      UInt32 Timeout,
      UInt32 TimeoutEnhanced,
      bool WaitUntilTimeout);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_WaitForMultipleMessage")> _
Public Shared Function WaitForMultipleMessage( _
    <MarshalAs(UnmanagedType.U1)> _
    ByVal CanChannel As TPUDSCANHandle, _
    <[In](), Out()> ByVal Buffer As TPUDSMsg(), _
    ByVal MaxCount As UInt32, _
    ByRef pCount As UInt32, _
    ByRef MessageRequest As TPUDSMsg, _
    ByVal TimeInterval As UInt32, _
    ByVal Timeout As UInt32, _
    ByVal TimeoutEnhanced As UInt32, _
    ByVal WaitUntilTimeout As Boolean) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Buffer	A buffer to store an array of TPUDSMsge
MaxCount	The maximum number of expected responses
pCount	Buffer to store the actual number of received responses
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously sent.
TimeInterval	Time to wait between polling for new UDS messages in milliseconds.
Timeout	Maximum time to wait (in milliseconds) for a message indication corresponding to the MessageRequest. Note: a zero value means unlimited time.
TimeoutEnhanced	Maximum time to wait (in milliseconds) for a message indication corresponding to the MessageRequest if an ECU asks for extended timing (NRC_EXTENDED_TIMING)
WaitUntilTimeout	States whether the function is interrupted if the number of received messages reaches MaxCount

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_OVERFLOW	Success but the buffer limit was reached. Check pCount variable to see how many messages were discarded

Remarks: For each response, the Timeout/TimeoutEnhanced parameter is ignored once a message indication matching the request is received (i.e. the first frame of the message). The function will then return once all messages are fully received or a network error occurred.

To prevent unexpected locking, the user can abort the function by calling the function UDS_Reset (**class-method:** Reset).

Note: That the criteria to identify if a response matches the message request is based only on the network addressing information and the UDS service identifier: if a same service is requested multiple times with different parameters (like service ReadDataByIdentifier with different Data IDs), the user will have to ensure that the extra content matches the original request.

The function handles the negative response code `PUDS_NRC_EXTENDED_TIMING` (0x78) in order to fetch all responses at the same time: if such a response is read, the function will switch the default timeout to TimeoutEnhanced and wait for a new response.

Example

The following example shows the use of the method WaitForMultipleMessage on the channel PUDS_ USBBUS1. It writes a UDS functional message on the CAN Bus, waits for the confirmation of the transmission and then waits to receive responses from ECUs until timeout occurs. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg confirmation = new TPUDSMsg();
UInt32 MessageArraySize = 5;
TPUDSMsg[] MessageArray = new TPUDSMsg[MessageArraySize];
uint count = 0;

// prepare an 11bit CAN ID, functionally addressed UDS message
request.DATA = new byte[4095];
// [...] fill data (functional message is limited to 1 CAN frame)
request.LEN = 7;
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST;

request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_FUNCTIONAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDSApi.Write(UDSApi.PUDS_USBBUS1, ref request);
if (result == TPUDSStatus.PUDS_ERROR_OK)
{
    // wait for the transmit confirmation
    result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, out confirmation, ref
request, true, 10, 100);
    if (result == TPUDSStatus.PUDS_ERROR_OK && confirmation.RESULT ==
TPUDSResult.PUDS_RESULT_N_OK)
    {
        MessageBox.Show(String.Format("Message was transmitted."));
        // wait for the responses
        result = UDSApi.WaitForMultipleMessage(UDSApi.PUDS_USBBUS1, MessageArray,
MessageArraySize, out count, ref request, 10, 100, 1000, true);
        MessageBox.Show(String.Format("Received messages count = {0}.", count));
    }
    else
        // An error occurred
        MessageBox.Show(String.Format("Error occured while waiting for transmit
confirmation: {0}", (int)result));
}
else
{
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
}
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ confirmation = gcnew TPUDSMsg();
UInt32 MessageArraySize = 5;
array<TPUDSMsg>^ MessageArray = gcnew array<TPUDSMsg>(MessageArraySize);
UInt32 count;


// prepare an 11bit CAN ID, functionnaly addressed UDS message
request->DATA = gcnew array<Byte>(4095);
// [...] fill data (functional message is limited to 1 CAN frame)
```

```
request->LEN = (unsigned short)7;
request->MSGTYPE = PUDS_MESSAGE_TYPE_REQUEST;

request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDSApi::Write(UDSApi::PUDS_USBBUS1, *request);
if (result == PUDS_ERROR_OK)
{
      // wait for the transmit confirmation
      result = UDSApi::WaitForSingleMessage(UDSApi::PUDS_USBBUS1, *confirmation,
*request, true, 10, 100);
    if (result == PUDS_ERROR_OK && confirmation->RESULT == PUDS_RESULT_N_OK)
    {
        MessageBox::Show(String::Format("Message was transmitted."));
        // wait for the responses
        result = UDSApi::WaitForMultipleMessage(UDSApi::PUDS_USBBUS1, MessageArray,
MessageArraySize, count, *request, 10, 100, 1000, true);
        MessageBox::Show(String::Format("Received messages count = {0}.", count));
    }
      else
            // An error occurred
            MessageBox::Show(String::Format("Error occured while waiting for
transmit confirmation: {0}", (int)result));
}
else
{
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
}
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim confirmation As TPUDSMsg = New TPUDSMsg()
Dim MessageArraySize As UInt32 = 5
Dim MessageArray As TPUDSMsg() = New TPUDSMsg(MessageArraySize) {}
Dim count As UInt32

' prepare an 11bit CAN ID, physically addressed UDS message
request.DATA = New Byte(4095) {}
' [...] fill data (functional message is limited to 1 CAN frame)
request.LEN = 7
request.MSGTYPE = TPUDSMessageType.PUDS_MESSAGE_TYPE_REQUEST

request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_FUNCTIONAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' The message is sent using the PCAN-USB
result = UDSApi.Write(UDSApi.PUDS_USBBUS1, request)
if (result = TPUDSStatus.PUDS_ERROR_OK) then
    ' wait for the transmit confirmation
```

```
    result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, confirmation,
request, true, 10, 100)
    if (result = TPUDSStatus.PUDS_ERROR_OK and confirmation.RESULT =
TPUDSResult.PUDS_RESULT_N_OK) then
        MessageBox.Show(String.Format("Message was transmitted."))
        ' wait for the responses
        result = UDSApi.WaitForMultipleMessage(UDSApi.PUDS_USBBUS1, MessageArray,
MessageArraySize, count, request, 10, 100, 1000, true)
        MessageBox.Show(String.Format("Received messages count = {0}.", count))
    else
        ' An error occurred
        MessageBox.Show(String.Format("Error occured while waiting for transmit
confirmation: {0}", result.ToString()))
    end if

Else
    ' An error occurred
    MessageBox.Show("Error occured: " + result.ToString())
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  confirmation: TPUDSMsg;
  result: TPUDSStatus;
  MessageArraySize: LongWord;
  MessageArray: array[0..4] of TPUDSMsg;
  count: LongWord;

begin
  MessageArraySize := 5;
  // prepare an 11bit CAN ID, functionnaly addressed UDS message
  // […] fill data data (functional message is limited to 1 CAN frame)
  request.LEN := 7;
  request.MSGTYPE := PUDS_MESSAGE_TYPE_REQUEST;

  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_FUNCTIONAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B  ;

  // The message is sent using the PCAN-USB
  result := TUdsApi.Write(TUdsApi.PUDS_USBBUS1, request);
  if (result = PUDS_ERROR_OK) then
  begin
    // wait for the transmit confirmation
    result := TUdsApi.WaitForSingleMessage(TUdsApi.PUDS_USBBUS1, confirmation, request, true, 10, 100);
    if (result = PUDS_ERROR_OK) and (confirmation.RESULT = PUDS_RESULT_N_OK) then
    begin
      MessageBox(0, 'Message transmitted', 'Success', MB_OK);
      // wait for the responses
      result := TUdsApi.WaitForMultipleMessage(TUdsApi.PUDS_USBBUS1, PTPUDSMsg(@MessageArray),
MessageArraySize, PLongWord(@count), request, 10, 100, 1000, true);
```

```
        if (result = PUDS_ERROR_OK) then
          MessageBox(0, 'Received messages', 'Success', MB_OK)
        else
          // An error occurred
          MessageBox(0, 'An error occured', 'Error', MB_OK);
      end
      else
        // An error occurred
        MessageBox(0, 'Error occured while waiting for transmit confirmation', 'Error', MB_OK);
    end
    else
      // An error occurred
      MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: Write on page 111, WaitForSingleMessageon page 117.

Plain function version: UDS_WaitForSingleMessage.

3.6.21 WaitForService

Handles the communication workflow for a UDS service expecting a single response. The function waits for a transmit confirmation then for a message response.

Syntax

Pascal OO

```
class function WaitForService(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    var MessageRequest: TPUDSMsg;
    MessageReqBuffer: PTPUDSMsg = nil): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForService")]
public static extern TPUDSStatus WaitForService(
    [MarshalAs(UnmanagedType.U1)]
    TPUDSCANHandle CanChannel,
    out TPUDSMsg MessageBuffer,
    ref TPUDSMsg MessageRequest,
    out TPUDSMsg MessageReqBuffer);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForService")]
static TPUDSStatus WaitForService(
      [MarshalAs(UnmanagedType::U1)]
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSMsg %MessageRequest,
      TPUDSMsg %MessageReqBuffer);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_WaitForService")> _
```

```
Public Shared Function WaitForService( _
    <MarshalAs(UnmanagedType.U1)> _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByRef MessageRequest As TPUDSMsg, _
    ByRef MessageReqBuffer As TPUDSMsg) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Message Buffer	A TPUDSMsg buffer to store the UDS response
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously sent.
MessageReqBuffer	A TPUDSMsg buffer to store the UDS transmit confirmation

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_CAN_ERROR	A network error occured either in the transmit confirmation or the response message

Remarks: The WaitForService function is a utility function that calls other UDS API functions to simplify UDS communication workflow:

- The function gets the defined timeouts (global API request and response timeouts and timeouts defined by the current session information with the ECU),
- Waits for the confirmation of request's transmition,
- On success, waits for the response confirmation.
- If a negative response code is received stating that the ECU requires extended timing (`PUDS_NRC_EXTENDED_TIMING`, 0x78), the function switches to the enhanced timeout and waits for another response.
- The function ProcessResponse is called automatically on the response.

Even if the SuppressPositiveResponseMessage flag is set in the UDS request, the function will still wait for an eventual Negative Response. If no error message is received the function will return PUDS_ERROR_NO_MESSAGE, although in this case it is must not be considered as an error. Moreover if a negative response code `PUDS_NRC_EXTENDED_TIMING` is received the SuppressPositiveResponseMessage flag is ignored as stated in ISO-14229-1.

Example

The following example shows the use of the method WaitForService on the channel PUDS_USBBUS1. A UDS physical service request is transmitted (service ECUReset), and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C# :

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Message
result = UDSApi.SvcECUReset(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamER.PUDS_SVC_PARAM_ER_SR);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();

// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Message
result = UDSApi::SvcECUReset(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamER::PUDS_SVC_PARAM_ER_SR);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
    // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()

' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
```

```
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ECUReset Message
result = UDSApi.SvcECUReset(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamER.PUDS_SVC_PARAM_ER_SR)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B  ;

  // Sends a Physical ECUReset Message
  result := TUdsApi.SvcECUReset(TUdsApi.PUDS_USBBUS1, request, PUDS_SVC_PARAM_ER_SR);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForServiceFunctional below.

Plain function version: UDS_WaitForService.

3.6.22 WaitForServiceFunctional

Handles the communication workflow for a UDS service requested with functional addressing, i.e. multiple responses can be expected. The function waits for a transmit confirmation then for responses.

Syntax

Pascal OO

```
class function WaitForServiceFunctional(
    CanChannel: TPUDSCANHandle;
    Buffer: PTPUDSMsg;
    MaxCount: LongWord;
    pCount: PLongWord;
    WaitUntilTimeout: Boolean;
    var MessageRequest: TPUDSMsg;
    MessageReqBuffer: PTPUDSMsg = nil): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForServiceFunctional")]
public static extern TPUDSStatus WaitForServiceFunctional(
    [MarshalAs(UnmanagedType.U1)]
    TPUDSCANHandle CanChannel,
    [In, Out]
    TPUDSMsg[] Buffer,
    UInt32 MaxCount,
    out UInt32 pCount,
    bool WaitUntilTimeout,
    ref TPUDSMsg MessageRequest,
    out TPUDSMsg MessageReqBuffer);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_WaitForServiceFunctional")]
static TPUDSStatus WaitForServiceFunctional(
      [MarshalAs(UnmanagedType::U1)]
      TPUDSCANHandle CanChannel,
      array<TPUDSMsg>^ Buffer,
      UInt32 MaxCount,
      UInt32 %pCount,
      bool WaitUntilTimeout,
      TPUDSMsg %MessageRequest,
      TPUDSMsg %MessageReqBuffer);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_WaitForServiceFunctional")> _
Public Shared Function WaitForServiceFunctional( _
    <MarshalAs(UnmanagedType.U1)> _
    ByVal CanChannel As TPUDSCANHandle, _
    <[In](), Out()> ByVal Buffer As TPUDSMsg(), _
    ByVal MaxCount As UInt32, _
    ByRef pCount As UInt32, _
    ByVal WaitUntilTimeout As Boolean, _
    ByRef MessageRequest As TPUDSMsg, _
    ByRef MessageReqBuffer As TPUDSMsg) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Buffer	A buffer to store an array of TPUDSMsg
MaxCount	The maximum number of expected responses

Parameters	Description
pCount	Buffer to store the actual number of received responses
WaitUntilTimeout	States whether the function is interrupted if the number of received messages reaches MaxCount
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously written
MessageReqBuffer	A TPUDSMsg buffer to store the UDS transmit confirmation

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_OVERFLOW	Success but the buffer limit was reached. Check pCount variable to see how many messages were discarded

Remarks: The WaitForServiceFunctional function is a utility function that calls other UDS API functions to simplify UDS communication workflow when requests involve functional addressing.

- The function gets the defined timeouts (global API request and response timeouts and timeouts defined by the current session information with the ECU),
- Waits for the confirmation of request's transmition.
- On success, it waits for the confirmations of the responses like the function WaitForMultipleMessage would.

The function automatically calls `ProcessResponse` on each received message.

Example

The following example shows the use of the method WaitForServiceFunctional on the channel PUDS_USBBUS1. A UDS functional service request is transmitted (service ECUReset), and the WaitForServiceFunctional function is called to get the responses. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
UInt32 MessageArraySize = 5;
TPUDSMsg[] MessageArray = new TPUDSMsg[MessageArraySize];
uint count = 0;
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_FUNCTIONAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Functional ECUReset Message
result = UDSApi.SvcECUReset(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamER.PUDS_SVC_PARAM_ER_SR);
```

```
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForServiceFunctional(UDSApi.PUDS_USBBUS1, MessageArray,
MessageArraySize, out count, true, ref request, out requestConfirmation);
if (count > 0)
    MessageBox.Show(String.Format("Received messages count = {0}.", count));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
UInt32 MessageArraySize = 5;
array<TPUDSMsg>^ MessageArray = gcnew array<TPUDSMsg>(MessageArraySize);
UInt32 count;

// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Message
result = UDSApi::SvcECUReset(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamER::PUDS_SVC_PARAM_ER_SR);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForServiceFunctional(UDSApi::PUDS_USBBUS1, MessageArray,
MessageArraySize, count, true, *request, *requestConfirmation);
if (count > 0)
      MessageBox::Show(String::Format("Received messages count = {0}.", count));

else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim MessageArraySize As UInt32 = 5
Dim MessageArray As TPUDSMsg() = New TPUDSMsg(MessageArraySize) {}
Dim count As UInt32

' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_FUNCTIONAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Functional ECUReset Message
result = UDSApi.SvcECUReset(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamER.PUDS_SVC_PARAM_ER_SR)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForServiceFunctional(UDSApi.PUDS_USBBUS1, MessageArray,
MessageArraySize, count, true, request, requestConfirmation)
End If
```

```
If (count > 0) Then
    MessageBox.Show(String.Format("Received messages count = {0}.", count))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  MessageArraySize: LongWord;
  MessageArray: array[0..4] of TPUDSMsg;
  count: LongWord;

begin
  // initialization
  MessageArraySize := 5;
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B  ;

  // Sends a Physical ECUReset Message
  result := TUdsApi.SvcECUReset(TUdsApi.PUDS_USBBUS1, request, PUDS_SVC_PARAM_ER_SR);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForServiceFunctional(TUdsApi.PUDS_USBBUS1, PTPUDSMsg(@MessageArray),
MessageArraySize, PLongWord(@count), true, request, PTPUDSMsg(@requestConfirmation));
  if (count > 0) then
    MessageBox(0, 'Responses were received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForSingleMessage on page 117, WaitForMultipleMessage on page 121, ProcessResponse below.

Plain function Version: UDS_ WaitForServiceFunctional.

3.6.23 ProcessResponse

Processes a UDS response message to manage ISO-14229/15765 features, like session information.

Syntax

Pascal OO

```
class function ProcessResponse(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_ProcessResponse")]
public static extern TPUDSStatus ProcessResponse(
    [MarshalAs(UnmanagedType.U1)]
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_ProcessResponse")]
static TPUDSStatus ProcessResponse(
      [MarshalAs(UnmanagedType::U1)]
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_ProcessResponse")> _
Public Shared Function ProcessResponse( _
    <MarshalAs(UnmanagedType.U1)> _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A TPUDSMsg buffer to store the UDS response

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_WRONG_PARAM	MessageBuffer is not valid (data length is zero or the network result indicates an error)

Remarks: The purpose of this function is to update internal UDS settings of the API: currently only the responses to DiagnosticSessionControl requests require this processing as they contain information on the active session.

Example

The following example shows the use of the method ProcessResponse on the channel PUDS_USBBUS1. It writes a UDS physical request (service DiagnosticSessionControl) on the CAN Bus, waits for the confirmation of the transmission and then waits to receive a response. Once received the ProcessResponse function is called on the received message, updating the session information inside the API.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg confirmation = new TPUDSMsg();
```

```
TPUDSMsg response = new TPUDSMsg();

// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DiagnosticSessionControl Message
result = UDSApi.SvcDiagnosticSessionControl(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_DS);
if (result == TPUDSStatus.PUDS_ERROR_OK)
{
    // wait for the transmit confirmation
    result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, out confirmation, ref
request, true, 10, 100);
    if (result == TPUDSStatus.PUDS_ERROR_OK)
    {
        // wait for a response
        result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, out response, ref
confirmation, false, 10, 100);
        if (result == TPUDSStatus.PUDS_ERROR_OK)
        {
            result = UDSApi.ProcessResponse(UDSApi.PUDS_USBBUS1, ref response);
            MessageBox.Show(String.Format("Response was processed: {0}",
(int)result));
        }
        else
            // An error occurred
            MessageBox.Show(String.Format("Error occured while waiting for
response: {0}", (int)result));
    }
    else
        // An error occurred
        MessageBox.Show(String.Format("Error occured while waiting for transmit
confirmation: {0}", (int)result));
}
else
{
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
}
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ confirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();

// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DiagnosticSessionControl Message
result = UDSApi::SvcDiagnosticSessionControl(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamDSC::PUDS_SVC_PARAM_DSC_DS);
if (result == PUDS_ERROR_OK)
{
    // wait for the transmit confirmation
      result = UDSApi::WaitForSingleMessage(UDSApi::PUDS_USBBUS1, *confirmation,
*request, true, 10, 100);
    if (result == PUDS_ERROR_OK)
    {
        // wait for a response
            result = UDSApi::WaitForSingleMessage(UDSApi::PUDS_USBBUS1, *response,
*confirmation, false, 10, 100);
        if (result == PUDS_ERROR_OK)
        {
                    result = UDSApi::ProcessResponse(UDSApi::PUDS_USBBUS1,
*response);
            MessageBox::Show(String::Format("Response was processed: {0}",
(int)result));
        }
        else
            // An error occurred
            MessageBox::Show(String::Format("Error occured while waiting for
response: {0}", (int)result));
    }
    else
        // An error occurred
        MessageBox::Show(String::Format("Error occured while waiting for transmit
confirmation: {0}", (int)result));
}
else
{
    // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
}
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim confirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()

' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
```

```
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical DiagnosticSessionControl Message
result = UDSApi.SvcDiagnosticSessionControl(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_DS)
if (result = TPUDSStatus.PUDS_ERROR_OK) then
    ' wait for the transmit confirmation
    result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, confirmation,
request, true, 10, 100)
    if (result = TPUDSStatus.PUDS_ERROR_OK) then
        ' wait for a response
        result = UDSApi.WaitForSingleMessage(UDSApi.PUDS_USBBUS1, response,
confirmation, false, 10, 100)
        if (result = TPUDSStatus.PUDS_ERROR_OK) then
            result = UDSApi.ProcessResponse(UDSApi.PUDS_USBBUS1, response)
            MessageBox.Show(String.Format("Response was processed: {0}",
result.ToString()))
        else
            ' An error occurred
            MessageBox.Show(String.Format("Error occured while waiting for
response: {0}", result.ToString()))
        End If
    else
        ' An error occurred
        MessageBox.Show(String.Format("Error occured while waiting for transmit
confirmation: {0}", result.ToString()))
    End If
else
    ' An error occurred
    MessageBox.Show("Error occured: " + result.ToString())
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  confirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B  ;

  // Sends a Physical ECUReset Message
  result := TUdsApi.SvcDiagnosticSessionControl(TUdsApi.PUDS_USBBUS1, request,
PUDS_SVC_PARAM_DSC_DS);
  if (result = PUDS_ERROR_OK) then
  begin
    // wait for the transmit confirmation
    result := TUdsApi.WaitForSingleMessage(TUdsApi.PUDS_USBBUS1, confirmation, request, true, 10, 100);
    if (result = PUDS_ERROR_OK) and (confirmation.RESULT = PUDS_RESULT_N_OK) then
```

```
    begin
      // wait for the responses
      result := TUdsApi.WaitForSingleMessage(TUdsApi.PUDS_USBBUS1, response, confirmation, false, 10,
100);
      if (result = PUDS_ERROR_OK) then
      begin
        result := TUdsApi.ProcessResponse(TUdsApi.PUDS_USBBUS1, response);
        MessageBox(0, 'Received messages', 'Success', MB_OK);
      end
      else
        // An error occurred
        MessageBox(0, 'An error occured while waiting for response', 'Error', MB_OK);
    end
    else
      // An error occurred
      MessageBox(0, 'An error occured while waiting for transmit confirmation', 'Error', MB_OK);
  end
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForSingleMessage on page 117.

Plain function Version: UDS_ProcessResponse.

3.6.24 SvcDiagnosticSessionControl

Writes a UDS request according to the DiagnosticSessionControl service's specifications.

The DiagnosticSessionControl service is used to enable different diagnostic sessions in the server.

Syntax

Pascal OO

```
class function SvcDiagnosticSessionControl(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    SessionType: TPUDSSvcParamDSC): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcDiagnosticSessionControl")]
public static extern TPUDSStatus SvcDiagnosticSessionControl(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamDSC SessionType);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcDiagnosticSessionControl")]
static TPUDSStatus SvcDiagnosticSessionControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamDSC SessionType);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcDiagnosticSessionControl")> _
Public Shared Function SvcDiagnosticSessionControl( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal SessionType As TPUDSSvcParamDSC) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
SessionType	Subfunction parameter: type of the session (see TPUDSSvcParamDSC)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

If this service is called with the `NO_POSITIVE_RESPONSE_MSG` parameter of the MessageBuffer set to ignore responses (i.e. value `PUDS_SUPPR_POS_RSP_MSG_INDICATION_BIT`), the API will automatically change the current session to the new one.

If the `NO_POSITIVE_RESPONSE_MSG` parameter is set to keep responses (i.e. `PUDS_KEEP_POS_RSP_MSG_INDICATION_BIT`), the session information will be updated when the response is received (only if `WaitForService`, `WaitForMultipleMessage` or the `WaitForServiceFunctional` is used, otherwise `ProcessResponse` function must be called).

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
```

```
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DiagnosticSessionControl Request
result = UDSApi.SvcDiagnosticSessionControl(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_ECUPS);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DiagnosticSessionControl Request
result = UDSApi::SvcDiagnosticSessionControl(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamDSC::PUDS_SVC_PARAM_DSC_ECUPS);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical DiagnosticSessionControl Request
result = UDSApi.SvcDiagnosticSessionControl(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamDSC.PUDS_SVC_PARAM_DSC_ECUPS)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
```

```
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical DiagnosticSessionControl Request
  result := TUdsApi.SvcDiagnosticSessionControl(TUdsApi.PUDS_USBBUS1, request,
PUDS_SVC_PARAM_DSC_ECUPS);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcDiagnosticSessionControl.

3.6.25 SvcECUReset

Writes a UDS request according to the ECUReset service's specifications.

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

Syntax

Pascal OO

```
class function SvcECUReset(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    ResetType: TPUDSSvcParamER): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcECUReset")]
public static extern TPUDSStatus SvcECUReset(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamER ResetType);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcECUReset")]
static TPUDSStatus SvcECUReset(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamER ResetType);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcECUReset")> _
Public Shared Function SvcECUReset( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal ResetType As TPUDSSvcParamER) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
ResetType	Subfunction parameter: type of Reset (see TPUDSSvcParamER)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
```

```
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Request
result = UDSApi.SvcECUReset(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamER.PUDS_SVC_PARAM_ER_SR);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Request
result = UDSApi::SvcECUReset(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamER::PUDS_SVC_PARAM_ER_SR);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ECUReset Request
result = UDSApi.SvcECUReset(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamER.PUDS_SVC_PARAM_ER_SR)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ECUReset Request
  result := TUdsApi.SvcECUReset(TUdsApi.PUDS_USBBUS1, request, PUDS_SVC_PARAM_ER_SR);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcECUReset.

3.6.26 SvcSecurityAccess

Writes a UDS request according to the SecurityAccess service's specifications.

SecurityAccess service provides a mean to access data and/or diagnostic services which have restricted access for security, emissions or safety reasons.

Syntax

Pascal OO

```
class function SvcSecurityAccess(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    SecurityAccessType: Byte;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcSecurityAccess")]
public static extern TPUDSStatus SvcSecurityAccess(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte SecurityAccessType,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcSecurityAccess")]
static TPUDSStatus SvcSecurityAccess(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      Byte SecurityAccessType,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcSecurityAccess")> _
Public Shared Function SvcSecurityAccess( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal SecurityAccessType As Byte, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
SecurityAccessType	Subfunction parameter: type of SecurityAccess (see PUDS_SVC_PARAM_SA_xxx definitions)

Parameters	Description
Buffer	If Requesting Seed, buffer is the optional data to transmit to a server (like identification). If Sending Key, data holds the value generated by the security algorithm corresponding to a specific “seed” value
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service’s specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical SecurityAccess Request
byte[] buffer = { 0xF0, 0xA1, 0xB2, 0xC3};
result = UDSApi.SvcSecurityAccess(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.PUDS_SVC_PARAM_SA_RSD_1, buffer, (ushort) buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical SecurityAccess Request
array<Byte>^ buffer = { 0xF0, 0xA1, 0xB2, 0xC3};
result = UDSApi::SvcSecurityAccess(UDSApi::PUDS_USBBUS1, *request,
UDSApi::PUDS_SVC_PARAM_SA_RSD_1, buffer, (unsigned short) buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical SecurityAccess Request
Dim buffer as Byte() = { &HF0, &HA1, &HB2, &HC3}
result = UDSApi.SvcSecurityAccess(UDSApi.PUDS_USBBUS1, request,
UDSApi.PUDS_SVC_PARAM_SA_RSD_1, buffer, buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
```

```
  result: TPUDSStatus;
  buffer: array[0..3] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical SecurityAccess Request
  buffer[0] := $F0;
  buffer[1] := $A1;
  buffer[2] := $B2;
  buffer[3] := $C3;
  result := TUdsApi.SvcSecurityAccess(TUdsApi.PUDS_USBBUS1, request,
TUdsApi.PUDS_SVC_PARAM_SA_RSD_1, @buffer, Length(buffer));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127, PCAN-UDS Service Parameter Definitions on page 334: SecurityAccess.

Plain function Version: UDS_SvcSecurityAccess.

3.6.27 SvcCommunicationControl

Writes a UDS request according to the CommunicationControl service's specifications.

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

Syntax

Pascal OO

```
class function SvcCommunicationControl(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    ControlType: TPUDSSvcParamCC;
    CommunicationType: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcCommunicationControl")]
public static extern TPUDSStatus SvcCommunicationControl(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
```

```
    TPUDSSvcParamCC ControlType,
    byte CommunicationType);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcCommunicationControl")]
static TPUDSStatus SvcCommunicationControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamCC ControlType,
      Byte CommunicationType);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcCommunicationControl")> _
Public Shared Function SvcCommunicationControl( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal ControlType As TPUDSSvcParamCC, _
    ByVal CommunicationType As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
ControlType	Subfunction parameter: type of CommunicationControl (see TPUDSSvcParamCC)
CommunicationType	A bit-code value to reference the kind of communication to be controlled, see PUDS_SVC_PARAM_CC_FLAG_xxx flags and ISO_14229-2006 §B.1 for bit-encoding

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical CommunicationControl Request
result = UDSApi.SvcCommunicationControl(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamCC.PUDS_SVC_PARAM_CC_ERXTX,
    UDSApi.PUDS_SVC_PARAM_CC_FLAG_APPL | UDSApi.PUDS_SVC_PARAM_CC_FLAG_NWM |
UDSApi.PUDS_SVC_PARAM_CC_FLAG_DENWRIRO);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical CommunicationControl Request
result = UDSApi::SvcCommunicationControl(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamCC::PUDS_SVC_PARAM_CC_ERXTX,
    UDSApi::PUDS_SVC_PARAM_CC_FLAG_APPL | UDSApi::PUDS_SVC_PARAM_CC_FLAG_NWM |
UDSApi::PUDS_SVC_PARAM_CC_FLAG_DENWRIRO);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
```

```
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical CommunicationControl Request
result = UDSApi.SvcCommunicationControl(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamCC.PUDS_SVC_PARAM_CC_ERXTX, _
    (UDSApi.PUDS_SVC_PARAM_CC_FLAG_APPL Or UDSApi.PUDS_SVC_PARAM_CC_FLAG_NWM Or
UDSApi.PUDS_SVC_PARAM_CC_FLAG_DENWRIRO))
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical CommunicationControl Request
  result := TUdsApi.SvcCommunicationControl(TUdsApi.PUDS_USBBUS1, request,
PUDS_SVC_PARAM_CC_ERXTX,
    TUdsApi.PUDS_SVC_PARAM_CC_FLAG_APPL Or TUdsApi.PUDS_SVC_PARAM_CC_FLAG_NWM Or
TUdsApi.PUDS_SVC_PARAM_CC_FLAG_DENWRIRO);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127, PCAN-UDS Service Parameter Definitions on page 334: CommunicationControl.

Plain function Version: UDS_SvcCommunication Control.

3.6.28 SvcTesterPresent

Writes a UDS request according to the TesterPresent service's specifications.

TesterPresent service indicates to a server (or servers) that a client is still connected to the vehicle and that certain diagnostic services and/or communications that have been previously activated are to remain active.

Syntax

Pascal OO

```
class function SvcTesterPresent(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    TesterPresentType: TPUDSSvcParamTP = PUDS_SVC_PARAM_TP_ZSUBF): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcTesterPresent")]
public static extern TPUDSStatus SvcTesterPresent(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamTP TesterPresentType);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcTesterPresent")]
static TPUDSStatus SvcTesterPresent(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamTP TesterPresentType);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcTesterPresent")> _
Public Shared Function SvcTesterPresent( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal TesterPresentType As TPUDSSvcParamTP) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
TesterPresentType	No Subfunction parameter by default (PUDS_SVC_PARAM_TP_ZSUBF)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical Tester Present
result = UDSApi.SvcTesterPresent(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamTP.PUDS_SVC_PARAM_TP_ZSUBF);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical Tester Present
result = UDSApi::SvcTesterPresent(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamTP::PUDS_SVC_PARAM_TP_ZSUBF);
```

```
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical Tester Present
result = UDSApi.SvcTesterPresent(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamTP.PUDS_SVC_PARAM_TP_ZSUBF)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical Tester Present
  result := TUdsApi.SvcTesterPresent(TUdsApi.PUDS_USBBUS1, request, PUDS_SVC_PARAM_TP_ZSUBF);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
```

```
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcTesterPresent.

3.6.29 SvcSecuredDataTransmission

Writes a UDS request according to the SecuredDataTransmission service's specifications.

SecuredDataTransmission service's purpose is to transmit data that is protected against attacks from third parties, which could endanger data security.

Syntax

Pascal OO

```
class function SvcSecuredDataTransmission(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcSecuredDataTransmission")]
public static extern TPUDSStatus SvcSecuredDataTransmission(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcSecuredDataTransmission")]
static TPUDSStatus SvcSecuredDataTransmission(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcSecuredDataTransmission")> _
Public Shared Function SvcSecuredDataTransmission( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
Buffer	buffer containing the data as processed by the Security Sub-Layer (See ISO-15764)
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical SecuredDataTransmission Request
byte[] buffer = { 0xF0, 0xA1, 0xB2, 0xC3 };
result = UDSApi.SvcSecuredDataTransmission(UDSApi.PUDS_USBBUS1, ref request,
    buffer, (ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical SecuredDataTransmission Request
array<Byte>^ buffer = { 0xF0, 0xA1, 0xB2, 0xC3 };
result = UDSApi::SvcSecuredDataTransmission(UDSApi::PUDS_USBBUS1, *request,
      buffer, (unsigned short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical SecuredDataTransmission Request
Dim buffer As Byte() = { &HF0, &HA1, &HB2, &HC3 }
result = UDSApi.SvcSecuredDataTransmission(UDSApi.PUDS_USBBUS1, request, buffer,
buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
   request: TPUDSMsg;
   requestConfirmation: TPUDSMsg;
   response: TPUDSMsg;
   result: TPUDSStatus;
   buffer: array[0..3] of Byte;

begin
   // initialization
   request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
   request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
   request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
   request.NETADDRINFO.RA := $00;
   request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

   // Sends a Physical SecuredDataTransmission Request
   buffer[0] := $F0;
   buffer[1] := $A1;
   buffer[2] := $B2;
   buffer[3] := $C3;
   result := TUdsApi.SvcSecuredDataTransmission(TUdsApi.PUDS_USBBUS1, request,
      @buffer, Length(buffer));
   if (result = PUDS_ERROR_OK) then
      result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
   if (result = PUDS_ERROR_OK) then
      MessageBox(0, 'Response was received.', 'Success', MB_OK)
   else
      // An error occurred
      MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvSecuredDataTransmission.

3.6.30 SvcControlDTCSetting

Writes a UDS request according to the ControlDTCSetting service's specifications.

ControlDTCSetting service shall be used by a client to stop or resume the setting of diagnostic trouble codes (DTCs) in the server(s).

Syntax

Pascal OO

```
class function SvcControlDTCSetting(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DTCSettingType: TPUDSSvcParamCDTCS;
    Buffer: PByte;
```

```
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcControlDTCSetting")]
public static extern TPUDSStatus SvcControlDTCSetting(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamCDTCS DTCSettingType,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcControlDTCSetting")]
static TPUDSStatus SvcControlDTCSetting(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamCDTCS DTCSettingType,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcControlDTCSetting")> _
Public Shared Function SvcControlDTCSetting( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DTCSettingType As TPUDSSvcParamCDTCS, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DTCSettingType	Subfunction parameter (see TPUDSSvcParamCDTCS)
Buffer	This parameter record is user-optional and transmits data to a server when controlling the DTC setting. It can contain a list of DTCs to be turned on or off
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ControlDTCSetting Request
byte[] buffer = { 0xF0, 0xA1, 0xB2, 0xC3 };
result = UDSApi.SvcControlDTCSetting(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamCDTCS.PUDS_SVC_PARAM_CDTCS_OFF, buffer, (ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ControlDTCSetting Request
array<Byte>^ buffer = { 0xF0, 0xA1, 0xB2, 0xC3 };
result = UDSApi::SvcControlDTCSetting(UDSApi::PUDS_USBBUS1, *request,
      UDSApi::TPUDSSvcParamCDTCS::PUDS_SVC_PARAM_CDTCS_OFF, buffer, (unsigned
short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
```

```
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ControlDTCSetting Request
Dim buffer As Byte() = { &HF0, &HA1, &HB2, &HC3 }
result = UDSApi.SvcControlDTCSetting(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamCDTCS.PUDS_SVC_PARAM_CDTCS_OFF, buffer, buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  buffer: array[0..3] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ControlDTCSetting Request
  buffer[0] := $F0;
  buffer[1] := $A1;
  buffer[2] := $B2;
  buffer[3] := $C3;
  result := TUdsApi.SvcControlDTCSetting(TUdsApi.PUDS_USBBUS1, request,
    PUDS_SVC_PARAM_CDTCS_OFF, @buffer, Length(buffer));
```

```
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcControlDTCSetting.

3.6.31 SvcResponseOnEvent

Writes a UDS request according to the ResponseOnEvent service's specifications.

The ResponseOnEvent service requests a server to start or stop transmission of responses on a specified event.

Syntax

Pascal OO

```
class function SvcResponseOnEvent(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    EventType: TPUDSSvcParamROE;
    StoreEvent: Boolean;
    EventWindowTime: Byte;
    EventTypeRecord: PByte;
    EventTypeRecordLength: Word;
    ServiceToRespondToRecord: PByte;
    ServiceToRespondToRecordLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcResponseOnEvent")]
public static extern TPUDSStatus SvcResponseOnEvent(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamROE EventType,
    bool StoreEvent,
    byte EventWindowTime,
    byte[] EventTypeRecord,
    ushort EventTypeRecordLength,
    byte[] ServiceToRespondToRecord,
    ushort ServiceToRespondToRecordLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcResponseOnEvent")]
static TPUDSStatus SvcResponseOnEvent(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamROE EventType,
```

```
      bool StoreEvent,
      Byte EventWindowTime,
      array<Byte>^ EventTypeRecord,
      unsigned short EventTypeRecordLength,
      array<Byte>^ ServiceToRespondToRecord,
      unsigned short ServiceToRespondToRecordLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcResponseOnEvent")> _
Public Shared Function SvcResponseOnEvent( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal EventType As TPUDSSvcParamROE, _
    ByVal StoreEvent As Boolean, _
    ByVal EventWindowTime As Byte, _
    ByVal EventTypeRecord As Byte(), _
    ByVal EventTypeRecordLength As UShort, _
    ByVal ServiceToRespondToRecord As Byte(), _
    ByVal ServiceToRespondToRecordLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
EventType	Subfunction parameter: event type (see TPUDSSvcParamROE).
StoreEvent	Storage State (TRUE = Store Event, FALSE = Do Not Store Event)
EventWindowTime	Specify a window for the event logic to be active in the server (see also PUDS_SVC_PARAM_ROE_EWT_ITTR)
EventTypeRecord	Additional parameters for the specified eventType
EventTypeRecordLength	Size in bytes of the EventType Record (see PUDS_SVC_PARAM_ROE_xxx_LEN definitions)
ServiceToRespondToRecord	Service parameters, with first byte as service Id (see TPUDSSvcParamROERecommendedServiceID)
ServiceToRespondToRecordLength	Size in bytes of the ServiceToRespondTo Record

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ResponseOnEvent Request
byte[] evTypeBuffer = { 0x08 };
byte[] siResponseBuffer = { (byte)
UDSApi.TPUDSSvcParamROERecommendedServiceID.PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI,
                (byte) UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RNODTCBSM,
                0x01};
result = UDSApi.SvcResponseOnEvent(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamROE.PUDS_SVC_PARAM_ROE_ONDTCS,
    false, 0x1A, evTypeBuffer, (ushort)evTypeBuffer.Length, siResponseBuffer,
(ushort)siResponseBuffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ResponseOnEvent Request
array<Byte>^ evTypeBuffer = { 0x08 };
array<Byte>^ siResponseBuffer = { (Byte)
UDSApi::TPUDSSvcParamROERecommendedServiceID::PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI,
                            (Byte)
UDSApi::TPUDSSvcParamRDTCI::PUDS_SVC_PARAM_RDTCI_RNODTCBSM,
                            0x01};
```

```
result = UDSApi::SvcResponseOnEvent(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamROE::PUDS_SVC_PARAM_ROE_ONDTCS,
    false, 0x1A, evTypeBuffer, (unsigned short)evTypeBuffer->Length,
siResponseBuffer, (unsigned short) siResponseBuffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ResponseOnEvent Request
Dim evTypeBuffer As Byte() = { &H08 }
Dim siResponseBuffer As Byte()= {
UDSApi.TPUDSSvcParamROERecommendedServiceID.PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI, _

UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RNODTCBSM, _
                              &H01}
result = UDSApi.SvcResponseOnEvent(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamROE.PUDS_SVC_PARAM_ROE_ONDTCS, _
    false, &H1A, evTypeBuffer, evTypeBuffer.Length, siResponseBuffer,
siResponseBuffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  evTypeBuffer: array[0..0] of Byte;
  siResponseBuffer: array[0..2] of Byte;
```

```
begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ResponseOnEvent Request
  evTypeBuffer[0] := $08;
  siResponseBuffer[0] := Byte(PUDS_SVC_PARAM_ROE_STRT_SI_RDTCI);
  siResponseBuffer[1] := Byte(PUDS_SVC_PARAM_RDTCI_RNODTCBSM);
  siResponseBuffer[2] := 01;
  result := TUdsApi.SvcResponseOnEvent(TUdsApi.PUDS_USBBUS1, request,
PUDS_SVC_PARAM_ROE_ONDTCS,
    false, $1A, @evTypeBuffer, Length(evTypeBuffer), @siResponseBuffer, Length(siResponseBuffer));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127, PCAN-UDS Service Parameter Definitions on page 334: ResponseOnEvent on page 334.

Plain function Version: UDS_SvcResponseOnEvent.

3.6.32 SvcLinkControl

Writes a UDS request according to the LinkControl service's specifications.

The LinkControl service is used to control the communication link baud rate between the client and the server(s) for the exchange of diagnostic data.

Syntax

Pascal OO

```
class function SvcLinkControl(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    LinkControlType: TPUDSSvcParamLC;
    BaudrateIdentifier: Byte;
    LinkBaudrate: LongWord): TPUDSStatus; overload;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcLinkControl")]
public static extern TPUDSStatus SvcLinkControl(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamLC LinkControlType,
    byte BaudrateIdentifier,
    UInt32 LinkBaudrate);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcLinkControl")]
static TPUDSStatus SvcLinkControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamLC LinkControlType,
      Byte BaudrateIdentifier,
      UInt32 LinkBaudrate);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcLinkControl")> _
Public Shared Function SvcLinkControl( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal LinkControlType As TPUDSSvcParamLC, _
    ByVal BaudrateIdentifier As Byte, _
    ByVal LinkBaudrate As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
LinkControlType	Subfunction parameter: Link Control Type (see TPUDSSvcParamLC).
BaudrateIdentifier	Defined baud rate identifier (see TPUDSSvcParamLCBaudrateIdentifier)
LinkBaudrate	Used only with PUDS_SVC_PARAM_LC_VBTWSBR parameter: a three-byte value baud rate (baudrate High, Middle and Low Bytes)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical LinkControl Message (Verify Specific Baudrate)
result = UDSApi.SvcLinkControl(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamLC.PUDS_SVC_PARAM_LC_VBTWSBR, 0, 500000);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical LinkControl Message (Verify Specific Baudrate)
result = UDSApi::SvcLinkControl(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamLC::PUDS_SVC_PARAM_LC_VBTWSBR, 0, 500000);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical LinkControl Message (Verify Specific Baudrate)
result = UDSApi.SvcLinkControl(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamLC.PUDS_SVC_PARAM_LC_VBTWSBR, 0, 500000)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical LinkControl Message (Verify Specific Baudrate)
  result := TUdsApi.SvcLinkControl(TUdsApi.PUDS_USBBUS1, request, PUDS_SVC_PARAM_LC_VBTWSBR, 0,
500000);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcLinkControl.

3.6.33 SvcReadDataByIdentifier

Writes a UDS request according to the ReadDataByIdentifier service's specifications.

The ReadDataByIdentifier service allows the client to request data record values from the server identified by one or more dataIdentifiers.

Syntax

Pascal OO

```
class function SvcReadDataByIdentifier(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    Buffer: PWord;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDataByIdentifier")]
public static extern TPUDSStatus SvcReadDataByIdentifier(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDataByIdentifier")]
static TPUDSStatus SvcReadDataByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      array<unsigned short>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDataByIdentifier")> _
Public Shared Function SvcReadDataByIdentifier( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal Buffer As UShort(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
Buffer	Buffer containing a list of two-byte Data Identifiers (see TPUDSSvcParamDI).
BufferLength	Number of elements in the buffer (size in WORD of the buffer)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDataByIdentifier Request
ushort[] buffer = { (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ADSDID,
        (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ECUMDDID };
result = UDSApi.SvcReadDataByIdentifier(UDSApi.PUDS_USBBUS1, ref request, buffer,
(ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDataByIdentifier Request
array<unsigned short>^ buffer = { (unsigned
short)UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_ADSDID,
                    (unsigned
short)UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_ECUMDDID };
result = UDSApi::SvcReadDataByIdentifier(UDSApi::PUDS_USBBUS1, *request, buffer,
(unsigned short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDataByIdentifier Request
Dim buffer As UShort() = { UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ADSDID, _
                    UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ECUMDDID }
result = UDSApi.SvcReadDataByIdentifier(UDSApi.PUDS_USBBUS1, request, buffer,
buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
   request: TPUDSMsg;
   requestConfirmation: TPUDSMsg;
   response: TPUDSMsg;
   result: TPUDSStatus;
   buffer: array[0..1] of Word;

begin
   // initialization
   request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
   request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
   request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
   request.NETADDRINFO.RA := $00;
   request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

   // Sends a Physical ReadDataByIdentifier Request
   buffer[0] := Word(PUDS_SVC_PARAM_DI_ADSDID);
   buffer[1] := Word(PUDS_SVC_PARAM_DI_ECUMDDID);
   result := TUdsApi.SvcReadDataByIdentifier(TUdsApi.PUDS_USBBUS1, request, @buffer, Length(buffer));
   if (result = PUDS_ERROR_OK) then
      result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
   if (result = PUDS_ERROR_OK) then
      MessageBox(0, 'Response was received.', 'Success', MB_OK)
   else
      // An error occurred
      MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDataByIdentifier.

3.6.34 SvcReadMemoryByAddress

Writes a UDS request according to the ReadMemoryByAddress service's specifications.

The ReadMemoryByAddress service allows the client to request memory data from the server via a provided starting address and to specify the size of memory to be read.

Syntax

Pascal OO

```
class function SvcReadMemoryByAddress(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    MemoryAddress: PByte;
    MemoryAddressLength: Byte;
    MemorySize: PByte;
    MemorySizeLength: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadMemoryByAddress")]
public static extern TPUDSStatus SvcReadMemoryByAddress(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte[] MemoryAddress,
    byte MemoryAddressLength,
    byte[] MemorySize,
    byte MemorySizeLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadMemoryByAddress")]
static TPUDSStatus SvcReadMemoryByAddress(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      array<Byte>^ MemoryAddress,
      Byte MemoryAddressLength,
      array<Byte>^ MemorySize,
      Byte MemorySizeLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadMemoryByAddress")> _
Public Shared Function SvcReadMemoryByAddress( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal MemoryAddress As Byte(), _
    ByVal MemoryAddressLength As Byte, _
    ByVal MemorySize As Byte(), _
    ByVal MemorySizeLength As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
MemoryAddress	Starting address of server memory from which data is to be retrieved
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Number of bytes to be read starting at the address specified by memoryAddress
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadMemoryByAddress Request
byte[] lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
byte[] lBufferSize = { 0x01, 0x11 };
result = UDSApi.SvcReadMemoryByAddress(UDSApi.PUDS_USBBUS1, ref request,
    lBufferAddr, (byte)lBufferAddr.Length, lBufferSize, (byte)lBufferSize.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadMemoryByAddress Request
array<Byte>^ lBufferAddr = {0xF0, 0xA1, 0x00, 0x13};
array<Byte>^ lBufferSize = {0x01, 0x11};
result = UDSApi::SvcReadMemoryByAddress(UDSApi::PUDS_USBBUS1, *request,
    lBufferAddr, (Byte)lBufferAddr->Length, lBufferSize, (Byte)lBufferSize-
>Length);
if (result == PUDS_ERROR_OK)
```

```
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadMemoryByAddress Request
Dim lBufferAddr As Byte() = {&HF0, &HA1, &H00, &H13}
Dim lBufferSize As Byte() = {&H01, &H11}
result = UDSApi.SvcReadMemoryByAddress(UDSApi.PUDS_USBBUS1, request, _
    lBufferAddr, lBufferAddr.Length, lBufferSize, lBufferSize.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  lBufferAddr: array[0..3] of Byte;
  lBufferSize: array[0..1] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadMemoryByAddress Request
  lBufferAddr[0] := $F0;
```

```
  lBufferAddr[1] := $A1;
  lBufferAddr[2] := $00;
  lBufferAddr[3] := $13;
  lBufferSize[0] := $01;
  lBufferSize[1] := $11;
  result := TUdsApi.SvcReadMemoryByAddress(TUdsApi.PUDS_USBBUS1, request,
    @lBufferAddr, Length(lBufferAddr), @lBufferSize, Length(lBufferSize));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadMemoryByAddress.

3.6.35 SvcReadScalingDataByIdentifier

Writes a UDS request according to the ReadScalingDataByIdentifier service's specifications.

The ReadScalingDataByIdentifier service allows the client to request scaling data record information from the server identified by a dataIdentifier.

Syntax

Pascal OO

```
class function SvcReadScalingDataByIdentifier(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DataIdentifier: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadScalingDataByIdentifier")]
public static extern TPUDSStatus SvcReadScalingDataByIdentifier(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DataIdentifier);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadScalingDataByIdentifier")]
static TPUDSStatus SvcReadScalingDataByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DataIdentifier);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadScalingDataByIdentifier")> _
Public Shared Function SvcReadScalingDataByIdentifier( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DataIdentifier As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadScalingDataByIdentifier Request
result = UDSApi.SvcReadScalingDataByIdentifier(UDSApi.PUDS_USBBUS1, ref request,
(ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_BSFPDID);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
```

```
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadScalingDataByIdentifier Request
result = UDSApi::SvcReadScalingDataByIdentifier(UDSApi::PUDS_USBBUS1, *request,
(unsigned short)UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_BSFPDID);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadScalingDataByIdentifier Request
result = UDSApi.SvcReadScalingDataByIdentifier(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_BSFPDID)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
   request: TPUDSMsg;
   requestConfirmation: TPUDSMsg;
   response: TPUDSMsg;
   result: TPUDSStatus;

begin
   // initialization
   request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
   request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
   request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
   request.NETADDRINFO.RA := $00;
   request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

   // Sends a Physical ReadScalingDataByIdentifier Request
   result := TUdsApi.SvcReadScalingDataByIdentifier(TUdsApi.PUDS_USBBUS1, request,
Word(PUDS_SVC_PARAM_DI_BSFPDID));
   if (result = PUDS_ERROR_OK) then
      result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
   if (result = PUDS_ERROR_OK) then
      MessageBox(0, 'Response was received.', 'Success', MB_OK)
   else
      // An error occurred
      MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadScalingDataByIdentifier.

3.6.36 SvcReadDataByPeriodicIdentifier

Writes a UDS request according to the ReadDataByPeriodicIdentifier service's specifications.

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

Syntax

Pascal OO

```
class function SvcReadDataByPeriodicIdentifier(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    TransmissionMode: TPUDSSvcParamRDBPI;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDataByPeriodicIdentifier")]
public static extern TPUDSStatus SvcReadDataByPeriodicIdentifier(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamRDBPI TransmissionMode,
    byte[] Buffer,
    ushort BufferLength);
#endregion
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDataByPeriodicIdentifier")]
static TPUDSStatus SvcReadDataByPeriodicIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamRDBPI TransmissionMode,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDataByPeriodicIdentifier")> _
Public Shared Function SvcReadDataByPeriodicIdentifier( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal TransmissionMode As TPUDSSvcParamRDBPI, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
TransmissionMode	Transmission rate code (see TPUDSSvcParamRDBPI).
Buffer	Buffer containing a list of Periodic Data Identifiers.
BufferLength	Number of elements in the buffer (size in WORD of the buffer)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDataByPeriodicIdentifier Request
byte[] buffer = { 0xE3 };
result = UDSApi.SvcReadDataByPeriodicIdentifier(UDSApi.PUDS_USBBUS1, ref request,
    UDSApi.TPUDSSvcParamRDBPI.PUDS_SVC_PARAM_RDBPI_SAMR, buffer,
(ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDataByPeriodicIdentifier Request
array<Byte>^ buffer = { 0xE3 };
result = UDSApi::SvcReadDataByPeriodicIdentifier(UDSApi::PUDS_USBBUS1, *request,
    UDSApi::TPUDSSvcParamRDBPI::PUDS_SVC_PARAM_RDBPI_SAMR,  buffer, (unsigned
short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
```

```
        MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDataByPeriodicIdentifier Request
Dim buffer As Byte() = { &HE3 }
result = UDSApi.SvcReadDataByPeriodicIdentifier(UDSApi.PUDS_USBBUS1, request, _
    UDSApi.TPUDSSvcParamRDBPI.PUDS_SVC_PARAM_RDBPI_SAMR,  buffer, buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  buffer: array[0..0] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical DiagnosticSessionControl Request
  buffer[0] := $E3;
  result := TUdsApi.SvcReadDataByPeriodicIdentifier(TUdsApi.PUDS_USBBUS1, request,
    PUDS_SVC_PARAM_RDBPI_SAMR, @buffer, Length(buffer));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
```

```
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDataByPeriodicIdentifier.

3.6.37 SvcDynamicallyDefineDataIdentifierDBID

Writes a UDS request according to the DynamicallyDefineDataIdentifier service's specifications.

The DynamicallyDefineDataIdentifier service allows the client to dynamically define in a server a data identifier that can be read via the ReadDataByIdentifier service at a later time. The Define By Identifier subfunction specifies that definition of the dynamic data identifier shall occur via a data identifier reference.

Syntax

Pascal OO

```
class function SvcDynamicallyDefineDataIdentifierDBID(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DynamicallyDefinedDataIdentifier: Word;
    SourceDataIdentifier: PWord;
    MemorySize: PByte;
    PositionInSourceDataRecord: PByte;
    BuffersLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint =
"UDS_SvcDynamicallyDefineDataIdentifierDBID")]
public static extern TPUDSStatus SvcDynamicallyDefineDataIdentifierDBID(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DynamicallyDefinedDataIdentifier,
    byte[] SourceDataIdentifier,
    byte[] MemorySize,
    byte[] PositionInSourceDataRecord,
    ushort BuffersLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint =
"UDS_SvcDynamicallyDefineDataIdentifierDBID")]
static TPUDSStatus SvcDynamicallyDefineDataIdentifierDBID(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DynamicallyDefinedDataIdentifier,
      array<Byte>^ SourceDataIdentifier,
      array<Byte>^ MemorySize,
      array<Byte>^ PositionInSourceDataRecord,
      unsigned short BuffersLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll",
EntryPoint:="UDS_SvcDynamicallyDefineDataIdentifierDBID")> _
Public Shared Function SvcDynamicallyDefineDataIdentifierDBID( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DynamicallyDefinedDataIdentifier As UShort, _
    ByVal SourceDataIdentifier As Byte(), _
    ByVal MemorySize As Byte(), _
    ByVal PositionInSourceDataRecord As Byte(), _
    ByVal BuffersLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DynamicallyDefinedDataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
SourceDataIdentifier	buffer containing the sources of information to be included into the dynamic data record
MemorySize	buffer containing the total numbers of bytes from the source data record address
PositionInSourceDataRecord	buffer containing the starting byte positions of the excerpt of the source data record
BuffersLength	Number of elements in the buffers (SourceDataIdentifier, MemoryAddress and PositionInSourceDataRecord)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
```

```
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifier 'Define By Identifier' Request
ushort bufferSize = 5;
ushort[] lBufferSourceDI = new ushort[bufferSize];
byte[] lBufferMemSize = new byte[bufferSize];
byte[] lBufferPosInSrc = new byte[bufferSize];
// Fill data [...]
result = UDSApi.SvcDynamicallyDefineDataIdentifierDBID(UDSApi.PUDS_USBBUS1, ref
request,
    (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_CDDID,
    lBufferSourceDI, lBufferMemSize, lBufferPosInSrc, bufferSize);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifier 'Define By Identifier' Request
unsigned short bufferSize = 5;
array<unsigned short>^ lBufferSourceDI = gcnew array<unsigned short>(bufferSize);
array<Byte>^ lBufferMemSize = gcnew array<Byte>(bufferSize);
array<Byte>^ lBufferPosInSrc = gcnew array<Byte>(bufferSize);
// Fill data [...]
result = UDSApi::SvcDynamicallyDefineDataIdentifierDBID(UDSApi::PUDS_USBBUS1,
*request,
    (unsigned short) UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_CDDID,
    lBufferSourceDI, lBufferMemSize, lBufferPosInSrc, bufferSize);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical DynamicallyDefineDataIdentifier 'Define By Identifier' Request
Dim bufferSize As ushort = 5
Dim lBufferSourceDI(bufferSize) As UShort
Dim lBufferMemSize(bufferSize) As Byte
Dim lBufferPosInSrc(bufferSize) As Byte
' Fill data [...]
result = UDSApi.SvcDynamicallyDefineDataIdentifierDBID(UDSApi.PUDS_USBBUS1,
request, _
     UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_CDDID, _
    lBufferSourceDI, lBufferMemSize, lBufferPosInSrc, bufferSize)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  bufferSize: Word;
  lBufferSourceDI: array[0..4] of Word;
  lBufferMemSize: array[0..4] of Byte;
  lBufferPosInSrc: array[0..4] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical DynamicallyDefineDataIdentifier 'Define By Identifier' Request
  bufferSize := 5;
  // Fill data [...]
```

```
  result := TUdsApi.SvcDynamicallyDefineDataIdentifierDBID(TUdsApi.PUDS_USBBUS1, request,
    Word(PUDS_SVC_PARAM_DI_CDDID), @lBufferSourceDI, @lBufferMemSize, @lBufferPosInSrc,
bufferSize);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_DynamicallyDefineDataIdentifier.

3.6.38 SvcDynamicallyDefineDataIdentifierDBMA

Writes a UDS request according to the DynamicallyDefineDataIdentifier service's specifications.

The DynamicallyDefineDataIdentifier service allows the client to dynamically define in a server a data identifier that can be read via the ReadDataByIdentifier service at a later time. The Define By Memory Address subfunction specifies that definition of the dynamic data identifier shall occur via an address reference.

Syntax

Pascal OO

```
class function SvcDynamicallyDefineDataIdentifierDBMA(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DynamicallyDefinedDataIdentifier: Word;
    MemoryAddressLength: Byte;
    MemorySizeLength: Byte;
    MemoryAddressBuffer: PByte;
    MemorySizeBuffer: PByte;
    BuffersLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint =
"UDS_SvcDynamicallyDefineDataIdentifierDBMA")]
public static extern TPUDSStatus SvcDynamicallyDefineDataIdentifierDBMA(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DynamicallyDefinedDataIdentifier,
    byte MemoryAddressLength,
    byte MemorySizeLength,
    byte[] MemoryAddressBuffer,
    byte[] MemorySizeBuffer,
    ushort BuffersLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint =
"UDS_SvcDynamicallyDefineDataIdentifierDBMA")]
static TPUDSStatus SvcDynamicallyDefineDataIdentifierDBMA(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DynamicallyDefinedDataIdentifier,
      Byte MemoryAddressLength,
      Byte MemorySizeLength,
      array<Byte>^ MemoryAddressBuffer,
      array<Byte>^ MemorySizeBuffer,
      unsigned short BuffersLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll",
EntryPoint:="UDS_SvcDynamicallyDefineDataIdentifierDBMA")> _
Public Shared Function SvcDynamicallyDefineDataIdentifierDBMA( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DynamicallyDefinedDataIdentifier As UShort, _
    ByVal MemoryAddressLength As Byte, _
    ByVal MemorySizeLength As Byte, _
    ByVal MemoryAddressBuffer As Byte(), _
    ByVal MemorySizeBuffer As Byte(), _
    ByVal BuffersLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DynamicallyDefinedDataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
MemoryAddressLength	Size in bytes of the MemoryAddress items in the MemoryAddressBuffer buffer (max.: 0xF)
MemorySizeLength	Size in bytes of the MemorySize items in the MemorySizeBuffer buffer (max.: 0xF)
MemoryAddressBuffer	Buffer containing the MemoryAddress buffers, must be an array of 'BuffersLength' entries which contains 'MemoryAddressLength' bytes (size is 'BuffersLength * MemoryAddressLength' bytes)
MemorySizeBuffer	Buffer containing the MemorySize buffers, must be an array of 'BuffersLength' entries which contains 'MemorySizeLength' bytes (size is 'BuffersLength * MemorySizeLength' bytes)
BuffersLength	Size in bytes of the MemoryAddressBuffer and MemorySizeBuffer buffers

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifier 'Define By Memory Address'
Request
byte buffLen = 3;
byte buffAddrLen = 5;
byte buffSizeLen = 3;
byte[] lBuffsAddr = new byte[buffLen * buffAddrLen];
byte[] lBuffsSize = new byte[buffLen * buffSizeLen];
for (int j = 0; j < buffLen; j++)
{
    for (int i = 0; i < buffAddrLen; i++)
    {
        lBuffsAddr[buffAddrLen * j + i] = (byte)((10 * j) + i + 1);
    }
    for (int i = 0; i < buffSizeLen; i++)
    {
        lBuffsSize[buffSizeLen * j + i] = (byte)(100 + (10 * j) + i + 1);
    }
}
result = UDSApi.SvcDynamicallyDefineDataIdentifierDBMA(UDSApi.PUDS_USBBUS1, ref
request,
    (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_CESWNDID,
    buffAddrLen, buffSizeLen, lBuffsAddr, lBuffsSize, buffLen);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
```

```
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifier 'Define By Memory Address'
Request
Byte buffLen = 3;
Byte buffAddrLen = 5;
Byte buffSizeLen = 3;
array<Byte>^ lBuffsAddr = gcnew array<Byte>(buffLen * buffAddrLen);
array<Byte>^ lBuffsSize = gcnew array<Byte>(buffLen * buffSizeLen);
for (int j = 0 ; j < buffLen ; j++)
{
    for (int i = 0 ; i < buffAddrLen ; i++) {
        lBuffsAddr[buffAddrLen*j+i] = (Byte)((10 * j) + i + 1);
    }
    for (int i = 0 ; i < buffSizeLen ; i++) {
        lBuffsSize[buffSizeLen*j+i] = (Byte)(100 + (10 * j) + i + 1);
    }
}
result = UDSApi::SvcDynamicallyDefineDataIdentifierDBMA(UDSApi::PUDS_USBBUS1,
*request,
    (unsigned short) UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_CESWNDID,
    buffAddrLen, buffSizeLen, lBuffsAddr, lBuffsSize, buffLen);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical DynamicallyDefineDataIdentifier 'Define By Memory Address'
Request
Dim buffLen As Byte = 3
Dim buffAddrLen As Byte = 5
Dim buffSizeLen As Byte = 3
Dim lBuffsAddr(buffLen * buffAddrLen) As Byte
Dim lBuffsSize(buffLen * buffSizeLen) As Byte
for j as Integer = 0 To buffLen - 1
    for i as Integer = 0 To buffAddrLen - 1
        lBuffsAddr(buffAddrLen*j+i) = ((10 * j) + i + 1)
    Next
    for i as Integer  = 0 To buffSizeLen - 1
        lBuffsSize(buffSizeLen*j+i) = (100 + (10 * j) + i + 1)
    Next
Next
result = UDSApi.SvcDynamicallyDefineDataIdentifierDBMA(UDSApi.PUDS_USBBUS1,
request, _
     UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_CESWNDID, _
    buffAddrLen, buffSizeLen, lBuffsAddr, lBuffsSize, buffLen)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  buffLen: Byte;
  buffAddrLen: Byte;
  buffSizeLen: Byte;
  lBuffsAddr: array[0..14] of Byte;  //  buffLen * buffAddrLen
  lBuffsSize: array[0..8] of Byte;   // buffLen * buffSizeLen
  i, j: Word;

begin
```

```
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical DynamicallyDefineDataIdentifier 'Define By Memory Address' Request
  buffLen := 3;
  buffAddrLen := 5;
  buffSizeLen := 3;
  for j := 0 to buffLen - 1 do
  begin
    for i := 0 to buffAddrLen - 1 do
      lBuffsAddr[buffAddrLen*j+i] := Byte((10 * j) + i + 1);
    for i := 0 to buffSizeLen - 1 do
      lBuffsSize[buffSizeLen*j+i] := Byte(100 + (10 * j) + i + 1);
  end;
  result := TUdsApi.SvcDynamicallyDefineDataIdentifierDBMA(TUdsApi.PUDS_USBBUS1, request,
    Word(PUDS_SVC_PARAM_DI_CESWNDID), buffAddrLen, buffSizeLen, @lBuffsAddr, @lBuffsSize, buffLen);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcDynamicallyDefineDataIdentifierDBMA.

3.6.39 SvcDynamicallyDefineDataIdentifierCDDDI

Writes a UDS request according to the Clear Dynamically Defined Data Identifier service's specifications.

The Clear Dynamically Defined Data Identifier subfunction shall be used to clear the specified dynamic data identifier.

Syntax

Pascal OO

```
class function SvcDynamicallyDefineDataIdentifierCDDDI(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DynamicallyDefinedDataIdentifier: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint =
"UDS_SvcDynamicallyDefineDataIdentifierCDDDI")]
public static extern TPUDSStatus SvcDynamicallyDefineDataIdentifierCDDDI(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DynamicallyDefinedDataIdentifier);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint =
"UDS_SvcDynamicallyDefineDataIdentifierCDDDI")]
static TPUDSStatus SvcDynamicallyDefineDataIdentifierCDDDI(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DynamicallyDefinedDataIdentifier);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll",
EntryPoint:="UDS_SvcDynamicallyDefineDataIdentifierCDDDI")> _
Public Shared Function SvcDynamicallyDefineDataIdentifierCDDDI( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DynamicallyDefinedDataIdentifier As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DynamicallyDefinedDataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifier 'Clear Dynamically Defined
Identifier' Request
result = UDSApi.SvcDynamicallyDefineDataIdentifierCDDDI(UDSApi.PUDS_USBBUS1, ref
request, (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_CESWNDID);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifier 'Clear Dynamically Defined
Identifier' Request
result = UDSApi::SvcDynamicallyDefineDataIdentifierCDDDI(UDSApi::PUDS_USBBUS1,
*request, (unsigned short)UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_CESWNDID);
if (result == PUDS_ERROR_OK)
    result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
    MessageBox::Show(String::Format("Response was received."));
else
    // An error occurred
    MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
```

```
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical DynamicallyDefineDataIdentifier 'Clear Dynamically Defined
Identifier' Request
result = UDSApi.SvcDynamicallyDefineDataIdentifierCDDDI(UDSApi.PUDS_USBBUS1,
request, UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_CESWNDID)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical DynamicallyDefineDataIdentifier 'Clear Dynamically Defined Identifier' Request
  result := TUdsApi.SvcDynamicallyDefineDataIdentifierCDDDI(TUdsApi.PUDS_USBBUS1, request,
Word(PUDS_SVC_PARAM_DI_CESWNDID));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcDynamicallyDefineDataIdentifierCDDDI.

3.6.40 SvcWriteDataByIdentifier

Writes a UDS request according to the WriteDataByIdentifier service's specifications.

The WriteDataByIdentifier service allows the client to write information into the server at an internal location specified by the provided data identifier.

Syntax

Pascal OO

```
class function SvcWriteDataByIdentifier(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DataIdentifier: Word;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcWriteDataByIdentifier")]
public static extern TPUDSStatus SvcWriteDataByIdentifier(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DataIdentifier,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcWriteDataByIdentifier")]
static TPUDSStatus SvcWriteDataByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DataIdentifier,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcWriteDataByIdentifier")> _
Public Shared Function SvcWriteDataByIdentifier( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DataIdentifier As UShort, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
Buffer	Buffer containing the data to write.
BufferLength	Size in bytes of the buffer.

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical WriteDataByIdentifier Request
byte[] buffer = { 0xE3, 0xA1, 0xB2 };
result = UDSApi.SvcWriteDataByIdentifier(UDSApi.PUDS_USBBUS1, ref request,
    (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ASFPDID, buffer,
(ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
```

```
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical WriteDataByIdentifier Request
array<Byte>^ buffer = { 0xE3, 0xA1, 0xB2 };
result = UDSApi::SvcWriteDataByIdentifier(UDSApi::PUDS_USBBUS1, *request,
    (unsigned short) UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_ASFPDID, buffer,
(unsigned short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical WriteDataByIdentifier Request
Dim buffer As Byte() = { &HE3, &HA1, &HB2 }
result = UDSApi.SvcWriteDataByIdentifier(UDSApi.PUDS_USBBUS1, request, _
     UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ASFPDID, buffer, buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
   request: TPUDSMsg;
   requestConfirmation: TPUDSMsg;
   response: TPUDSMsg;
   result: TPUDSStatus;
   buffer: array[0..2] of Byte;

begin
   // initialization
   request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
   request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
   request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
   request.NETADDRINFO.RA := $00;
   request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

   // Sends a Physical WriteDataByIdentifier Request
   buffer[0] := $E3;
   buffer[1] := $A1;
   buffer[2] := $B2;
   result := TUdsApi.SvcWriteDataByIdentifier(TUdsApi.PUDS_USBBUS1, request,
      Word(PUDS_SVC_PARAM_DI_ASFPDID), @buffer, Length(buffer));
   if (result = PUDS_ERROR_OK) then
      result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
   if (result = PUDS_ERROR_OK) then
      MessageBox(0, 'Response was received.', 'Success', MB_OK)
   else
      // An error occurred
      MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcWriteDataByIdentifier.

3.6.41 SvcWriteMemoryByAddress

Writes a UDS request according to the WriteMemoryByAddress service's specifications.

The WriteMemoryByAddress service allows the client to write information into the server at one or more contiguous memory locations.

Syntax

Pascal OO

```
class function SvcWriteMemoryByAddress(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DataIdentifier: Word;
    MemoryAddress: PByte;
```

```
    MemoryAddressLength: Byte;
    MemorySize: PByte;
    MemorySizeLength: Byte;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcWriteMemoryByAddress")]
public static extern TPUDSStatus SvcWriteMemoryByAddress(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DataIdentifier,
    byte[] MemoryAddress,
    byte MemoryAddressLength,
    byte[] MemorySize,
    byte MemorySizeLength,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcWriteMemoryByAddress")]
static TPUDSStatus SvcWriteMemoryByAddress(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DataIdentifier,
      array<Byte>^ MemoryAddress,
      Byte MemoryAddressLength,
      array<Byte>^ MemorySize,
      Byte MemorySizeLength,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcWriteMemoryByAddress")> _
Public Shared Function SvcWriteMemoryByAddress( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DataIdentifier As UShort, _
    ByVal MemoryAddress As Byte(), _
    ByVal MemoryAddressLength As Byte, _
    ByVal MemorySize As Byte(), _
    ByVal MemorySizeLength As Byte, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
MemoryAddress	Starting address of server memory to which data is to be written
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Number of bytes to be written starting at the address specified by memoryAddress

Parameters	Description
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)
Buffer	Buffer containing the data to write
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical WriteMemoryByAddress Request
byte[] lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
byte[] lBufferSize = { 0x01, 0x05 };
byte[] lBuffer = new byte[0x105];
// Fill lBuffer [...]
result = UDSApi.SvcWriteMemoryByAddress(UDSApi.PUDS_USBBUS1, ref request,
    (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ASFPDID,
    lBufferAddr, (byte)lBufferAddr.Length, lBufferSize, (byte)lBufferSize.Length,
    lBuffer, (ushort)lBuffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
```

```
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical WriteMemoryByAddress Request
array<Byte>^ lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
array<Byte>^ lBufferSize = { 0x01, 0x05 };
array<Byte>^ lBuffer = gcnew array<Byte>(0x105);
// Fill lBuffer [...]
result = UDSApi::SvcWriteMemoryByAddress(UDSApi::PUDS_USBBUS1, *request,
    (unsigned short)UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_ASFPDID,
    lBufferAddr, (Byte)lBufferAddr->Length, lBufferSize, (Byte)lBufferSize->Length,
    lBuffer, (unsigned short)lBuffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical WriteMemoryByAddress Request
Dim lBufferAddr As Byte() = { &HF0, &HA1, &H00, &H13 }
Dim lBufferSize As Byte() = { &H01, &H05 }
Dim lBuffer(&H105) As Byte
' Fill lBuffer [...]
result = UDSApi.SvcWriteMemoryByAddress(UDSApi.PUDS_USBBUS1, request, _
    UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_ASFPDID, _
    lBufferAddr, lBufferAddr.Length, lBufferSize, lBufferSize.Length, _
    lBuffer, lBuffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
```

```
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  lBufferAddr: array[0..3] of Byte;
  lBufferSize: array[0..1] of Byte;
  lBuffer: array[0..$105] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical WriteMemoryByAddress Request
  lBufferAddr[0] := $F0;
  lBufferAddr[1] := $A1;
  lBufferAddr[2] := $00;
  lBufferAddr[3] := $13;
  lBufferSize[0] := $01;
  lBufferSize[1] := $05;
  // Fill lBuffer [...]
  result := TUdsApi.SvcWriteMemoryByAddress(TUdsApi.PUDS_USBBUS1, request,
    Word(PUDS_SVC_PARAM_DI_ASFPDID),
    @lBufferAddr, Length(lBufferAddr), @lBufferSize, Length(lBufferSize),
    @lBuffer, Length(lBuffer));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcWriteMemoryByAddress.

3.6.42 SvcClearDiagnosticInformation

Writes a UDS request according to the ClearDiagnosticInformation service's specifications.

The ClearDiagnosticInformation service is used by the client to clear diagnostic information in one server's or multiple servers' memory.

Syntax

Pascal OO

```
class function SvcClearDiagnosticInformation(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    groupOfDTC: LongWord): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcClearDiagnosticInformation")]
public static extern TPUDSStatus SvcClearDiagnosticInformation(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    UInt32 groupOfDTC);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcClearDiagnosticInformation")]
static TPUDSStatus SvcClearDiagnosticInformation(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      UInt32 groupOfDTC);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcClearDiagnosticInformation")> _
Public Shared Function SvcClearDiagnosticInformation( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal groupOfDTC As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
groupOfDTC	A three-byte value indicating the group of DTCs (e.g. powertrain, body, chassis) or the particular DTC to be cleared (see PUDS_SVC_PARAM_CDI_xxx definitions)

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ClearDiagnosticInformation Request
result = UDSApi.SvcClearDiagnosticInformation(UDSApi.PUDS_USBBUS1, ref request,
0xF1A2B3);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ClearDiagnosticInformation Request
```

```
result = UDSApi::SvcClearDiagnosticInformation(UDSApi::PUDS_USBBUS1, *request,
0xF1A2B3);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ClearDiagnosticInformation Request
result = UDSApi.SvcClearDiagnosticInformation(UDSApi.PUDS_USBBUS1, request,
&HF1A2B3)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ClearDiagnosticInformation Request
  result := TUdsApi.SvcClearDiagnosticInformation(TUdsApi.PUDS_USBBUS1, request, $F1A2B3);
  if (result = PUDS_ERROR_OK) then
```

```
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127, PCAN-UDS Service Parameter Definitions on page 334: ClearDiagnosticInformation on page 335.

Plain function Version: UDS_SvcClearDiagnosticInformation.

3.6.43 SvcReadDTCInformation

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportNumberOfDTCByStatusMask, reportDTCByStatusMask, reportMirrorMemoryDTCByStatusMask, reportNumberOfMirrorMemoryDTCByStatusMask, reportNumberOfEmissionsRelatedOBDDTCByStatusMask, reportEmissionsRelatedOBDDTCByStatusMask Sub-functions are allowed.

Syntax

Pascal OO

```
class function SvcReadDTCInformation(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    RDTCIType: TPUDSSvcParamRDTCI;
    DTCStatusMask: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformation")]
public static extern TPUDSStatus SvcReadDTCInformation(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamRDTCI RDTCIType,
    byte DTCStatusMask);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformation")]
static TPUDSStatus SvcReadDTCInformation(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamRDTCI RDTCIType,
      Byte DTCStatusMask);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformation")> _
Public Shared Function SvcReadDTCInformation( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal RDTCIType As TPUDSSvcParamRDTCI, _
    ByVal DTCStatusMask As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RNODTCBSM, PUDS_SVC_PARAM_RDTCI_RDTCBSM, PUDS_SVC_PARAM_RDTCI_RMMDTCBSM, PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM, PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM, PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM.
DTCStatusMask	Contains eight DTC status bit.

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
```

```
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation Request
result = UDSApi.SvcReadDTCInformation(UDSApi.PUDS_USBBUS1, ref request,
    UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RNODTCBSM, 0xF1);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation Request
result = UDSApi::SvcReadDTCInformation(UDSApi::PUDS_USBBUS1, *request,
    UDSApi::TPUDSSvcParamRDTCI::PUDS_SVC_PARAM_RDTCI_RNODTCBSM, 0xF1);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation Request
result = UDSApi.SvcReadDTCInformation(UDSApi.PUDS_USBBUS1, request, _
    UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RNODTCBSM, &HF1)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
```

```
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation Request
  result := TUdsApi.SvcReadDTCInformation(TUdsApi.PUDS_USBBUS1, request,
    PUDS_SVC_PARAM_RDTCI_RNODTCBSM, $F1);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformation.

3.6.44 SvcReadDTCInformationRDTCSSBDTC

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. The sub-function reportDTCSnapshotRecordByDTCNumber (PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC) is implicit.

Syntax

Pascal OO

```
class function SvcReadDTCInformationRDTCSSBDTC(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DTCMask:LongWord;
```

```
    DTCSnapshotRecordNumber: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationRDTCSSBDTC")]
public static extern TPUDSStatus SvcReadDTCInformationRDTCSSBDTC(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    UInt32 DTCMask,
    byte DTCSnapshotRecordNumber);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationRDTCSSBDTC")]
static TPUDSStatus SvcReadDTCInformationRDTCSSBDTC(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      UInt32 DTCMask,
      Byte DTCSnapshotRecordNumber);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformationRDTCSSBDTC")> _
Public Shared Function SvcReadDTCInformationRDTCSSBDTC( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DTCMask As UInt32, _
    ByVal DTCSnapshotRecordNumber As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DTCStatusMask	A unique identification number (three byte value) for a specific diagnostic trouble code
DTCSnapshotRecordNumber	The number of the specific DTCSnapshot data records

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation 'reportDTCSnapshotRecordByDTCNumber' Request
result = UDSApi.SvcReadDTCInformationRDTCSSBDTC(UDSApi.PUDS_USBBUS1, ref request,
    0x00A1B2B3, 0x12);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation 'reportDTCSnapshotRecordByDTCNumber' Request
result = UDSApi::SvcReadDTCInformationRDTCSSBDTC(UDSApi::PUDS_USBBUS1, *request,
    0x00A1B2B3, 0x12);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation 'reportDTCSnapshotRecordByDTCNumber' Request
result = UDSApi.SvcReadDTCInformationRDTCSSBDTC(UDSApi.PUDS_USBBUS1, request, _
    &H00A1B2B3, &H12)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation 'reportDTCSnapshotRecordByDTCNumber' Request
  result := TUdsApi.SvcReadDTCInformationRDTCSSBDTC(TUdsApi.PUDS_USBBUS1, request,
    $00A1B2B3, $12);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformationRDTCSSBDTC.

3.6.45 SvcReadDTCInformationRDTCSSBRN

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. The sub-function reportDTCSnapshotByRecordNumber (PUDS_SVC_PARAM_RDTCI_RDTCSSBRN) is implicit.

Syntax

Pascal OO

```
class function SvcReadDTCInformationRDTCSSBRN(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DTCSnapshotRecordNumber: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationRDTCSSBRN")]
public static extern TPUDSStatus SvcReadDTCInformationRDTCSSBRN(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte DTCSnapshotRecordNumber);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationRDTCSSBRN")]
static TPUDSStatus SvcReadDTCInformationRDTCSSBRN(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      Byte DTCSnapshotRecordNumber);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformationRDTCSSBRN")> _
Public Shared Function SvcReadDTCInformationRDTCSSBRN( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DTCSnapshotRecordNumber As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DTCSnapshotRecordNumber	The number of the specific DTCSnapshot data records

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation 'reportDTCSnapshotByRecordNumber' Request
result = UDSApi.SvcReadDTCInformationRDTCSSBRN(UDSApi.PUDS_USBBUS1, ref request,
0x12);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation 'reportDTCSnapshotRecordByDTCNumber' Request
```

```
result = UDSApi::SvcReadDTCInformationRDTCSSBRN(UDSApi::PUDS_USBBUS1, *request,
0x12);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation 'reportDTCSnapshotByRecordNumber' Request
result = UDSApi.SvcReadDTCInformationRDTCSSBRN(UDSApi.PUDS_USBBUS1, request, &H12)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation 'reportDTCSnapshotByRecordNumber' Request
  result := TUdsApi.SvcReadDTCInformationRDTCSSBRN(TUdsApi.PUDS_USBBUS1, request, $12);
  if (result = PUDS_ERROR_OK) then
```

```
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformationRDTCSSBRN.

3.6.46 SvcReadDTCInformationReportExtended

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportDTCExtendedDataRecordByDTCNumber and reportMirrorMemoryDTCExtendedDataRecordByDTCNumber Sub-functions are allowed.

Syntax

Pascal OO

```
class function SvcReadDTCInformationReportExtended(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    RDTCIType: TPUDSSvcParamRDTCI;
    DTCMask: LongWord;
    DTCExtendedDataRecordNumber: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationReportExtended")]
public static extern TPUDSStatus SvcReadDTCInformationReportExtended(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamRDTCI RDTCIType,
    UInt32 DTCMask,
    byte DTCExtendedDataRecordNumber);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationReportExtended")]
static TPUDSStatus SvcReadDTCInformationReportExtended(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamRDTCI RDTCIType,
      UInt32 DTCMask,
      Byte DTCExtendedDataRecordNumber);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformationReportExtended")>
_
Public Shared Function SvcReadDTCInformationReportExtended( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal RDTCIType As TPUDSSvcParamRDTCI, _
    ByVal DTCMask As UInt32, _
    ByVal DTCExtendedDataRecordNumber As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN
DTCMask	A unique identification number (three byte value) for a specific diagnostic trouble code
DTCExtendedDataRecordNumber	The number of the specific DTCExtendedData record requested

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
```

```
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation ReportExtended Request
result = UDSApi.SvcReadDTCInformationReportExtended(UDSApi.PUDS_USBBUS1, ref
request,
    UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, 0x00A1B2B3, 0x12);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation ReportExtended Request
result = UDSApi::SvcReadDTCInformationReportExtended(UDSApi::PUDS_USBBUS1,
*request,
    UDSApi::TPUDSSvcParamRDTCI::PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, 0x00A1B2B3, 0x12);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation ReportExtended Request
result = UDSApi.SvcReadDTCInformationReportExtended(UDSApi.PUDS_USBBUS1, request, _
    UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, &H00A1B2B3, &H12)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
```

```
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation ReportExtended Request
  result := TUdsApi.SvcReadDTCInformationReportExtended(TUdsApi.PUDS_USBBUS1, request,
    PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, $00A1B2B3, $12);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformationReportExtended.

3.6.47 SvcReadDTCInformationReportSeverity

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportNumberOfDTCBySeverityMaskRecord and reportDTCSeverityInformation Sub-functions are allowed.

Syntax

Pascal OO

```
class function SvcReadDTCInformationReportSeverity(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    RDTCIType: TPUDSSvcParamRDTCI;
    DTCSeverityMask: Byte;
    DTCStatusMask: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationReportSeverity")]
public static extern TPUDSStatus SvcReadDTCInformationReportSeverity(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamRDTCI RDTCIType,
    byte DTCSeverityMask,
    byte DTCStatusMask);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationReportSeverity")]
static TPUDSStatus SvcReadDTCInformationReportSeverity(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamRDTCI RDTCIType,
      Byte DTCSeverityMask,
      Byte DTCStatusMask);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformationReportSeverity")>
_
Public Shared Function SvcReadDTCInformationReportSeverity( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal RDTCIType As TPUDSSvcParamRDTCI, _
    ByVal DTCSeverityMask As Byte, _
    ByVal DTCStatusMask As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, PUDS_SVC_PARAM_RDTCI_RDTCBSMR
DTCSeverityMask	A mask of eight (8) DTC severity bits (see TPUDSSvcParamRDTCI_DTCSVM).
DTCStatusMask	A mask of eight (8) DTC status bits.

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation ReportSeverity Request
result = UDSApi.SvcReadDTCInformationReportSeverity(UDSApi.PUDS_USBBUS1, ref
request,
    UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, 0xF1, 0x12);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation ReportSeverity Request
result = UDSApi::SvcReadDTCInformationReportSeverity(UDSApi::PUDS_USBBUS1,
*request,
    UDSApi::TPUDSSvcParamRDTCI::PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, 0xF1, 0x12);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation ReportSeverity Request
result = UDSApi.SvcReadDTCInformationReportSeverity(UDSApi.PUDS_USBBUS1, request, _
    UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, &HF1, &H12)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
```

```
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation ReportSeverity Request
  result := TUdsApi.SvcReadDTCInformationReportSeverity(TUdsApi.PUDS_USBBUS1, request,
    PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, $F1, $12);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformationReportSeverity.

3.6.48 SvcReadDTCInformationRSIODTC

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. The sub-function reportSeverityInformationOfDTC (PUDS_SVC_PARAM_RDTCI_RSIODTC) is implicit.

Syntax

Pascal OO

```
class function SvcReadDTCInformationRSIODTC(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DTCMask: LongWord): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationRSIODTC")]
public static extern TPUDSStatus SvcReadDTCInformationRSIODTC(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    UInt32 DTCMask);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationRSIODTC")]
static TPUDSStatus SvcReadDTCInformationRSIODTC(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      UInt32 DTCMask);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformationRSIODTC")> _
Public Shared Function SvcReadDTCInformationRSIODTC( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DTCMask As UInt32) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DTCMask	A unique identification number for a specific diagnostic trouble code

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
```

```
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation 'reportSeverityInformationOfDTC' Request
result = UDSApi.SvcReadDTCInformationRSIODTC(UDSApi.PUDS_USBBUS1, ref request,
0xF1A1B2B3);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation 'reportSeverityInformationOfDTC' Request
result = UDSApi::SvcReadDTCInformationRSIODTC(UDSApi::PUDS_USBBUS1, *request,
0xF1A1B2B3);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation 'reportSeverityInformationOfDTC' Request
result = UDSApi.SvcReadDTCInformationRSIODTC(UDSApi.PUDS_USBBUS1, request,
&HF1A1B2B3UI)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
```

```
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation 'reportSeverityInformationOfDTC' Request
  result := TUdsApi.SvcReadDTCInformationRSIODTC(TUdsApi.PUDS_USBBUS1, request, $F1A1B2B3);
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformationRSIODTC.

3.6.49 SvcReadDTCInformationNoParam

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportSupportedDTC, reportFirstTestFailedDTC, reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, reportMostRecentConfirmedDTC, reportDTCFaultDetectionCounter, reportDTCWithPermanentStatus, and reportDTCSnapshotIdentification Sub-functions are allowed.

Syntax

Pascal OO

```
class function SvcReadDTCInformationNoParam(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    RDTCIType: TPUDSSvcParamRDTCI): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationNoParam")]
public static extern TPUDSStatus SvcReadDTCInformationNoParam(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamRDTCI RDTCIType);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcReadDTCInformationNoParam")]
static TPUDSStatus SvcReadDTCInformationNoParam(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamRDTCI RDTCIType);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcReadDTCInformationNoParam")> _
Public Shared Function SvcReadDTCInformationNoParam( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal RDTCIType As TPUDSSvcParamRDTCI) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RFTFDTC, PUDS_SVC_PARAM_RDTCI_RFCDTC, PUDS_SVC_PARAM_RDTCI_RMRTFDTC, PUDS_SVC_PARAM_RDTCI_RMRCDTC, PUDS_SVC_PARAM_RDTCI_RSUPDTC, PUDS_SVC_PARAM_RDTCI_RDTCWPS, PUDS_SVC_PARAM_RDTCI_RDTCSSI.

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation NoParam Request
result = UDSApi.SvcReadDTCInformationNoParam(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RSUPDTC);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformation NoParam Request
```

```
result = UDSApi::SvcReadDTCInformationNoParam(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamRDTCI::PUDS_SVC_PARAM_RDTCI_RSUPDTC);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical ReadDTCInformation NoParam Request
result = UDSApi.SvcReadDTCInformationNoParam(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamRDTCI.PUDS_SVC_PARAM_RDTCI_RSUPDTC)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical ReadDTCInformation NoParam Request
  result := TUdsApi.SvcReadDTCInformationNoParam(TUdsApi.PUDS_USBBUS1, request,
PUDS_SVC_PARAM_RDTCI_RSUPDTC);
```

```
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcReadDTCInformationNoParam.

3.6.50 SvcInputOutputControlByIdentifier

Writes a UDS request according to the InputOutputControlByIdentifier service's specifications.

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

Syntax

Pascal OO

```
class function SvcInputOutputControlByIdentifier(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    DataIdentifier: Word;
    ControlOptionRecord: PByte;
    ControlOptionRecordLength: Word;
    ControlEnableMaskRecord: PByte;
    ControlEnableMaskRecordLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcInputOutputControlByIdentifier")]
public static extern TPUDSStatus SvcInputOutputControlByIdentifier(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    ushort DataIdentifier,
    byte[] ControlOptionRecord,
    ushort ControlOptionRecordLength,
    byte[] ControlEnableMaskRecord,
    ushort ControlEnableMaskRecordLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcInputOutputControlByIdentifier")]
static TPUDSStatus SvcInputOutputControlByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      unsigned short DataIdentifier,
      array<Byte>^ ControlOptionRecord,
      unsigned short ControlOptionRecordLength,
      array<Byte>^ ControlEnableMaskRecord,
```

```
      unsigned short ControlEnableMaskRecordLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcInputOutputControlByIdentifier")> _
Public Shared Function SvcInputOutputControlByIdentifier( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal DataIdentifier As UShort, _
    ByVal ControlOptionRecord As Byte(), _
    ByVal ControlOptionRecordLength As UShort, _
    ByVal ControlEnableMaskRecord As Byte(), _
    ByVal ControlEnableMaskRecordLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
ControlOptionRecord	First byte can be used as either an InputOutputControlParameter that describes how the server shall control its inputs or outputs (see TPUDSSvcParamIOCBI), or as an additional controlState byte
ControlOptionRecordLength	Size in bytes of the ControlOptionRecord buffer
ControlEnableMaskRecord	The ControlEnableMask shall only be supported when the inputOutputControlParameter is used and the dataIdentifier to be controlled consists of more than one parameter (i.e. the dataIdentifier is bit-mapped or packeted by definition). There shall be one bit in the ControlEnableMask corresponding to each individual parameter defined within the dataIdentifier
ControlEnableMaskRecordLength	Size in bytes of the controlEnableMaskRecord buffer

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical InputOutputControlByIdentifier Request
byte[] lBufferOption = new byte[20];
byte[] lBufferEnableMask = new byte[20];
// fill buffers [...]
result = UDSApi.SvcInputOutputControlByIdentifier(UDSApi.PUDS_USBBUS1, ref request,
    (ushort)UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_SSECUSWVNDID,
    lBufferOption, (byte)lBufferOption.Length, lBufferEnableMask,
(byte)lBufferEnableMask.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical InputOutputControlByIdentifier Request
array<Byte>^ lBufferOption = gcnew array<Byte>(20);
array<Byte>^ lBufferEnableMask = gcnew array<Byte>(20);
// fill buffers [...]
result = UDSApi::SvcInputOutputControlByIdentifier(UDSApi::PUDS_USBBUS1, *request,
    (unsigned short)UDSApi::TPUDSSvcParamDI::PUDS_SVC_PARAM_DI_SSECUSWVNDID,
    lBufferOption, (Byte)lBufferOption->Length, lBufferEnableMask,
(Byte)lBufferEnableMask->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical InputOutputControlByIdentifier Request
Dim lBufferOption(20) As Byte
Dim lBufferEnableMask(20) As Byte
' fill buffers [...]
result = UDSApi.SvcInputOutputControlByIdentifier(UDSApi.PUDS_USBBUS1, request, _
    UDSApi.TPUDSSvcParamDI.PUDS_SVC_PARAM_DI_SSECUSWVNDID, _
    lBufferOption, lBufferOption.Length, lBufferEnableMask,
lBufferEnableMask.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  lBufferOption: array[0..19] of Byte;
  lBufferEnableMask: array[0..19] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical InputOutputControlByIdentifier Request
  // fill buffers [...]
  result := TUdsApi.SvcInputOutputControlByIdentifier(TUdsApi.PUDS_USBBUS1, request,
    Word(PUDS_SVC_PARAM_DI_SSECUSWVNDID),
    @lBufferOption, Length(lBufferOption), @lBufferEnableMask, Length(lBufferEnableMask));
  if (result = PUDS_ERROR_OK) then
```

```
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcInputOutputControlByIdentifier.

3.6.51 SvcRoutineControl

Writes a UDS request according to the RoutineControl service's specifications.

The RoutineControl service is used by the client to start/stop a routine, and request routine results.

Syntax

Pascal OO

```
class function SvcRoutineControl(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    RoutineControlType: TPUDSSvcParamRC;
    RoutineIdentifier: Word;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRoutineControl")]
public static extern TPUDSStatus SvcRoutineControl(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    TPUDSSvcParamRC RoutineControlType,
    ushort RoutineIdentifier,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRoutineControl")]
static TPUDSStatus SvcRoutineControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      TPUDSSvcParamRC RoutineControlType,
      unsigned short RoutineIdentifier,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcRoutineControl")> _
```

```
Public Shared Function SvcRoutineControl( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal RoutineControlType As TPUDSSvcParamRC, _
    ByVal RoutineIdentifier As UShort, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
RoutineControlType	Subfunction parameter: RoutineControl type (see TPUDSSvcParamRC).
RoutineIdentifier	Server Local Routine Identifier (see TPUDSSvcParamRC_RID).
Buffer	Buffer containing the Routine Control Options (only with start and stop routine sub-functions).
BufferLength	Size in bytes of the buffer.

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;
```

```
// Sends a Physical RoutineControl Request
byte[] buffer = { 0xF1, 0xA1, 0xB2, 0xC3 };
result = UDSApi.SvcRoutineControl(UDSApi.PUDS_USBBUS1, ref request,
UDSApi.TPUDSSvcParamRC.PUDS_SVC_PARAM_RC_RRR,
    0xF1A2, buffer, (ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RoutineControl Request
array<Byte>^ buffer = { 0xF1, 0xA1, 0xB2, 0xC3 };
result = UDSApi::SvcRoutineControl(UDSApi::PUDS_USBBUS1, *request,
UDSApi::TPUDSSvcParamRC::PUDS_SVC_PARAM_RC_RRR,
    0xF1A2, buffer, (unsigned short) buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical RoutineControl Request
Dim buffer As Byte() = { &HF1, &HA1, &HB2, &HC3 }
result = UDSApi.SvcRoutineControl(UDSApi.PUDS_USBBUS1, request,
UDSApi.TPUDSSvcParamRC.PUDS_SVC_PARAM_RC_RRR, _
    &HF1A2, buffer,  buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
```

```
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  buffer: array[0..3] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical RoutineControl Request
  buffer[0] := $F0;
  buffer[1] := $A1;
  buffer[2] := $B2;
  buffer[3] := $C3;
  result := TUdsApi.SvcRoutineControl(TUdsApi.PUDS_USBBUS1, request, PUDS_SVC_PARAM_RC_RRR,
    $F1A2, @buffer, Length(buffer));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcRoutineControl.

3.6.52 SvcRequestDownload

Writes a UDS request according to the RequestDownload service's specifications.

The RequestDownload service is used by the client to initiate a data transfer from the client to the server (download).

Syntax

Pascal OO

```
class function SvcRequestDownload(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    CompressionMethod: Byte;
    EncryptingMethod: Byte;
    MemoryAddress: PByte;
    MemoryAddressLength: Byte;
    MemorySize: PByte;
    MemorySizeLength: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRequestDownload")]
public static extern TPUDSStatus SvcRequestDownload(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte CompressionMethod,
    byte EncryptingMethod,
    byte[] MemoryAddress,
    byte MemoryAddressLength,
    byte[] MemorySize,
    byte MemorySizeLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRequestDownload")]
static TPUDSStatus SvcRequestDownload(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      Byte CompressionMethod,
      Byte EncryptingMethod,
      array<Byte>^ MemoryAddress,
      Byte MemoryAddressLength,
      array<Byte>^ MemorySize,
      Byte MemorySizeLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcRequestDownload")> _
Public Shared Function SvcRequestDownload( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal CompressionMethod As Byte, _
    ByVal EncryptingMethod As Byte, _
    ByVal MemoryAddress As Byte(), _
    ByVal MemoryAddressLength As Byte, _
    ByVal MemorySize As Byte(), _
    ByVal MemorySizeLength As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
CompressionMethod	A nibble-value that specifies the "compressionMethod", the value 0x0 specifies that no compressionMethod is used
EncryptingMethod	A nibble-value that specifies the "encryptingMethod", the value 0x0 specifies that no encryptingMethod is used
MemoryAddress	Starting address of server memory to which data is to be written
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Used by the server to compare the uncompressed memory size with the total amount of data transferred during the TransferData service
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestDownload Request
byte[] lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
byte[] lBufferSize = { 0x01, 0x05 };
result = UDSApi.SvcRequestDownload(UDSApi.PUDS_USBBUS1, ref request, 0x01, 0x02,
```

```
    lBufferAddr, (byte)lBufferAddr.Length, lBufferSize, (byte)lBufferSize.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestDownload Request
array<Byte>^ lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
array<Byte>^ lBufferSize = { 0x01, 0x05 };
result = UDSApi::SvcRequestDownload(UDSApi::PUDS_USBBUS1, *request, 0x01, 0x02,
    lBufferAddr, (Byte)lBufferAddr->Length, lBufferSize, (Byte)lBufferSize-
>Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical RequestDownload Request
Dim lBufferAddr As Byte() = { &HF0, &HA1, &H00, &H13 }
Dim lBufferSize As Byte() = { &H01, &H05 }
result = UDSApi.SvcRequestDownload(UDSApi.PUDS_USBBUS1, request, &H01, &H02, _
    lBufferAddr, lBufferAddr.Length, lBufferSize, lBufferSize.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
```

```
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  lBufferAddr: array[0..3] of Byte;
  lBufferSize: array[0..1] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical RequestDownload Request
  lBufferAddr[0] := $F0;
  lBufferAddr[1] := $A1;
  lBufferAddr[2] := $00;
  lBufferAddr[3] := $13;
  lBufferSize[0] := $01;
  lBufferSize[1] := $05;
  result := TUdsApi.SvcRequestDownload(TUdsApi.PUDS_USBBUS1, request, $01, $02,
    @lBufferAddr, Length(lBufferAddr), @lBufferSize, Length(lBufferSize));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcRequestDownload.

3.6.53 SvcRequestUpload

Writes a UDS request according to the RequestUpload service's specifications.

The RequestUpload service is used by the client to initiate a data transfer from the server to the client (upload).

Syntax

Pascal OO

```
class function SvcRequestUpload(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    CompressionMethod: Byte;
    EncryptingMethod: Byte;
    MemoryAddress: PByte;
    MemoryAddressLength: Byte;
    MemorySize: PByte;
    MemorySizeLength: Byte): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRequestUpload")]
public static extern TPUDSStatus SvcRequestUpload(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte CompressionMethod,
    byte EncryptingMethod,
    byte[] MemoryAddress,
    byte MemoryAddressLength,
    byte[] MemorySize,
    byte MemorySizeLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRequestUpload")]
static TPUDSStatus SvcRequestUpload(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      Byte CompressionMethod,
      Byte EncryptingMethod,
      array<Byte>^ MemoryAddress,
      Byte MemoryAddressLength,
      array<Byte>^ MemorySize,
      Byte MemorySizeLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcRequestUpload")> _
Public Shared Function SvcRequestUpload( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal CompressionMethod As Byte, _
    ByVal EncryptingMethod As Byte, _
    ByVal MemoryAddress As Byte(), _
    ByVal MemoryAddressLength As Byte, _
    ByVal MemorySize As Byte(), _
    ByVal MemorySizeLength As Byte) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message

Parameters	Description
CompressionMethod	A nibble-value that specifies the "compressionMethod", the value 0x0 specifies that no compressionMethod is used
EncryptingMethod	A nibble-value that specifies the "encryptingMethod", the value 0x0 specifies that no encryptingMethod is used
MemoryAddress	Starting address of server memory to which data is to be written
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Used by the server to compare the uncompressed memory size with the total amount of data transferred during the TransferData service
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestUpload Request
byte[] lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
byte[] lBufferSize = { 0x01, 0x05 };
result = UDSApi.SvcRequestUpload(UDSApi.PUDS_USBBUS1, ref request, 0x01, 0x02,
    lBufferAddr, (byte)lBufferAddr.Length, lBufferSize, (byte)lBufferSize.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
```

```
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestUpload Request
array<Byte>^ lBufferAddr = { 0xF0, 0xA1, 0x00, 0x13 };
array<Byte>^ lBufferSize = { 0x01, 0x05 };
result = UDSApi::SvcRequestUpload(UDSApi::PUDS_USBBUS1, *request, 0x01, 0x02,
    lBufferAddr, (Byte)lBufferAddr->Length, lBufferSize, (Byte)lBufferSize-
>Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical RequestUpload Request
Dim lBufferAddr As Byte()= { &HF0, &HA1, &H00, &H13 }
Dim lBufferSize As Byte()= { &H01, &H05 }
result = UDSApi.SvcRequestUpload(UDSApi.PUDS_USBBUS1, request, &H01, &H02, _
    lBufferAddr, lBufferAddr.Length, lBufferSize, lBufferSize.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
```

```
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  lBufferAddr: array[0..3] of Byte;
  lBufferSize: array[0..1] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical RequestUpload Request
  lBufferAddr[0] := $F0;
  lBufferAddr[1] := $A1;
  lBufferAddr[2] := $00;
  lBufferAddr[3] := $13;
  lBufferSize[0] := $01;
  lBufferSize[1] := $05;
  result := TUdsApi.SvcRequestUpload(TUdsApi.PUDS_USBBUS1, request, $01, $02,
    @lBufferAddr, Length(lBufferAddr), @lBufferSize, Length(lBufferSize));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcRequestUpload.

3.6.54 SvcTransferData

Writes a UDS request according to the TransferData service's specifications.

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

Syntax

Pascal OO

```
class function SvcTransferData(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    BlockSequenceCounter: Byte;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcTransferData")]
public static extern TPUDSStatus SvcTransferData(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte BlockSequenceCounter,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcTransferData")]
static TPUDSStatus SvcTransferData(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      Byte BlockSequenceCounter,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcTransferData")> _
Public Shared Function SvcTransferData( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal BlockSequenceCounter As Byte, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
BlockSequenceCounter	The blockSequenceCounter parameter value starts at 01 hex with the first TransferData request that follows the RequestDownload (34 hex) or RequestUpload (35 hex) service. Its value is incremented by 1 for each subsequent TransferData request. At the value of FF hex, the blockSequenceCounter rolls over and starts at 00 hex with the next TransferData request message

Parameters	Description
Buffer	Buffer containing the required transfer parameters
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical TransferData Request
byte[] buffer = { 0xE3, 0xA1, 0xB2 };
result = UDSApi.SvcTransferData(UDSApi.PUDS_USBBUS1, ref request, 0x01, buffer,
(ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical TransferData Request
array<Byte>^ buffer = { 0xE3, 0xA1, 0xB2 };
result = UDSApi::SvcTransferData(UDSApi::PUDS_USBBUS1, *request, 0x01, buffer,
(unsigned short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical TransferData Request
Dim buffer As Byte() = { &HE3, &HA1, &HB2 }
result = UDSApi.SvcTransferData(UDSApi.PUDS_USBBUS1, request, &H01, buffer,
buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
   request: TPUDSMsg;
   requestConfirmation: TPUDSMsg;
   response: TPUDSMsg;
   result: TPUDSStatus;
   buffer: array[0..3] of Byte;

begin
   // initialization
   request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
   request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
   request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
   request.NETADDRINFO.RA := $00;
   request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

   // Sends a Physical TransferData Request
   buffer[0] := $E3;
   buffer[1] := $A1;
   buffer[2] := $B2;
   result := TUdsApi.SvcTransferData(TUdsApi.PUDS_USBBUS1, request, $01, @buffer, Length(buffer));
   if (result = PUDS_ERROR_OK) then
      result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
   if (result = PUDS_ERROR_OK) then
      MessageBox(0, 'Response was received.', 'Success', MB_OK)
   else
      // An error occurred
      MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcTransferData.

3.6.55 SvcRequestTransferExit

Writes a UDS request according to the RequestTransferExit service's specifications.

The RequestTransferExit service is used by the client to terminate a data transfer between client and server (upload or download).

Syntax

Pascal OO

```
class function SvcRequestTransferExit(
    CanChannel: TPUDSCANHandle;
    var MessageBuffer: TPUDSMsg;
    Buffer: PByte;
    BufferLength: Word): TPUDSStatus;
```

C#

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRequestTransferExit")]
public static extern TPUDSStatus SvcRequestTransferExit(
    TPUDSCANHandle CanChannel,
    ref TPUDSMsg MessageBuffer,
    byte[] Buffer,
    ushort BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-UDS.dll", EntryPoint = "UDS_SvcRequestTransferExit")]
static TPUDSStatus SvcRequestTransferExit(
      TPUDSCANHandle CanChannel,
      TPUDSMsg %MessageBuffer,
      array<Byte>^ Buffer,
      unsigned short BufferLength);
```

Visual Basic

```
<DllImport("PCAN-UDS.dll", EntryPoint:="UDS_SvcRequestTransferExit")> _
Public Shared Function SvcRequestTransferExit( _
    ByVal CanChannel As TPUDSCANHandle, _
    ByRef MessageBuffer As TPUDSMsg, _
    ByVal Buffer As Byte(), _
    ByVal BufferLength As UShort) As TPUDSStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A PUDS message
Buffer	Buffer containing the required transfer parameters
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C#:

```
TPUDSStatus result;
TPUDSMsg request = new TPUDSMsg();
TPUDSMsg requestConfirmation = new TPUDSMsg();
TPUDSMsg response = new TPUDSMsg();
// initialization
request.NETADDRINFO.SA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = (byte)TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestTransferExit Request
byte[] buffer = { 0xE3, 0xA1, 0xB2 };
result = UDSApi.SvcRequestTransferExit(UDSApi.PUDS_USBBUS1, ref request, buffer,
(ushort)buffer.Length);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, out response, ref request,
out requestConfirmation);
if (result == TPUDSStatus.PUDS_ERROR_OK)
    MessageBox.Show(String.Format("Response was received."));
else
    // An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", (int)result));
```

C++/CLR:

```
TPUDSStatus result;
TPUDSMsg^ request = gcnew TPUDSMsg();
TPUDSMsg^ requestConfirmation = gcnew TPUDSMsg();
TPUDSMsg^ response = gcnew TPUDSMsg();
// initialization
request->NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request->NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request->NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request->NETADDRINFO.RA = 0x00;
request->NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestTransferExit Request
array<Byte>^ buffer = { 0xE3, 0xA1, 0xB2 };
result = UDSApi::SvcRequestTransferExit(UDSApi::PUDS_USBBUS1, *request, buffer,
(unsigned short)buffer->Length);
if (result == PUDS_ERROR_OK)
      result = UDSApi::WaitForService(UDSApi::PUDS_USBBUS1, *response, *request,
*requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox::Show(String::Format("Response was received."));
else
      // An error occurred
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
```

Visual Basic:

```
Dim result As TPUDSStatus
Dim request As TPUDSMsg = New TPUDSMsg()
Dim requestConfirmation As TPUDSMsg = New TPUDSMsg()
Dim response As TPUDSMsg = New TPUDSMsg()
' initialization
request.NETADDRINFO.SA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT
request.NETADDRINFO.TA = TPUDSAddress.PUDS_ISO_15765_4_ADDR_ECU_1
request.NETADDRINFO.TA_TYPE = TPUDSAddressingType.PUDS_ADDRESSING_PHYSICAL
request.NETADDRINFO.RA = &H0
request.NETADDRINFO.PROTOCOL = TPUDSProtocol.PUDS_PROTOCOL_ISO_15765_2_11B

' Sends a Physical RequestTransferExit Request
Dim buffer As Byte() = { &HE3, &HA1, &HB2 }
result = UDSApi.SvcRequestTransferExit(UDSApi.PUDS_USBBUS1, request, buffer,
buffer.Length)
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    result = UDSApi.WaitForService(UDSApi.PUDS_USBBUS1, response, request,
requestConfirmation)
End If
If (result = TPUDSStatus.PUDS_ERROR_OK) Then
    MessageBox.Show(String.Format("Response was received."))
Else
    ' An error occurred
    MessageBox.Show(String.Format("Error occured: {0}", result.ToString()))
End If
```

Pascal OO:

```
var
  request: TPUDSMsg;
  requestConfirmation: TPUDSMsg;
  response: TPUDSMsg;
  result: TPUDSStatus;
  buffer: array[0..3] of Byte;

begin
  // initialization
  request.NETADDRINFO.SA := Byte(PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT);
  request.NETADDRINFO.TA := Byte(PUDS_ISO_15765_4_ADDR_ECU_1);
  request.NETADDRINFO.TA_TYPE := PUDS_ADDRESSING_PHYSICAL;
  request.NETADDRINFO.RA := $00;
  request.NETADDRINFO.PROTOCOL := PUDS_PROTOCOL_ISO_15765_2_11B;

  // Sends a Physical RequestTransferExit Request
  buffer[0] := $E3;
  buffer[1] := $A1;
  buffer[2] := $B2;
  result := TUdsApi.SvcRequestTransferExit(TUdsApi.PUDS_USBBUS1, request, @buffer, Length(buffer));
  if (result = PUDS_ERROR_OK) then
    result := TUdsApi.WaitForService(TUdsApi.PUDS_USBBUS1, response, request,
PTPUDSMsg(@requestConfirmation));
  if (result = PUDS_ERROR_OK) then
    MessageBox(0, 'Response was received.', 'Success', MB_OK)
  else
```

```
    // An error occurred
    MessageBox(0, 'An error occured', 'Error', MB_OK)
end;
```

See also: WaitForService on page 127.

Plain function Version: UDS_SvcRequestTransferExit.

3.7 Functions

The functions of the PCAN UDS are divided in 4 groups of functionality.

Connection

	Function	Description
	UDS_Initialize	Initializes a PUDS channel
	UDS_Uninitialize	Uninitializes a PUDS channel

Configuration

	Function	Description
	UDS_SetValue	Sets a configuration or information value within a PUDS Channel

Information

	Function	Description
	UDS_GetValue	Retrieves information from a PUDS Channel
	UDS_GetStatus	Retrieves the current BUS status of a PUDS Channel

Communication

	Function	Description
	UDS_Read	Reads a CAN message from the receive queue of a PUDS Channel
	UDS_Write	Writes to the transmit queue a CAN message using a connected PUDS Channel
	UDS_Reset	Resets the receive and transmit queues of a PUDS Channel
	UDS_WaitForSingleMessage	Waits for a confirmation or response message based on a UDS Message Request
	UDS_WaitForMultipleMessage	Waits for multiple reponses based on a UDS Message Request
	UDS_WaitForService	Waits for the confirmation of the transmission of a UDS service request and the reception of its response
	UDS_WaitForServiceFunctional	Waits for the confirmation of the transmission of a functional UDS service request and the reception of its responses
	UDS_ProcessResponse	Processes a UDS response message to update internal UDS features
	UDS_SvcDiagnosticSessionControl	Writes to the transmit queue a request for UDS service DiagnosticSessionControl
	UDS_SvcECUReset	Writes to the transmit queue a request for UDS service ECUReset
	UDS_SvcSecurityAccess	Writes to the transmit queue a request for UDS service SecurityAccess
	UDS_SvcCommunicationControl	Writes to the transmit queue a request for UDS service CommunicationControl
	UDS_SvcTesterPresent	Writes to the transmit queue a request for UDS service TesterPresent
	UDS_SvcSecuredDataTransmission	Writes to the transmit queue a request for UDS service SecuredDataTransmission
	UDS_SvcControlDTCSetting	Writes to the transmit queue a request for UDS service ControlDTCSetting
	UDS_SvcResponseOnEvent	Writes to the transmit queue a request for UDS service ResponseOnEvent
	UDS_SvcLinkControl	Writes to the transmit queue a request for UDS service LinkControl
	UDS_SvcReadDataByIdentifier	Writes to the transmit queue a request for UDS service ReadDataByIdentifier

	Function	Description
	UDS_SvcReadMemoryByAddress	Writes to the transmit queue a request for UDS service ReadMemoryByAddress
	UDS_SvcReadScalingDataByIdentifier	Writes to the transmit queue a request for UDS service ReadScalingDataByIdentifier
	UDS_SvcReadDataByPeriodicIdentifier	Writes to the transmit queue a request for UDS service ReadDataByPeriodicIdentifier
	UDS_SvcDynamicallyDefineDataIdentifierDBID	Writes to the transmit queue a request for UDS service DynamicallyDefineDataIdentifier
	UDS_SvcDynamicallyDefineDataIdentifierDBMA	Writes to the transmit queue a request for UDS service DynamicallyDefineDataIdentifier
	UDS_SvcDynamicallyDefineDataIdentifierCDDDI	Writes to the transmit queue a request for UDS service DynamicallyDefineDataIdentifier
	UDS_SvcWriteDataByIdentifier	Writes to the transmit queue a request for UDS service WriteDataByIdentifier
	UDS_SvcWriteMemoryByAddress	Writes to the transmit queue a request for UDS service WriteMemoryByAddress
	UDS_SvcClearDiagnosticInformation	Writes to the transmit queue a request for UDS service ClearDiagnosticInformation
	UDS_SvcReadDTCInformation	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcReadDTCInformationRDTCSSBDTC	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcReadDTCInformationRDTCSSBRN	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcReadDTCInformationReportExtended	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcReadDTCInformationReportSeverity	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcReadDTCInformationRSIODTC	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcReadDTCInformationNoParam	Writes to the transmit queue a request for UDS service ReadDTCInformation
	UDS_SvcInputOutputControlByIdentifier	Writes to the transmit queue a request for UDS service InputOutputControlByIdentifier
	UDS_SvcRoutineControl	Writes to the transmit queue a request for UDS service RoutineControl
	UDS_SvcRequestDownload	Writes to the transmit queue a request for UDS service RequestDownload
	UDS_SvcRequestUpload	Writes to the transmit queue a request for UDS service RequestUpload
	UDS_SvcTransferData	Writes to the transmit queue a request for UDS service TransferData
	UDS_SvcRequestTransferExit	Writes to the transmit queue a request for UDS service RequestTransferExit

3.7.1 UDS_Initialize

Initializes a PUDS Channel which represents a **Non**-Plug & Play PCAN-Device.

Syntax

C++

```
TPUDSStatus __stdcall UDS_Initialize(
        TPUDSHandle Channel,
        TPUDSBaudrate Baudrate,
        TPUDSHWType HwType = 0,
        DWORD IOPort = 0,
        WORD Interrupt = 0);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Baudrate	The speed for the communication (see TPUDSBaudrate on page 25)
HwType	The type of hardware (see TPUDSHWType on page 28)
IOPort	The I/O address for the parallel port
Interrupt	Interrupt number of the parallel port

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

PCANTP_ERROR_ALREADY_INITIALIZED	Indicates that the desired PUDS Channel is already in use
PCANTP_ERROR_CAN_ERROR	This error flag states that the error is composed of a more precise PCAN-Basic error

Remarks: As indicated by its name, the Initialize method initiates a PUDS 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 PUDS_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCUDS Channel means:

- to reserve the Channel for the calling application/process
- to allocate channel resources, like receive and transmit queues
- to forward initialization to PCAN-ISO-TP API and PCAN-Basic API, hence registering/connecting the Hardware denoted by the channel handle
- to set-up the default values of the different parameters (See GetValue) and configure default standard ISO-TP mappings:
    - 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)

The Initialization process will fail if an application tries to initialize a PUDS-Channel that has already been initialized within the same process.

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

Example

The following example shows the initialize and uninitialize processes for a Plug-And-Play channel (channel 2 of the PCAN-PCI).

C++

```
TPUDSStatus result;

// The Plug & Play Channel (PCAN-PCI) is initialized
result = UDS_Initialize(PUDS_PCIBUS2, PUDS_BAUD_500K);
if (result != PUDS_ERROR_OK)
      MessageBox(NULL, "Initialization failed", "Error", MB_OK);
else
      MessageBox(NULL, "PCAN-PCI (Ch-2) was initialized", "Success", MB_OK);

// All initialized channels are released
UDS_Uninitialize(PUDS_NONEBUS);
```

See also: UDS_Uninitialize below, UDS_GetValue on page 262, Understanding PCAN-UDS on page 8.

Class-method Version: UDS_Initialize.

3.7.2 UDS_Uninitialize

Uninitializes a PUDS Channel.

Syntax

C++

```
TPUDSStatus __stdcall UDS_Uninitialize(
        TPUDSCANHandle CanChannel);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)

Returns

The return value is a TPUDSStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

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

Remarks: A PUDS Channel can be released using one of this possibilities:

- **Single-Release:** Given a handle of a PUDS 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 PUDS_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

The following example shows the initialize and uninitializes processes for a Plug-And-Play channel (channel 2 of a PCAN-PCI hardware).

C++

```
TPUDSStatus result;

// The Plug & Play Channel (PCAN-PCI) is initialized
result = UDS_Initialize(PUDS_PCIBUS2, PUDS_BAUD_500K);
if (result != PUDS_ERROR_OK)
      MessageBox(NULL, "Initialization failed", "Error", MB_OK);
else
      MessageBox(NULL, "PCAN-PCI (Ch-2) was initialized", "Success", MB_OK);

// Release channel
UDS_Uninitialize(PUDS_PCIBUS2);
```

See also: UDS_Initialize on page 259.

Class-method version: Uninitilize.

3.7.3 UDS_SetValue

Sets a configuration or information value within a PUDS Channel.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SetValue(
        TPUDSCANHandle CanChannel,
        TPUDSParameter Parameter,
        void* Buffer,
            DWORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to be set (see TPUDSParameter on page 31)
Buffer	The buffer containing the numeric value to be set
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with an integer buffer

Remarks: Use the method SetValue to set configuration information or environment values of a PUDS Channel.

Note: That any calls with non ISO-TP parameters (ie. TPUDSParameter) will be forwarded to PCAN-Basic API.

More information about the parameters and values that can be set can be found in Parameter Value Definitions. Since most of the UDS parameters require a numeric value (byte or integer) this is the most common and useful override.

Example

he following example shows the use of the method SetValue on the channel PUDS_PCIBUS2 to enable debug mode.

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

C++

```
TPUDSStatus result;
unsigned int iBuffer = 0;

// Enable CAN DEBUG mode
iBuffer = PUDS_DEBUG_CAN;
result = UDS_SetValue(PUDS_PCIBUS2, PUDS_PARAM_DEBUG, &iBuffer, sizeof(unsigned
int));
if (result != PUDS_ERROR_OK)
      MessageBox(NULL, "Failed to set value", "Error", MB_OK);
else
      MessageBox(NULL, "Value changed successfully ", "Success", MB_OK);
```

See also: UDS_GetValue below, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Class-method Version: GetValue.

3.7.4 UDS_GetValue

Retrieves information from a PUDS Channel.

Syntax

C++

```
TPUDSStatus __stdcall UDS_GetValue(
        TPUDSCANHandle CanChannel,
        TPUDSParameter Parameter,
        void* Buffer,
        DWORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP Channel (see TPUDSCANHandle on page 22)
Parameter	The code of the value to retrieve (see TPUDSParameter)
Buffer	The buffer to return the required value
BufferLength	The length in bytes of the given buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_WRONG_PARAM	Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with a string buffer

Example

The following example shows the use of the method GetValue to retrieve the version of the ISO-TP API. Depending on the result, a message will be shown to the user.

C++

```
TPUDSCANHandle CanChannel = PUDS_USBBUS1;
TPUDSStatus result;
unsigned int iBuffer = 0;
char strMsg[256];

// Get the value of the ISO-TP Separation Time (STmin) parameter
result = UDS_GetValue(CanChannel, PUDS_PARAM_SEPERATION_TIME, &iBuffer,
sizeof(unsigned int));
if (result != PUDS_ERROR_OK)
      MessageBox(NULL, "Failed to get value", "Error", MB_OK);
else
{
      sprintf(strMsg, "%d", iBuffer);
      MessageBox(NULL, strMsg, "Success", MB_OK);
}
```

See also: UDS_SetValue on page 261, TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Class-method Version: GetValue.

3.7.5 UDS_GetStatus

Gets the current BUS status of a PUDS Channel.

Syntax

C++

```
TPUDSStatus __stdcall UDS_GetStatus(
              TPUDSCANHandle CanChannel);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_OK:	Indicates that the status of the given PUDS Channel is OK.

PUDS_ERROR_BUSLIGHT:	Indicates a bus error within the given PUDS Channel. The hardware is in bus-light status.
PUDS_ERROR_BUSHEAVY:	Indicates a bus error within the given PUDS Channel. The hardware is in bus-heavy status.
PUDS_ERROR_BUSOFF:	Indicates a bus error within the given PUDS Channel. The hardware is in bus-off status.
PUDS_ERROR_NOT_INITIALIZED:	Indicates that the given PUDS channel cannot be used because it 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 uninitialize 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 PUDS_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
TPUDSStatus result;

// Check the status of the PCI Channel
result = UDS_GetStatus(PUDS_PCIBUS1);
switch (result)
{
case PUDS_ERROR_BUSLIGHT:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...",
"Success", MB_OK);
      break;
case PUDS_ERROR_BUSHEAVY:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...",
"Success", MB_OK);
      break;
case PUDS_ERROR_BUSOFF:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-OFF status...", "Success",
MB_OK);
      break;
case PUDS_ERROR_OK:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Status is OK", "Success", MB_OK);
      break;
default:
      // An error occurred);
      MessageBox(NULL, "Failed to retrieve status", "Error", MB_OK);
      break;
}
```

See also: TPUDSParameter on page 31, Parameter Value Definitions on page 332.

Class-method Version: GetStatus.

3.7.6 UDS_Read

Reads a UDS message from the receive queue of a PUDS Channel.

Syntax

C++

```
TPUDSStatus __stdcall UDS_Read(
        TPUDSCANHandle CanChannel,
              TPUDSMsg* MessageBuffer);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Message Buffer	A TPUDSMsg buffer to store the CAN UDS message

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NO_MESSAGE	Indicates that the receive queue of the Channel is empty
PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application

Remarks: The message type (see TPUDSMessageType) of a CAN UDS message indicates if the message is a complete received UDS message, a transmission confirmation or an indication of a pending message. This value should be checked every time a message has been read successfully, along with the RESULT value as it contains the network status of the message.

Note: That higher level functions like WaitForSingleMessage or WaitForService should be preferred in cases where a client just has to read the response from a service request.

Example

The following example shows the use of the method Read on the channel PUDS_USBBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
TPCANTPStatus result;
TPCANTPMsg msg;
bool bStop = false;

do
{      // Read the first message in the queue
       result = UDS_Read(PUDS_USBBUS1, &msg);
       if (result == PUDS_ERROR_OK)
       {            // Process the received message
              MessageBox(NULL, "A message was received", "Success", MB_OK);
              //ProcessMessage(msg);
       }
       Else
       {
              // An error occurred
```

```
            MessageBox(NULL, "An error ocured", "Error", MB_OK);
            // Here can be decided if the loop has to be terminated
            //bStop = HandleReadError(result);
        }
} while (!bStop);
```

See also: Write.

Class-method Version: Read, UUDT Read/Write example p.344.

3.7.7 UDS_Write

Transmits a CAN UDS message.

Syntax

C++

```
TPUDSStatus __stdcall UDS_Write(
        TPUDSCANHandle CanChannel,
        TPUDSPMsg* MessageBuffer);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A TPUDSMsg buffer containing the CAN UDS message to be sent

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application or that a required CAN ID mapping was not found
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The Write function do not actually send the UDS message, the transmission is asynchronous. Should a message fail to be transmitted, it will be added to the reception queue with a specific network error code in the RESULT value of the TPUDSMsg.

Note: To transmit a standard UDS service request, it is recommended to use the corresponding API Service method starting with Svc (like `SvcDiagnosticSessionControl`).

Example

The following example shows the use of the method Write on the channel PUDS_USBBUS1. It adds to the transmit queue a UDS request from source 0xF1 to ECU #1 and then waits until a confirmation message is received. Depending on the result, a message will be shown to the user.

Note: it is assumed that the channel was already initialized and mapping were configured, the content of DATA is not initialized in the example.

C++

```
TPUDSStatus result;
// prepare an 11bit CAN ID, physically addressed UDS message containing 4095 BYTEs
of data
TPUDSMsg request;
char strMsg[256];

request.LEN = 4095;
request.MSGTYPE = PUDS_MESSAGE_TYPE_REQUEST;

request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDS_Write(PUDS_USBBUS1, &request);
if (result == PUDS_ERROR_OK)
{
      // Loop until the transmission confirmation is received
      do
      {
            result = UDS_Read(PUDS_USBBUS1, &request);
            sprintf(strMsg, "Read = %d, type=%d, result=%d", result,
request.MSGTYPE, request.RESULT);
            MessageBox(NULL, strMsg, "Error", MB_OK);
      } while (result == PUDS_ERROR_NO_MESSAGE);
}
else
{
      // An error occurred
      sprintf(strMsg, "Error occured: %d", result);
      MessageBox(NULL, strMsg, "Error", MB_OK);
}
```

See also: UDS_Read on page 265, UUDT Read/Write example p.344.

Class-method Version: Read.

3.7.8 UDS_Reset

Resets the receive and transmit queues of a PUDS Channel.

Syntax

C++

```
TPUDSStatus __stdcall UDS_Reset(
        TPUDSCANHandle CanChannel);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application

Remarks: Calling this method ONLY clear the queues of a Channel. A reset of the CAN controller doesn't take place.

Example

The following example shows the use of the method Reset on the channel PUDS_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
TPUDSStatus result;

// The PCI Channel is reset
result = UDS_Reset(PUDS_PCIBUS1);
if (result != PUDS_ERROR_OK)
{
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
}
else
      MessageBox(NULL, "PCAN-PCI (Ch-1) was reset", "Success", MB_OK);
```

See also: UDS_Uninitialize on page 260.

Class-method Version: Reset.

3.7.9 UDS_WaitForSingleMessage

Waits for a UDS response or transmit confirmation based on a UDS request.

Syntax

C++

```
TPUDSStatus __stdcall UDS_WaitForSingleMessage(
            TPUDSCANHandle CanChannel,
            TPUDSMsg * MessageBuffer,
            TPUDSMsg * MessageRequest,
            BOOL IsWaitForTransmit,
            DWORD TimeInterval,
            DWORD Timeout);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A TPUDSMsg buffer to store the UDS message

Parameters	Description
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously sent
IsWaitForTransmit	States whether the message to wait is a transmit confirmation or a UDS response
TimeInterval	Time to wait between polling for new UDS messages in milliseconds
Timeout	Maximum time to wait (in milliseconds) for a message indication corresponding to the MessageRequest. Note: a zero value means unlimited time

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel

Remarks: The Timeout parameter is ignored once a message indication matching the request is received (i.e. the first frame of the message). The function will then return once the message is fully received or a network error occurred.

To prevent unexpected locking, the user can abort the function by calling the function UDS_Reset (**class-method:** Reset).

Note: That the criteria to identify if a response matches the message request is based only on the network addressing information and the UDS service identifier: if a same service is requested multiple times with different parameters (like service ReadDataByIdentifier with different Data IDs), the user will have to ensure that the extra content matches the original request.

Example

The following example shows the use of the method WaitForSingleMessage on the channel PUDS_USBBUS1. It writes a UDS message on the CAN Bus and waits for the confirmation of the transmition. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg confirmation;

// prepare an 11bit CAN ID, physically addressed UDS  message containing 4095 Bytes
of data
memset(&request, 0, sizeof(TPUDSMsg));
memset(&confirmation, 0, sizeof(TPUDSMsg));
// [...] fill data
request.LEN = 4095;
request.MSGTYPE = PUDS_MESSAGE_TYPE_REQUEST;

request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
```

```
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDS_Write(PUDS_USBBUS1, &request);
if (result == PUDS_ERROR_OK)
{
      // wait for the transmit confirmation
      result = UDS_WaitForSingleMessage(PUDS_USBBUS1, &confirmation, &request,
true, 10, 100);
      if (result == PUDS_ERROR_OK)
            MessageBox(NULL, "Message was transmitted.", "Success", MB_OK);
      else
            // An error occurred
            MessageBox(NULL, "Error occured while waiting for transmit
confirmation.", "Error", MB_OK);
}
else
{
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
}
```

See also: UDS_Write on page 266.

Class-method Version: WaitForSingleMessage.

3.7.10 UDS_WaitForMultipleMessage

Waits for multiple UDS responses based on a UDS request (multiple responses can be obtained when a functional UDS request is transmitted).

Syntax

C++

```
TPUDSStatus __stdcall UDS_WaitForMultipleMessage(
            TPUDSCANHandle CanChannel,
            TPUDSMsg * Buffer,
            DWORD MaxCount,
            DWORD *pCount,
            TPUDSMsg * MessageRequest,
            DWORD TimeInterval,
            DWORD Timeout,
            DWORD TimeoutEnhanced,
            BOOLEAN WaitUntilTimeout);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Buffer	A buffer to store an array of TPUDSMsg
MaxCount	The maximum number of expected responses
pCount	Buffer to store the actual number of received responses
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously transmitted
TimeInterval	Time to wait between polling for new UDS messages in milliseconds
Timeout	Maximum time to wait (in milliseconds) for a message indication corresponding to the MessageRequest. Note: a zero value means unlimited time

Parameters	Description
TimeoutEnhanced	Maximum time to wait (in milliseconds) for a message indication corresponding to the MessageRequest if an ECU asks for extended timing (NRC_EXTENDED_TIMING)
WaitUntilTimeout	States whether the function is interrupted if the number of received messages reaches MaxCount

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_OVERFLOW	Success but the buffer limit was reached. Check pCount variable to see how many messages were discarded

Remarks: For each response, the Timeout/TimeoutEnhanced parameter is ignored once a message indication matching the request is received (i.e. the first frame of the message). The function will then return once all messages are fully received or a network error occurred.

To prevent unexpected locking, the user can abort the function by calling the function UDS_Reset (**class-method:** Reset).

Note: that the criteria to identify if a response matches the message request is based only on the network addressing information and the UDS service identifier: if a same service is requested multiple times with different parameters (like service ReadDataByIdentifier with different Data IDs), the user will have to ensure that the extra content matches the original request.

The function handles the negative response code `PUDS_NRC_EXTENDED_TIMING` (0x78) in order to fetch all responses at the same time: if such a response is read, the function will switch the default timeout to TimeoutEnhanced and wait for a new response.

The function automatically calls `ProcessResponse` on each received message.

Example

The following example shows the use of the method WaitForMultipleMessage on the channel PUDS_ USBBUS1. It writes a UDS functional message on the CAN Bus, waits for the confirmation of the transmission and then waits to receive responses from ECUs until timeout occurs. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg confirmation;
DWORD MessageArraySize = 5;
TPUDSMsg MessageArray[5];
DWORD count;

// prepare an 11bit CAN ID, functionally addressed UDS message
memset(&request, 0, sizeof(TPUDSMsg));
memset(&confirmation, 0, sizeof(TPUDSMsg));
memset(MessageArray, 0, sizeof(TPUDSMsg) * MessageArraySize);
// [...] fill data (functional message is limited to 1 CAN frame)
request.LEN = 7;
request.MSGTYPE = PUDS_MESSAGE_TYPE_REQUEST;

request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// The message is sent using the PCAN-USB
result = UDS_Write(PUDS_USBBUS1, &request);
if (result == PUDS_ERROR_OK)
{
      // wait for the transmit confirmation
      result = UDS_WaitForSingleMessage(PUDS_USBBUS1, &confirmation, &request,
true, 10, 100);
    if (result == PUDS_ERROR_OK && confirmation.RESULT == PUDS_RESULT_N_OK)
    {
            MessageBox(NULL, "Message was transmitted.", "Success", MB_OK);
        // wait for the responses
        result = UDS_WaitForMultipleMessage(PUDS_USBBUS1, MessageArray,
MessageArraySize, &count, &request, 10, 100, 1000, true);
        if (count > 0)
                MessageBox(NULL, "Responses were received", "Success", MB_OK);
            else
                MessageBox(NULL, "No response was received", "Error", MB_OK);
    }
      else
            // An error occurred
            MessageBox(NULL, "Error occured while waiting for transmit
confirmation.", "Error", MB_OK);
}
else
{
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
}
```

See also: UDS_Write on page 266, UDS_WaitForSingleMessage on page 268.

Class-method Version: WaitForMultipleMessage.

3.7.11 UDS_WaitForService

Handles the communication workflow for a UDS service expecting a single response. The function waits for a transmit confirmation then for a message response.

Syntax

C++

```
TPUDSStatus __stdcall UDS_WaitForService(
            TPUDSCANHandle CanChannel,
            TPUDSMsg * MessageBuffer,
            TPUDSMsg * MessageRequest,
            TPUDSMsg * MessageReqBuffer = NULL);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	A TPUDSMsg buffer to store the UDS response
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously written
MessageReqBuffer	A TPUDSMsg buffer to store the UDS transmit confirmation

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_OVERFLOW	A network error occured either in the transmit confirmation or the response message

Remarks: The WaitForService function is a utility function that calls other UDS API functions to simplify UDS communication workflow:

1. The function gets the defined timeouts (global API request and response timeouts and timeouts defined by the current session information with the ECU),
2. Waits for the confirmation of request's transmition,
3. On success, waits for the response confirmation.
4. If a negative response code is received stating that the ECU requires extended timing (`PUDS_NRC_EXTENDED_TIMING`, 0x78), the function switches to the enhanced timeout and waits for another response.
5. The function ProcessResponse is called automatically on the response.

Even if the SuppressPositiveResponseMessage flag is set in the UDS request, the function will still wait for an eventual Negative Response. If no error message is received the function will return PUDS_ERROR_NO_MESSAGE, although in this case it is must not be considered as an error. Moreover if a negative response code `PUDS_NRC_EXTENDED_TIMING` is received the SuppressPositiveResponseMessage flag is ignored as stated in ISO-14229-1.

Example

The following example shows the use of the method WaitForService on the channel PUDS_USBBUS1. A UDS physical service request is transmitted (service ECUReset), and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Message
result = UDS_SvcECUReset(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_ER_SR);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForServiceFunctional.

Class-method Version: WaitForService.

3.7.12 UDS_WaitForServiceFunctional

Handles the communication workflow for a UDS service requested with functional addressing, i.e. multiple responses can be expected. The function waits for a transmit confirmation then for responses.

Syntax

C++

```
TPUDSStatus __stdcall UDS_WaitForServiceFunctional(
            TPUDSCANHandle CanChannel,
            TPUDSMsg * Buffer,
            DWORD MaxCount,
            DWORD *pCount,
            BOOLEAN WaitUntilTimeout,
            TPUDSMsg * MessageRequest,
            TPUDSMsg * MessageReqBuffer = NULL);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
Buffer	A buffer to store an array of TPUDSMsg
MaxCount	The maximum number of expected responses
pCount	Buffer to store the actual number of received responses
WaitUntilTimeout	States whether the function is interrupted if the number of received messages reaches MaxCount
MessageRequest	A TPUDSMsg buffer containing the UDS message that was previously written
MessageReqBuffer	A TPUDSMsg buffer to store the UDS transmit confirmation

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_NO_MESSAGE	Indicates that no matching message was received in the time given
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_OVERFLOW	Success but the buffer limit was reached. Check pCount variable to see how many messages were discarded

Remarks: The WaitForServiceFunctional function is a utility function that calls other UDS API functions to simplify UDS communication workflow when requests involve functional addressing:

1. The function gets the defined timeouts (global API request and response timeouts and timeouts defined by the current session information with the ECU),
2. Waits for the confirmation of request's transmition,
3. On success, it waits for the confirmations of the responses like the function WaitForMultipleMessage would.
4. The function ProcessResponse is called on each response.

Example

The following example shows the use of the method WaitForServiceFunctional on the channel PUDS_USBBUS1. A UDS functional service request is transmitted (service ECUReset), and the WaitForServiceFunctional function is called to get the responses. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
WORD MessageArraySize = 5;
TPUDSMsg MessageArray[5];
DWORD count;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
```

```
memset(MessageArray, 0, sizeof(TPUDSMsg) * MessageArraySize);
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Message
result = UDS_SvcECUReset(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_ER_SR);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForServiceFunctional(PUDS_USBBUS1, MessageArray,
MessageArraySize, &count, true, &request, &requestConfirmation);
if (result == PUDS_ERROR_OK)
{
      if (count > 0)
            MessageBox(NULL, "Responses were received", "Success", MB_OK);
      else
            MessageBox(NULL, "No response was received", "Error", MB_OK);
}
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForSingleMessage on page 268, UDS_WaitForMultipleMessage on page 270, UDS_ProcessResponse below.

Class-method Version: WaitForServiceFunctional.

3.7.13 UDS_ProcessResponse

Processes a UDS response message to manage ISO-14229/15765 features, like session information.

Syntax

C++

```
TPUDSStatus __stdcall UDS_ProcessResponse(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
BufferBuffer	A TPUDSMsg buffer to store the UDS response

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_WRONG_PARAM	MessageBuffer is not valid (data length is zero or the network result indicates an error)

Remarks: The purpose of this function is to update internal UDS settings of the API: currently only the responses to DiagnosticSessionControl requests require this processing as they contain information on the active session.

Example

The following example shows the use of the method ProcessResponse on the channel PUDS_USBBUS1. It writes a UDS physical request (service DiagnosticSessionControl) on the CAN Bus, waits for the confirmation of the transmission and then waits to receive a response. Once received the ProcessResponse function is called on the received message, updating the session information inside the API.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg confirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&confirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_OBD_FUNCTIONAL;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ECUReset Message
result = UDS_SvcDiagnosticSessionControl(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DSC_DS);
if (result == PUDS_ERROR_OK)
{
    // wait for the transmit confirmation
      result = UDS_WaitForSingleMessage(PUDS_USBBUS1, &confirmation, &request,
true, 10, 100);
      if (result == PUDS_ERROR_OK)
      {
            // wait for a response
            result = UDS_WaitForSingleMessage(PUDS_USBBUS1, &response,
&confirmation, false, 10, 100);
            if (result == PUDS_ERROR_OK)
            {
                  result = UDS_ProcessResponse(PUDS_USBBUS1, &response);
                  MessageBox(NULL, "Response received", "Success", MB_OK);
            }
            else
                  MessageBox(NULL, "Error occured while waiting for response",
"Error", MB_OK);
      }
      else
                  MessageBox(NULL, "Error occured while waiting for transmit
confirmation", "Error", MB_OK);

}
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForSingleMessage on page 268.

Class-method Version: ProcessResponse.

3.7.14 UDS_SvcDiagnosticSessionControl

Writes a UDS request according to the DiagnosticSessionControl service's specifications.

The DiagnosticSessionControl service is used to enable different diagnostic sessions in the server.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcDiagnosticSessionControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE SessionType);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS Message
SessionType	Subfunction parameter: type of the session (see TPUDSSvcParamDSC)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

If this service is called with the NO_POSITIVE_RESPONSE_MSG parameter of the MessageBuffer set to ignore responses (i.e. value PUDS_SUPPR_POS_RSP_MSG_INDICATION_BIT), the API will automatically change the current session to the new one.

If the NO_POSITIVE_RESPONSE_MSG parameter is set to keep responses (i.e. PUDS_KEEP_POS_RSP_MSG_INDICATION_BIT), the session information will be updated when the response is received (only if WaitForService, WaitForMultipleMessage or the WaitForServiceFunctional is used, otherwise ProcessResponse function must be called).

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DiagnosticSessionControl Message
result = UDS_SvcDiagnosticSessionControl(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DSC_DS);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcDiagnosticSessionControl.

3.7.15 UDS_SvcECUReset

Writes a UDS request according to the ECUReset service's specifications.

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

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcECUReset(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE ResetType);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS Message
ResetType	Subfunction parameter: type of Reset (see TPUDSSvcParamER)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
DWORD dwBuffer;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical SecurityAccess Message
DWORD valueLittleEndian = 0xF0A1B2C3;
// Note: next a function is called to set MSB as 1st byte in the buffer
// (Win32 uses little endian format, UDS expects big endian)
dwBuffer = ReverseDword(&valueLittleEndian);
result = UDS_SvcSecurityAccess(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_SA_RSD_1,
(BYTE*) &dwBuffer, 4);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcECUReset.

3.7.16 UDS_SvcSecurityAccess

Writes a UDS request according to the SecurityAccess service's specifications.

SecurityAccess service provides a mean to access data and/or diagnostic services which have restricted access for security, emissions or safety reasons.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcSecurityAccess(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE SecurityAccessType,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS Message
SecurityAccessType	Subfunction parameter: type of SecurityAccess (see PUDS_SVC_PARAM_SA_xxx definitions)
Buffer	If Requesting Seed, buffer is the optional data to transmit to a server (like identification). If Sending Key, data holds the value generated by the security algorithm corresponding to a specific "seed" value
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

The example shows the use of a numeric buffer and points out the fact that Windows uses Little Endian Byte order and the UDS_SvcXXX functions expects data in Big Endian Byte Order.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
DWORD dwBuffer;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical SecuredDataTransmission Message
DWORD valueLittleEndian = 0xF0A1B2C3;
// Note: next a function is called to set MSB as 1st byte in the buffer
// (Win32 uses little endian format, UDS expects big endian)
dwBuffer = ReverseDword(&valueLittleEndian);
result = UDS_SvcSecuredDataTransmission(PUDS_USBBUS1, &request, (BYTE*) &dwBuffer,
4);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273, PCAN-UDS Service Parameter Definitions on page 334: SecurityAccess on page 334.

Class-method Version: SvcSecurityAccess.

3.7.17 UDS_SvcCommunicationControl

Writes a UDS request according to the CommunicationControl service's specifications.

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

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcCommunicationControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE ControlType,
      BYTE CommunicationType);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS Message
ControlType	Subfunction parameter: type of CommunicationControl (see TPUDSSvcParamCC)
CommunicationType	A bit-code value to reference the kind of communication to be controlled, see PUDS_SVC_PARAM_CC_FLAG_xxx flags and ISO_14229-2006 §B.1 for bit-encoding

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical CommunicationControl Message
result = UDS_SvcCommunicationControl(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_CC_ERXTX,
      PUDS_SVC_PARAM_CC_FLAG_APPL | PUDS_SVC_PARAM_CC_FLAG_NWM |
PUDS_SVC_PARAM_CC_FLAG_DENWRIRO);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
```

```
        MessageBox(NULL, "Response was received", "Success", MB_OK);
else
        // An error occurred
        MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273, PCAN-UDS Service Parameter Definitions on page 334: CommunicationControl on page 334.

Class-method Version: SvcCommunicationControl.

3.7.18 UDS_SvcTesterPresent

Writes a UDS request according to the TesterPresent service's specifications.

TesterPresent service indicates to a server (or servers) that a client is still connected to the vehicle and that certain diagnostic services and/or communications that have been previously activated are to remain active.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcTesterPresent(
        TPUDSCANHandle CanChannel,
        TPUDSMsg * MessageBuffer,
        BYTE TesterPresentType = PUDS_SVC_PARAM_TP_ZSUBF);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS Message
TesterPresentType	No Subfunction parameter by default (PUDS_SVC_PARAM_TP_ZSUBF)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a physical TesterPresent Message with no positive response
request.NO_POSITIVE_RESPONSE_MSG = PUDS_SUPPR_POS_RSP_MSG_INDICATION_BIT;
result = UDS_SvcTesterPresent(PUDS_USBBUS1, &request);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else if (result == PUDS_ERROR_NO_MESSAGE)
      MessageBox(NULL, "No error response", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcTesterPresent.

3.7.19 UDS_SvcSecuredDataTransmission

Writes a UDS request according to the SecuredDataTransmission service's specifications.

SecuredDataTransmission service's purpose is to transmit data that is protected against attacks from third parties, which could endanger data security.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcSecuredDataTransmission(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
Buffer	Buffer containing the data as processed by the Security Sub-Layer (See ISO-15764)

Parameters	Description
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
DWORD dwBuffer;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Note: next a function is called to set MSB as 1st byte in the buffer
// (Win32 uses little endian format, UDS expects big endian)
DWORD valueLittleEndian = 0xF0A1B2C3;
dwBuffer = ReverseDword(&valueLittleEndian);
// Sends a Physical SecuredDataTransmission Message
result = UDS_SvcSecuredDataTransmission(PUDS_USBBUS1, &request, (BYTE*) &dwBuffer,
4);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
```

```
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcSecuredDataTransmission.

3.7.20 UDS_SvcControlDTCSetting

Writes a UDS request according to the ControlDTCSetting service's specifications.

ControlDTCSetting service shall be used by a client to stop or resume the setting of diagnostic trouble codes (DTCs) in the server(s).

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcControlDTCSetting(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE DTCSettingType,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DTCSettingType	Subfunction parameter (see TPUDSSvcParamCDTCS)
Buffer	This parameter record is user-optional and transmits data to a server when controlling the DTC setting. It can contain a list of DTCs to be turned on or off
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel.
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
DWORD dwBuffer;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Note: next a function is called to set MSB as 1st byte in the buffer
// (Win32 uses little endian format, UDS expects big endian)
DWORD valueLittleEndian = 0xF0A1B2C3;
dwBuffer = ReverseDword(&valueLittleEndian);
// Sends a Physical ControlDTCSetting Message
result = UDS_SvcControlDTCSetting(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_CDTCS_OFF,
(BYTE*)&dwBuffer, 3);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcControlDTCSetting.

3.7.21 UDS_SvcResponseOnEvent

Writes a UDS request according to the ResponseOnEvent service's specifications.

The ResponseOnEvent service requests a server to start or stop transmission of responses on a specified event.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcResponseOnEvent(
      TPUDSCANHandle CanChannel,
```

```
    TPUDSMsg * MessageBuffer,
    BYTE EventType,
    BOOLEAN StoreEvent,
    BYTE EventWindowTime,
    BYTE * EventTypeRecord,
    WORD EventTypeRecordLength,
    BYTE * ServiceToRespondToRecord,
    WORD ServiceToRespondToRecordLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DTCSettingType	Subfunction parameter (see TPUDSSvcParamCDTCS)
EventType	Subfunction parameter: event type (see TPUDSSvcParamROE)
StoreEvent	Storage State (TRUE = Store Event, FALSE = Do Not Store Event)
EventWindowTime	Specify a window for the event logic to be active in the server (see also PUDS_SVC_PARAM_ROE_EWT_ITTR)
EventTypeRecord	Additional parameters for the specified eventType
EventTypeRecordLength	Size in bytes of the EventType Record (see PUDS_SVC_PARAM_ROE_xxx_LEN definitions)
ServiceToRespondToRecord	Service parameters, with first byte as service Id (see TPUDSSvcParamROERecommendedServiceID)
ServiceToRespondToRecordLength	Size in bytes of the ServiceToRespondTo Record

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE evBuffer[5];
```

```
BYTE siBuffer[5];

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ResponseOnEvent Message
evBuffer[0] = 0x08;
evBuffer[0] = PUDS_SI_ReadDTCInformation;
siBuffer[1] = PUDS_SVC_PARAM_RDTCI_RNODTCBSM;
siBuffer[2] = 0x01;
result = UDS_SvcResponseOnEvent(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_ROE_ONDTCS,
      FALSE, 0x08, evBuffer, 2, siBuffer, 2);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273, PCAN-UDS Service Parameter Definitions on page 334: ResponseOnEvent on page 334.

Class-method Version: SvcResponseOnEvent.

3.7.22 UDS_SvcLinkControl

Writes a UDS request according to the LinkControl service's specifications.

The LinkControl service is used to control the communication link baud rate between the client and the server(s) for the exchange of diagnostic data.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcLinkControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE LinkControlType,
      BYTE BaudrateIdentifier,
      DWORD LinkBaudrate = 0x0);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
LinkControlType	Subfunction parameter: Link Control Type (see TPUDSSvcParamLC)

Parameters	Description
BaudrateIdentifier	Defined baud rate identifier (see TPUDSSvcParamLCBaudrateIdentifier)
LinkBaudrate	Used only with PUDS_SVC_PARAM_LC_VBTWSBR parameter: a three-byte value baud rate (baudrate High, Middle and Low Bytes)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical LinkControl Message (Verify Fixed Baudrate)
result = UDS_SvcLinkControl(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_LC_VBTWFBR,
PUDS_SVC_PARAM_LC_BAUDRATE_CAN_250K, 0);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcLinkControl.

3.7.23 UDS_SvcReadDataByIdentifier

Writes a UDS request according to the ReadDataByIdentifier service's specifications.

The ReadDataByIdentifier service allows the client to request data record values from the server identified by one or more dataIdentifiers.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDataByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
Buffer	Buffer containing a list of two-byte Data Identifiers (see TPUDSSvcParamDI)
BufferLength	Number of elements in the buffer (size in WORD of the buffer)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
```

```
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDataByIdentifier Message
WORD buffer[2] = {PUDS_SVC_PARAM_DI_ADSDID, PUDS_SVC_PARAM_DI_ECUMDDID};
result = UDS_SvcReadDataByIdentifier(PUDS_USBBUS1, &request, buffer, 2);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDataByIdentifier.

3.7.24 UDS_SvcReadMemoryByAddress

Writes a UDS request according to the ReadMemoryByAddress service's specifications.

The ReadMemoryByAddress service allows the client to request memory data from the server via a provided starting address and to specify the size of memory to be read.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadMemoryByAddress(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE * MemoryAddress,
      BYTE MemoryAddressLength,
      BYTE * MemorySize,
      BYTE MemorySizeLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
MemoryAddress	Starting address of server memory from which data is to be retrieved
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Number of bytes to be read starting at the address specified by memoryAddress
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBufferAddr[10] = {};
BYTE lBufferSize[10] = {};
BYTE buffAddrLen = 10;
BYTE buffSizeLen = 3;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill data
for (int i = 0 ; i < buffAddrLen ; i++) {
      lBufferAddr[i] = 'A' + i;
      lBufferSize[i] = '1' + i;
}
// Sends a Physical ReadMemoryByAddress Message
result = UDS_SvcReadMemoryByAddress(PUDS_USBBUS1, &request, lBufferAddr,
buffAddrLen, lBufferSize, buffSizeLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
```

```
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadMemoryByAddress.

3.7.25 UDS_SvcReadScalingDataByIdentifier

Writes a UDS request according to the ReadScalingDataByIdentifier service's specifications.

The ReadScalingDataByIdentifier service allows the client to request scaling data record information from the server identified by a dataIdentifier.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadScalingDataByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DataIdentifier);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadScalingDataByIdentifier Message
result = UDS_SvcReadScalingDataByIdentifier(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DI_BSFPDID);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadScalingDataByIdentifier.

3.7.26 UDS_SvcReadDataByPeriodicIdentifier

Writes a UDS request according to the ReadDataByPeriodicIdentifier service's specifications.

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

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDataByPeriodicIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE TransmissionMode,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
TransmissionMode	Transmission rate code (see TPUDSSvcParamRDBPI)
Buffer	Buffer containing a list of Periodic Data Identifiers

Parameters	Description
BufferLength	Number of elements in the buffer (size in WORD of the buffer)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBuffer[10] = {};
WORD buffLen = 10;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill Data
for (int i = 0 ; i < buffLen ; i++) {
      lBuffer[i] = 'A' + i;
}
// Sends a Physical ReadDataByPeriodicIdentifier Message
result = UDS_SvcReadDataByPeriodicIdentifier(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_RDBPI_SAMR, lBuffer, buffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
```

```
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDataByPeriodicIdentifier.

3.7.27 UDS_SvcDynamicallyDefineDataIdentifierDBID

Writes a UDS request according to the DynamicallyDefineDataIdentifier service's specifications.

The DynamicallyDefineDataIdentifier service allows the client to dynamically define in a server a data identifier that can be read via the ReadDataByIdentifier service at a later time. The Define By Identifier subfunction specifies that definition of the dynamic data identifier shall occur via a data identifier reference.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcDynamicallyDefineDataIdentifierDBID(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DynamicallyDefinedDataIdentifier,
      WORD * SourceDataIdentifier,
      BYTE * MemorySize,
      BYTE * PositionInSourceDataRecord,
      WORD BuffersLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DynamicallyDefinedDataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
SourceDataIdentifier	Buffer containing the sources of information to be included into the dynamic data record
MemorySize	Buffer containing the total numbers of bytes from the source data record address
PositionInSourceDataRecord	Buffer containing the starting byte positions of the excerpt of the source data record
BuffersLength	Number of elements in the buffers (SourceDataIdentifier, MemoryAddress and PositionInSourceDataRecord)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
WORD lBufferSourceDI[10] = {};
BYTE lBufferMemSize[10] = {};
BYTE lBufferPosInSrc[10] = {};
WORD buffLen = 10;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill Data
for (int i = 0 ; i < buffLen ; i++) {
      lBufferSourceDI[i] = ((0xF0+i) << 8) + ('A' + i);
      lBufferMemSize[i] = i + 1;
      lBufferPosInSrc[i] = 100 + i;
}
// Sends a Physical DynamicallyDefineDataIdentifierDBID Message
result = UDS_SvcDynamicallyDefineDataIdentifierDBID(PUDS_USBBUS1, &request,
      PUDS_SVC_PARAM_DI_CDDID, lBufferSourceDI, lBufferMemSize, lBufferPosInSrc,
buffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: DynamicallyDefineDataIdentifier.

3.7.28 UDS_SvcDynamicallyDefineDataIdentifierDBMA

Writes a UDS request according to the DynamicallyDefineDataIdentifier service's specifications.

The DynamicallyDefineDataIdentifier service allows the client to dynamically define in a server a data identifier that can be read via the ReadDataByIdentifier service at a later time. The Define By Memory Address subfunction specifies that definition of the dynamic data identifier shall occur via an address reference.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcDynamicallyDefineDataIdentifierDBMA(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DynamicallyDefinedDataIdentifier,
      BYTE MemoryAddressLength,
      BYTE MemorySizeLength,
      BYTE * MemoryAddressBuffer,
      BYTE * MemorySizeBuffer,
      WORD BuffersLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DynamicallyDefinedDataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
MemoryAddressLength	Size in bytes of the MemoryAddress items in the MemoryAddressBuffer buffer (max.: 0xF)
MemorySizeLength	Size in bytes of the MemorySize items in the MemorySizeBuffer buffer (max.: 0xF)
MemoryAddressBuffer	Buffer containing the MemoryAddress buffers, must be an array of 'BuffersLength' entries which contains 'MemoryAddressLength' bytes (size is 'BuffersLength * MemoryAddressLength' bytes)
MemorySizeBuffer	Buffer containing the MemorySize buffers, must be an array of 'BuffersLength' entries which contains 'MemorySizeLength' bytes (size is 'BuffersLength * MemorySizeLength' bytes)
BuffersLength	Size in bytes of the MemoryAddressBuffer and MemorySizeBuffer buffers

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
WORD buffLen = 3;
BYTE lBuffsAddr[15] = {};
BYTE lBuffsSize[9] = {};
BYTE buffAddrLen = 5;
BYTE buffSizeLen = 3;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill data
for (int j = 0 ; j < buffLen ; j++)
{
      for (int i = 0 ; i < buffAddrLen ; i++) {
            lBuffsAddr[buffAddrLen*j+i] = (10 * j) + i + 1;
      }
      for (int i = 0 ; i < buffSizeLen ; i++) {
            lBuffsSize[buffSizeLen*j+i] = 100 + (10 * j) + i + 1;
      }
}
// Sends a Physical DynamicallyDefineDataIdentifierDBMA Message
result = UDS_SvcDynamicallyDefineDataIdentifierDBMA(PUDS_USBBUS1, &request,
      PUDS_SVC_PARAM_DI_CESWNDID, buffAddrLen, buffSizeLen, lBuffsAddr, lBuffsSize,
buffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcDynamicallyDefineDataIdentifierDBMA.

3.7.29 UDS_SvcDynamicallyDefineDataIdentifierCDDDI

Writes a UDS request according to the Clear Dynamically Defined Data Identifier service's specifications.

The Clear Dynamically Defined Data Identifier subfunction shall be used to clear the specified dynamic data identifier.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcDynamicallyDefineDataIdentifierCDDDI(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DynamicallyDefinedDataIdentifier);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DynamicallyDefinedDataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
```

```
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical DynamicallyDefineDataIdentifierCDDDI Message
result = UDS_SvcDynamicallyDefineDataIdentifierCDDDI(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DI_CESWNDID);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcDynamicallyDefineDataIdentifierCDDDI.

3.7.30 UDS_SvcWriteDataByIdentifier

Writes a UDS request according to the WriteDataByIdentifier service's specifications.

The WriteDataByIdentifier service allows the client to write information into the server at an internal location specified by the provided data identifier.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcWriteDataByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DataIdentifier,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
Buffer	Buffer containing the data to write
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel

PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBuffer[10] = {};
WORD buffLen = 10;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill Data
for (int i = 0 ; i < buffLen ; i++) {
      lBuffer[i] = 'A' + i;
}
// Sends a Physical WriteDataByIdentifier Message
result = UDS_SvcWriteDataByIdentifier(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DI_ASFPDID, lBuffer, buffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcWriteDataByIdentifier.

3.7.31 UDS_SvcWriteMemoryByAddress

Writes a UDS request according to the WriteMemoryByAddress service's specifications.

The WriteMemoryByAddress service allows the client to write information into the server at one or more contiguous memory locations.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcWriteMemoryByAddress(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DataIdentifier,
      BYTE * MemoryAddress,
      BYTE MemoryAddressLength,
      BYTE * MemorySize,
      BYTE MemorySizeLength,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
MemoryAddress	Starting address of server memory to which data is to be written
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Number of bytes to be written starting at the address specified by memoryAddress
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)
Buffer	Buffer containing the data to write
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBuffer[50] = {};
BYTE lBufferMemAddr[50] = {};
BYTE lBufferMemSize[50] = {};
WORD buffLen = 50;
BYTE buffAddrLen = 5;
BYTE buffSizeLen = 3;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill Data
for (int i = 0 ; i < buffLen ; i++) {
      lBuffer[i] = i+1;
      lBufferMemAddr[i] = 'A' + i;
      lBufferMemSize[i] = 10 + i;
}
// Sends a Physical WriteMemoryByAddress Message
result = UDS_SvcWriteMemoryByAddress(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DI_ASFPDID,
      lBufferMemAddr, buffAddrLen, lBufferMemSize, buffSizeLen, lBuffer, buffLen);
result = UDS_SvcECUReset(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_ER_SR);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcWriteMemoryByAddress.

3.7.32 UDS_SvcClearDiagnosticInformation

Writes a UDS request according to the ClearDiagnosticInformation service's specifications.

The ClearDiagnosticInformation service is used by the client to clear diagnostic information in one server's or multiple servers' memory.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcClearDiagnosticInformation(
```

```
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      DWORD groupOfDTC);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
groupOfDTC	A three-byte value indicating the group of DTCs (e.g. powertrain, body, chassis) or the particular DTC to be cleared (see PUDS_SVC_PARAM_CDI_xxx definitions)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ClearDiagnosticInformation Message
result = UDS_SvcClearDiagnosticInformation(PUDS_USBBUS1, &request, 0xF1A2B3);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
```

```
        MessageBox(NULL, "Response was received", "Success", MB_OK);
else
        // An error occurred
        MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273, PCAN-UDS Service Parameter Definitions on page 334: ClearDiagnosticInformation on page 335.

Class-method Version: SvcClearDiagnosticInformation.

3.7.33 UDS_SvcReadDTCInformation

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportNumberOfDTCByStatusMask, reportDTCByStatusMask, reportMirrorMemoryDTCByStatusMask, reportNumberOfMirrorMemoryDTCByStatusMask, reportNumberOfEmissionsRelatedOBDDTCByStatusMask, reportEmissionsRelatedOBDDTCByStatusMask Sub-functions are allowed.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformation(
        TPUDSCANHandle CanChannel,
        TPUDSMsg * MessageBuffer,
        BYTE RDTCIType,
        BYTE DTCStatusMask);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RNODTCBSM, PUDS_SVC_PARAM_RDTCI_RDTCBSM, PUDS_SVC_PARAM_RDTCI_RMMDTCBSM, PUDS_SVC_PARAM_RDTCI_RNOMMDTCBSM, PUDS_SVC_PARAM_RDTCI_RNOOBDDTCBSM, PUDS_SVC_PARAM_RDTCI_ROBDDTCBSM
DTCStatusMask	Contains eight DTC status bit

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid or RDTCIType is not allowed
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationRDTCSSBDTC Message
result = UDS_SvcReadDTCInformationRDTCSSBDTC(PUDS_USBBUS1, &request, 0x00A1B2B3,
0x12);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformation.

3.7.34 UDS_SvcReadDTCInformationRDTCSSBDTC

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. The sub-function reportDTCSnapshotRecordByDTCNumber (PUDS_SVC_PARAM_RDTCI_RDTCSSBDTC) is implicit.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformationRDTCSSBDTC(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      DWORD DTCMask,
      BYTE DTCSnapshotRecordNumber);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DTCMask	A unique identification number (three byte value) for a specific diagnostic trouble code
DTCSnapshotRecordNumber	The number of the specific DTCSnapshot data records

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
```

```
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationRDTCSSBDTC Message
result = UDS_SvcReadDTCInformationRDTCSSBDTC(PUDS_USBBUS1, &request, 0x00A1B2B3,
0x12);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformationRDTCSSBDTC.

3.7.35 UDS_SvcReadDTCInformationRDTCSSBRN

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. The sub-function reportDTCSnapshotByRecordNumber (PUDS_SVC_PARAM_RDTCI_RDTCSSBRN) is implicit.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformationRDTCSSBRN(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE DTCSnapshotRecordNumber);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DTCSnapshotRecordNumber	The number of the specific DTCSnapshot data records

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid

PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationRDTCSSBRN Message
result = UDS_SvcReadDTCInformationRDTCSSBRN(PUDS_USBBUS1, &request, 0x12);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformationRDTCSSBRN.

3.7.36 UDS_SvcReadDTCInformationReportExtended

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportDTCExtendedDataRecordByDTCNumber and reportMirrorMemoryDTCExtendedDataRecordByDTCNumber Sub-functions are allowed.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformationReportExtended(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE RDTCIType,
      DWORD DTCMask,
      BYTE DTCExtendedDataRecordNumber);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, PUDS_SVC_PARAM_RDTCI_RMMDEDRBDN
DTCMask	A unique identification number (three byte value) for a specific diagnostic trouble code
DTCExtendedDataRecordNumber	The number of the specific DTCExtendedData record requested

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid or RDTCIType is not allowed
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
```

```
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationReportExtended Message
result = UDS_SvcReadDTCInformationReportExtended(PUDS_USBBUS1, &request,
      PUDS_SVC_PARAM_RDTCI_RDTCEDRBDN, 0x00A1B2B3, 0x12);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformationReportExtended.

3.7.37 UDS_SvcReadDTCInformationReportSeverity

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportNumberOfDTCBySeverityMaskRecord and reportDTCSeverityInformation Sub-functions are allowed.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformationReportSeverity(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE RDTCIType,
      BYTE DTCSeverityMask,
      BYTE DTCStatusMask);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, PUDS_SVC_PARAM_RDTCI_RDTCBSMR
DTCSeverityMask	A mask of eight (8) DTC severity bits (see TPUDSSvcParamRDTCI_DTCSVM)
DTCStatusMask	A mask of eight (8) DTC status bits

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid or RDTCIType is not allowed
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationReportSeverity Message
result = UDS_SvcReadDTCInformationReportSeverity(PUDS_USBBUS1, &request,
      PUDS_SVC_PARAM_RDTCI_RNODTCBSMR, 0xF1, 0x12);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformationReportSeverity.

3.7.38 UDS_SvcReadDTCInformationRSIODTC

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. The sub-function reportSeverityInformationOfDTC (PUDS_SVC_PARAM_RDTCI_RSIODTC) is implicit.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformationRSIODTC(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      DWORD DTCMask);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DTCMask	A unique identification number for a specific diagnostic trouble code.

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
```

```
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationRSIODTC Message
result = UDS_SvcReadDTCInformationRSIODTC(PUDS_USBBUS1, &request, 0xF1A1B2B3);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformationRSIODTC.

3.7.39 UDS_SvcReadDTCInformationNoParam

Writes a UDS request according to the ReadDTCInformation service's specifications.

This service allows a client to read the status of server-resident Diagnostic Trouble Code (DTC) information. Only reportSupportedDTC, reportFirstTestFailedDTC, reportFirstConfirmedDTC, reportMostRecentTestFailedDTC, reportMostRecentConfirmedDTC, reportDTCFaultDetectionCounter, reportDTCWithPermanentStatus, and reportDTCSnapshotIdentification Sub-functions are allowed.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcReadDTCInformationNoParam(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE RDTCIType);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
RDTCIType	Subfunction parameter: ReadDTCInformation type, use one of the following: PUDS_SVC_PARAM_RDTCI_RFTFDTC, PUDS_SVC_PARAM_RDTCI_RFCDTC, PUDS_SVC_PARAM_RDTCI_RMRTFDTC, PUDS_SVC_PARAM_RDTCI_RMRCDTC, PUDS_SVC_PARAM_RDTCI_RSUPDTC, PUDS_SVC_PARAM_RDTCI_RDTCWPS, PUDS_SVC_PARAM_RDTCI_RDTCSSI

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid or RDTCIType is not allowed
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical ReadDTCInformationNoParam Message
result = UDS_SvcReadDTCInformationNoParam(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_RDTCI_RSUPDTC);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcReadDTCInformationNoParam.

3.7.40 UDS_SvcInputOutputControlByIdentifier

Writes a UDS request according to the InputOutputControlByIdentifier service's specifications.

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

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcInputOutputControlByIdentifier(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      WORD DataIdentifier,
      BYTE * ControlOptionRecord,
      WORD ControlOptionRecordLength,
      BYTE * ControlEnableMaskRecord,
      WORD ControlEnableMaskRecordLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
DataIdentifier	A two-byte Data Identifier (see TPUDSSvcParamDI)
ControlOptionRecord	First byte can be used as either an InputOutputControlParameter that describes how the server shall control its inputs or outputs (see TPUDSSvcParamIOCBI), or as an additional controlState byte
ControlOptionRecordLength	Size in bytes of the ControlOptionRecord buffer
ControlEnableMaskRecord	The ControlEnableMask shall only be supported when the inputOutputControlParameter is used and the dataIdentifier to be controlled consists of more than one parameter (i.e. the dataIdentifier is bit-mapped or packeted by definition). There shall be one bit in the ControlEnableMask corresponding to each individual parameter defined within the dataIdentifier
ControlEnableMaskRecordLength	Size in bytes of the controlEnableMaskRecord buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBufferOption[20] = {};
BYTE lBufferEnableMask[20] = {};
WORD lBuffOptionLen = 10;
WORD lBuffMaskLen = 5;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill Data
for (int i = 0 ; i < lBuffOptionLen ; i++) {
      lBufferOption[i] = 'A' + i;
      lBufferEnableMask[i] = 10 + i;
}
// Sends a Physical InputOutputControlByIdentifier Message
result = UDS_SvcInputOutputControlByIdentifier(PUDS_USBBUS1, &request,
PUDS_SVC_PARAM_DI_SSECUSWVNDID,
      lBufferOption, lBuffOptionLen, lBufferEnableMask, lBuffMaskLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcInputOutputControlByIdentifier.

3.7.41 UDS_SvcRoutineControl

Writes a UDS request according to the RoutineControl service's specifications.

The RoutineControl service is used by the client to start/stop a routine, and request routine results.

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcRoutineControl(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE RoutineControlType,
```

```
    WORD RoutineIdentifier,
    BYTE * Buffer,
    WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message
RoutineControlType	Subfunction parameter: RoutineControl type (see TPUDSSvcParamRC)
RoutineIdentifier	Server Local Routine Identifier (see TPUDSSvcParamRC_RID)
Buffer	Buffer containing the Routine Control Options (only with start and stop routine sub-functions)
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBuffer[20] = {};
WORD lBuffLen = 10;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill data
for (int i = 0 ; i < lBuffLen ; i++) {
      lBuffer[i] = 'A' + i;
}
// Sends a Physical RoutineControl Message
result = UDS_SvcRoutineControl(PUDS_USBBUS1, &request, PUDS_SVC_PARAM_RC_RRR,
      0xF1A2, lBuffer, lBuffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcRoutineControl.

3.7.42 UDS_SvcRequestDownload

Writes a UDS request according to the RequestDownload service's specifications.

The RequestDownload service is used by the client to initiate a data transfer from the client to the server (download).

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcRequestDownload(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE CompressionMethod,
      BYTE EncryptingMethod,
      BYTE * MemoryAddress,
      BYTE MemoryAddressLength,
      BYTE * MemorySize,
      BYTE MemorySizeLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
CompressionMethod	A nibble-value that specifies the "compressionMethod", the value 0x0 specifies that no compressionMethod is used
EncryptingMethod	A nibble-value that specifies the "encryptingMethod", the value 0x0 specifies that no encryptingMethod is used
MemoryAddress	Starting address of server memory to which data is to be written
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)
MemorySize	Used by the server to compare the uncompressed memory size with the total amount of data transferred during the TransferData service
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBufferMemAddr[50] = {};
BYTE lBufferMemSize[50] = {};
BYTE buffAddrLen = 15;
BYTE buffSizeLen = 8;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
```

```
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill data
for (int i = 0 ; i < buffAddrLen ; i++) {
      lBufferMemAddr[i] = 'A' + i;
      lBufferMemSize[i] = 10 + i;
}
// Sends a Physical RequestDownload Message
result = UDS_SvcRequestDownload(PUDS_USBBUS1, &request, 0x01, 0x02,
      lBufferMemAddr, buffAddrLen, lBufferMemSize, buffSizeLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcRequestDownload.

3.7.43 UDS_SvcRequestUpload

Writes a UDS request according to the RequestUpload service's specifications.

The RequestUpload service is used by the client to initiate a data transfer from the server to the client (upload).

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcRequestUpload(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE CompressionMethod,
      BYTE EncryptingMethod,
      BYTE * MemoryAddress,
      BYTE MemoryAddressLength,
      BYTE * MemorySize,
      BYTE MemorySizeLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
CompressionMethod	A nibble-value that specifies the "compressionMethod", the value 0x0 specifies that no compressionMethod is used
EncryptingMethod	A nibble-value that specifies the "encryptingMethod", the value 0x0 specifies that no encryptingMethod is used
MemoryAddress	Starting address of server memory from which data is to be retrieved
MemoryAddressLength	Size in bytes of the MemoryAddress buffer (max.: 0xF)

Parameters	Description
MemorySize	Used by the server to compare the uncompressed memory size with the total amount of data transferred during the TransferData service
MemorySizeLength	Size in bytes of the MemorySize buffer (max.: 0xF)

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBufferMemAddr[50] = {};
BYTE lBufferMemSize[50] = {};
BYTE buffAddrLen = 21;
BYTE buffSizeLen = 32;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Sends a Physical RequestUpload Message
result = UDS_SvcRequestUpload(PUDS_USBBUS1, &request, 0x01, 0x02,
      lBufferMemAddr, buffAddrLen, lBufferMemSize, buffSizeLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
```

```
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcRequestUpload.

3.7.44 UDS_SvcTransferData

Writes a UDS request according to the TransferData service's specifications.

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

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcTransferData(
      TPUDSCANHandle CanChannel,
      TPUDSMsg * MessageBuffer,
      BYTE BlockSequenceCounter,
      BYTE * Buffer,
      WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
BlockSequenceCounter	The blockSequenceCounter parameter value starts at 01 hex with the first TransferData request that follows the RequestDownload (34 hex) or RequestUpload (35 hex) service. Its value is incremented by 1 for each subsequent TransferData request. At the value of FF hex, the blockSequenceCounter rolls over and starts at 00 hex with the next TransferData request message
Buffer	Buffer containing the required transfer parameters
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBuffer[50] = {};
BYTE buffLen = 50;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill data
for (int i = 0 ; i < buffLen ; i++) {
      lBuffer[i] = 'A' + i;
}
// Sends a Physical TransferData Message
result = UDS_SvcTransferData(PUDS_USBBUS1, &request, 0x01, lBuffer, buffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcTransferData.

3.7.45 UDS_SvcRequestTransferExit

Writes a UDS request according to the RequestTransferExit service's specifications.

The RequestTransferExit service is used by the client to terminate a data transfer between client and server (upload or download).

Syntax

C++

```
TPUDSStatus __stdcall UDS_SvcRequestTransferExit(
      TPUDSCANHandle CanChannel,
```

```
    TPUDSMsg * MessageBuffer,
    BYTE * Buffer,
    WORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PUDS Channel (see TPUDSCANHandle on page 22)
MessageBuffer	The PUDS message (NO_POSITIVE_RESPONSE_MSG is ignored)
Buffer	Buffer containing the required transfer parameters
BufferLength	Size in bytes of the buffer

Returns

The return value is a TPUDSStatus code. PUDS_ERROR_OK is returned on success. The typical errors in case of failure are:

PUDS_ERROR_NOT_INITIALIZED	Indicates that the given PUDS channel was not found in the list of initialized channels of the calling application
PUDS_ERROR_TIMEOUT	Failed to gain access to the API mutual exclusion or the function was aborted by a call to reset the channel
PUDS_ERROR_WRONG_PARAM	The network addressing information of the message is not valid
PUDS_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue
PUDS_ERROR_OVERFLOW:	Buffer size is too big, the resulting UDS message data size is bigger than the maximum data length

Remarks: The function reads the MessageBuffer NETADDRINFO parameter and sets the DATA with the given parameters according to the service's specifications. It then writes the message to the transmit queue.

Example

The following example shows the use of the service method on the channel PUDS_USBBUS1. A UDS physical service request is transmitted, and the WaitForService function is called to get the response. Depending on the result, a message will be shown to the user.

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

C++:

```
TPUDSStatus result;
TPUDSMsg request;
TPUDSMsg requestConfirmation;
TPUDSMsg response;
BYTE lBuffer[50] = {};
BYTE buffLen = 20;

// initialization
memset(&request, 0, sizeof(TPUDSMsg));
memset(&requestConfirmation, 0, sizeof(TPUDSMsg));
memset(&response, 0, sizeof(TPUDSMsg));
request.NETADDRINFO.SA = PUDS_ISO_15765_4_ADDR_TEST_EQUIPMENT;
request.NETADDRINFO.TA = PUDS_ISO_15765_4_ADDR_ECU_1;
request.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
request.NETADDRINFO.RA = 0x00;
request.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;

// Fill data
for (int i = 0 ; i < buffLen ; i++) {
```

```
      lBuffer[i] = 'A' + i;
}
// Sends a Physical RequestTransferExit Message
result = UDS_SvcRequestTransferExit(PUDS_USBBUS1, &request, lBuffer, buffLen);
if (result == PUDS_ERROR_OK)
      result = UDS_WaitForService(PUDS_USBBUS1, &response, &request,
&requestConfirmation);
if (result == PUDS_ERROR_OK)
      MessageBox(NULL, "Response was received", "Success", MB_OK);
else
      // An error occurred
      MessageBox(NULL, "An error occured", "Error", MB_OK);
```

See also: UDS_WaitForService on page 273.

Class-method Version: SvcRequestTransferExit.

3.8 Definitions

The PCAN-UDS API defines the following values:

Name	Description
PCAN-UDS HandleDefinitions	Defines the handles for the different PCAN channels
Parameter Value Definitions	Defines the possible values for setting and getting PCAN's environment information with the functions UDS_SetValue and UDS_GetValue
TPUDSMsg Member Value Definitions	Defines values and constants for the members of the TPUDSMsg structure
PCAN-UDS Service Parameter Definitions	Defines constants to be used with some UDS service functions

3.8.1 PCAN-UDS Handle Definitions

Defines the handles for the different PCAN buses (Channels) within a class. This values are used as parameter where a TPUDSCANHandle on page 22is needed.

Default/Undefined handle:

	Type	Constant	Value	Description
	TPUDSCANHandle	PUDS_NONEBUS	0x0	Undefined/default value for a PCAN-UDS Channel

Handles for the **ISA** Bus (**Not Plug & Play**):

	Type	Constant	Value	Description
	TPUDSCANHandle	PUDS_ISABUS1	0x21	PCAN-ISA interface, channel 1
	TPUDSCANHandle	PUDS_ISABUS2	0x22	PCAN-ISA interface, channel 2
	TPUDSCANHandle	PUDS_ISABUS3	0x23	PCAN-ISA interface, channel 3
	TPUDSCANHandle	PUDS_ISABUS4	0x24	PCAN-ISA interface, channel 4
	TPUDSCANHandle	PUDS_ISABUS5	0x25	PCAN-ISA interface, channel 5
	TPUDSCANHandle	PUDS_ISABUS6	0x26	PCAN-ISA interface, channel 6
	TPUDSCANHandle	PUDS_ISABUS7	0x27	PCAN-ISA interface, channel 7
	TPUDSCANHandle	PUDS_ISABUS8	0x28	PCAN-ISA interface, channel 8

Handles for the **Dongle** Bus (**Not Plug & Play**)

	Type	Constant	Value	Description
	TPUDSCANHandle	PUDS_DNGBUS1	0x31	PCAN-Dongle/LPT interface, channel 1

Handles for the **PCI** Bus:

	Type	Constant	Value	Description
	TPUDSCANHandle	PUDS_PCIBUS1	0x41	PCAN-PCI interface, channel 1
	TPUDSCANHandle	PUDS_PCIBUS2	0x42	PCAN-PCI interface, channel 2
	TPUDSCANHandle	PUDS_PCIBUS3	0x43	PCAN-PCI interface, channel 3
	TPUDSCANHandle	PUDS_PCIBUS4	0x44	PCAN-PCI interface, channel 4
	TPUDSCANHandle	PUDS_PCIBUS5	0x45	PCAN-PCI interface, channel 5
	TPUDSCANHandle	PUDS_PCIBUS6	0x46	PCAN-PCI interface, channel 6
	TPUDSCANHandle	PUDS_PCIBUS7	0x47	PCAN-PCI interface, channel 7
	TPUDSCANHandle	PUDS_PCIBUS8	0x48	PCAN-PCI interface, channel 8

Handles for the **USB** Bus:

	Type	Constant	Value	Description
	TPUDSCANHandle	PUDS_USBBUS1	0x51	PCAN-USB interface, channel 1
	TPUDSCANHandle	PUDS_USBBUS2	0x52	PCAN-USB interface, channel 2
	TPUDSCANHandle	PUDS_USBBUS3	0x53	PCAN-USB interface, channel 3
	TPUDSCANHandle	PUDS_USBBUS4	0x54	PCAN-USB interface, channel 4
	TPUDSCANHandle	PUDS_USBBUS5	0x55	PCAN-USB interface, channel 5
	TPUDSCANHandle	PUDS_USBBUS6	0x56	PCAN-USB interface, channel 6
	TPUDSCANHandle	PUDS_USBBUS7	0x57	PCAN-USB interface, channel 7
	TPUDSCANHandle	PUDS_USBBUS8	0x58	PCAN-USB interface, channel 8

Handles for the **PC Card** Bus:

	Type	Constant	Value	Description
	TPUDSCANHandle	PUDS_PCCBUS1	0x61	PCAN-PC Card interface, channel 1
	TPUDSCANHandle	PUDS_PCCBUS2	0x62	PCAN-PC Card interface, channel 2

Note: These definitions are constants values in an object oriented environment (Delphi, .NET Framework) and declared as defines in C++ and Pascal (plain API).

Hardware Type and Channels

Not Plug & Play: The hardware channels of this kind are used as registered. This mean, for example, it is allowed to register the PUDS_ISABUS3 without having registered PUDS_ISA1 and PUDS_ISA2. It is a decision of each user, how to associate a PCAN-Channel (logical part) and a port/interrupt pair (physical part).

Plug & Play: For hardware handles of PCI, USB and PC-Card, the availability of the channels is determined by the count of hardware connected to a computer in a given moment, in conjunction with their internal handle. This means that having four PCAN-USB connected to a computer will let the user to connect the channels PUDS_USBBUS1 to PUDS_USBBUS4. The association of each channel with a hardware is managed internally using the handle of a hardware.

See also: Parameter Value Definitions on page 332.

3.8.2 Parameter Value Definitions

Defines the possible values for setting and getting PCAN-UDS environment information with the functions PUDS_SetValue and PUDS_GetValue.

Debug-Configuration values:

	Type	Constant	Value	Description
	Int32	PUDS_DEBUG_NONE	0	No CAN debug messages are being generated
	Int32	PUDS_DEBUG_CAN	1	CAN debug messages are written to the stdout output

Channel-Available values:

	Type	Constant	Value	Description
	Int32	PUDS_CHANNEL_UNAVAILABLE	0	The UDS PCAN-Channel handle is illegal, or its associated hardware is not available
	Int32	PUDS_CHANNEL_AVAILABLE	1	The UDS PCAN-Channel handle is valid to connect/initialize. Furthermore, for plug&play hardware, this means that the hardware is plugged-in
	Int32	PUDS_CHANNEL_OCCUPIED	2	The UDS PCAN-Channel handle is valid, and is currently being used

Server address and filter parameter values:

	Type	Constant	Value	Description
	Int32	PUDS_SERVER_ADDR_TEST_EQUIPMENT	0xF1	The standard physical address for external equipment
	Int32	PUDS_SERVER_ADDR_REQUEST_OBD_SYSTEM	0x33	The standard functional request address with OBD system
	Int32	PUDS_SERVER_ADDR_FLAG_ENHANCED_ISO_15765_3	0x1000	A flag defining that the associated address is an ISO 14229-1:2006 enhanced address
	Int32	PUDS_SERVER_ADDR_MASK_ENHANCED_ISO_15765_3	0x07FF	The mask used to check ISO 14229-1:2006 enhanced addresses

Server filter parameter values:

	Type	Constant	Value	Description
	Int32	PUDS_SERVER_FILTER_IGNORE	0x0000	Flag to remove an address from the server filter list
	Int32	PUDS_SERVER_FILTER_LISTEN	0x8000	Flag to add an address to the server filter list, allowing messages with this address to be fetched from the UDS receive queue

Timeout parameter values:

	Type	Constant	Value	Description
	Int32	PUDS_TIMEOUT_REQUEST	10000	Default maximum timeout in milliseconds for UDS transmit confirmation
	Int32	PUDS_TIMEOUT_RESPONSE	10000	Default maximum timeout in milliseconds for UDS response reception (excluding server timeout performance requirements)

UDS Session Information values:

	Type	Constant	Value	Description
	Int32	PUDS_P2CAN_DEFAULT_SERVER_MAX	10000	Default value in milliseconds for the server performance requirement
	Int32	PUDS_P2CAN_ENHANCED_SERVER_MAX	10000	Default value in milliseconds for the enhanced server performance requirement

ISO-TP data padding values:

	Type	Constant	Value	Description
	Int32	PUDS_CAN_DATA_PADDING_NONE	0x00	CAN frame data optimization is enabled
	Int32	PUDS_CAN_DATA_PADDING_ON	0x01	CAN frame data optimization is disabled: CAN data length is always 8 and data is padded with zeros

Remarks: These definitions are constants values in an object oriented environment (Delphi, .NET Framework) and declared as defines in C++ (plain API).

See also: TPUDSParameter on page 31, PCAN-UDS Handle Definitions on page 330.

3.8.3 TPUDSMsg Member Value Definitions

The following definitions apply to members of the TPUDSMsg structure.

DATA information:

	Type	Constant	Value	Description
	Int32	PUDS_MAX_DATA	4095	Maximum data length in bytes of UDS message

Negative Response Code (NRC) values:

	Type	Constant	Value	Description
	Int32	PUDS_NRC_EXTENDED_TIMING	0x78 (120)	Negative UDS response code stating that the server/ECU requests more time to transmit a response
	Int32	PUDS_MAX_DATA	4095	Maximum data length in bytes of UDS message
	Int32	PUDS_SI_POSITIVE_RESPONSE	0x40 (64)	Service Identifier offset for positive response message (i.e.: Service Identifier for a positive response to a UDS request is 0x40 + Service Request Identifier)

Service Identifier (SI) values:

	Type	Constant	Value	Description
	Int32	PUDS_SI_POSITIVE_RESPONSE	0x40 (64)	Service Identifier offset for positive response message (i.e.: Service Identifier for a positive response to a UDS request is 0x40 + Service Request Identifier)

NO_POSITIVE_RESPONSE_MSG values:

	Type	Constant	Value	Description
	Int32	PUDS_SUPPR_POS_RSP_MSG_INDICATION_BIT	0x80	Value for NO_POSITIVE_RESPONSE_MSG stating to discard any positive response
	Int32	PUDS_KEEP_POS_RSP_MSG_INDICATION_BIT	0	Default value for NO_POSITIVE_RESPONSE_MSG stating to return any responses

See also: TPUDSMsg on page 14.

3.8.4 PCAN-UDS Service Parameter Definitions

SecurityAccess

The following constants are a reminder of some Request Seed and Send Key values. Note: ranges for system-supplier-specific use and ISO/SAE reserved values have been discarded.

	Type	Constant	Value	Description
	BYTE	PUDS_SVC_PARAM_SA_RSD_1	1	Request seed (odd numbers)
	BYTE	PUDS_SVC_PARAM_SA_RSD_3	3	Request seed (odd numbers)
	BYTE	PUDS_SVC_PARAM_SA_RSD_5	5	Request seed (odd numbers)
	BYTE	PUDS_SVC_PARAM_SA_RSD_MIN	7	Request seed (odd numbers)
	BYTE	PUDS_SVC_PARAM_SA_RSD_MAX	95 (0x5F)	Request seed (odd numbers)
	BYTE	PUDS_SVC_PARAM_SA_SK_2	2	Send Key (even numbers)
	BYTE	PUDS_SVC_PARAM_SA_SK_4	4	Request seed (even numbers)
	BYTE	PUDS_SVC_PARAM_SA_SK_6	6	Request seed (even numbers)
	BYTE	PUDS_SVC_PARAM_SA_SK_MIN	8	Request seed (even numbers)
	BYTE	PUDS_SVC_PARAM_SA_SK_MAX	96 (0x60)	Request seed (even numbers)

CommunicationControl

The communicationType parameter is a bit-code value which allows control of multiple communication types at the same time.

The following table lists the coding of the communicationType data parameter:

- the bit-encoded low nibble of this byte represents the comminucationTypes,
- the high nibble defines which of the subnets connected to the receiving node shall be disabled/enabled

	Type	Constant	Value	Description
	BYTE	PUDS_SVC_PARAM_CC_FLAG_APPL	1	CommunicationType Flag: Application (01b)
	BYTE	PUDS_SVC_PARAM_CC_FLAG_NWM	2	CommunicationType Flag: NetworkManagement (10b)
	BYTE	PUDS_SVC_PARAM_CC_FLAG_DESCTIRNCN	0	CommunicationType Flag: Disable/Enable specified communicationType (see Flags APPL/NMW)
	BYTE	PUDS_SVC_PARAM_CC_FLAG_DENWRIRO	240 (0xF0)	CommunicationType Flag: Disable/Enable network which request is received on
	BYTE	PUDS_SVC_PARAM_CC_FLAG_DESNIBNN_MIN	16 (0x10)	CommunicationType Flag: Disable/Enable specific network identified by network number (minimum value)
	BYTE	PUDS_SVC_PARAM_CC_FLAG_DESNIBNN_MAX	224 (0xE0)	CommunicationType Flag: Disable/Enable specific network identified by network number (maximum value)
	BYTE	PUDS_SVC_PARAM_CC_FLAG_DESNIBNN_MASK	240 (0xF0)	CommunicationType Flag: Mask for DESNIBNN bits

ResponseOnEvent

The following table defines the expected size of the EventTypeRecord based on the EventType (TPUDSParamROE):

	Type	Constant	Value	Description
	BYTE	PUDS_SVC_PARAM_ROE_STPROE_LEN	0	expected size of EventTypeRecord for ROE_STPROE
	BYTE	PUDS_SVC_PARAM_ROE_ONDTCS_LEN	1	expected size of EventTypeRecord for ROE_ONDTCS
	BYTE	PUDS_SVC_PARAM_ROE_OTI_LEN	1	expected size of EventTypeRecord for ROE_OTI
	BYTE	PUDS_SVC_PARAM_ROE_OCODID_LEN	2	expected size of EventTypeRecord for ROE_OCODID
	BYTE	PUDS_SVC_PARAM_ROE_RAE_LEN	0	expected size of EventTypeRecord for ROE_RAE

	Type	Constant	Value	Description
	BYTE	PUDS_SVC_PARAM_ROE_STRTROE_LEN	0	expected size of EventTypeRecord for ROE_STRTROE
	BYTE	PUDS_SVC_PARAM_ROE_CLRROE_LEN	0	expected size of EventTypeRecord for ROE_CLRROE
	BYTE	PUDS_SVC_PARAM_ROE_OCOV_LEN	10	expected size of EventTypeRecord for ROE_OCOV

The following table lists extra constants to be used with ResponseOnEvent service:

	Type	Constant	Value	Description
	BYTE	PUDS_SVC_PARAM_ROE_EWT_ITTR	2	Infinite Time To Response (eventWindowTime parameter)

ClearDiagnosticInformation

The following table lists constants to be used as the GroupOfDTC parameter with the ClearDiagnosticInformation service:

	Type	Constant	Value	Description
	UInt32	PUDS_SVC_PARAM_CDI_ERS	0x000000	groupOfDTC : Emissions-related systems group of DTCs
	UInt32	PUDS_SVC_PARAM_CDI_AGDTC	0xFFFFFF	groupOfDTC : All Groups of DTCs

4 Additional Information

PCAN is the platform for PCAN-OBDII, PCAN-UDS and PCAN-Basic. In the following topics there is an overview of PCAN and the fundamental practice with the interface DLL CanApi2 (PCAN-API).

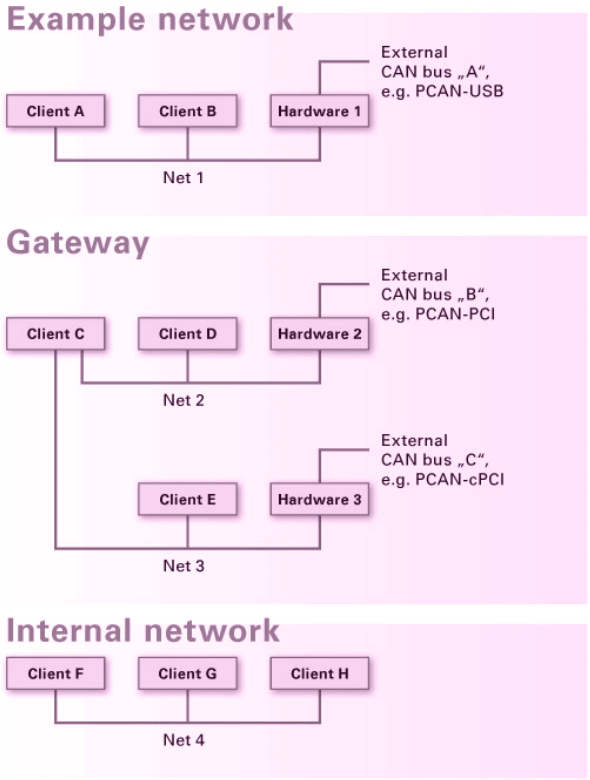
Topics	Description
PCAN Fundamentals	This section contains an introduction to PCAN
PCAN-Basic	This section contains general information about the PCAN-Basic API
UDS and ISO-TP Network Addressing Information	This section contains general information about the ISO-TP network addressing format

4.1 PCAN Fundamentals

PCAN is a synonym for PEAK CAN APPLICATIONS and is a flexible system for planning, developing, and using a CAN Bus System. Developers as well as end users are getting a helpful and powerful product.

Basis for the communication between PCs and external hardware via CAN is a series of Windows Kernel Mode Drivers (Virtual Device Drivers) e.g. PCAN_USB.SYS, PCAN_PCI.SYS, PCAN_xxx.SYS. These drivers are the core of a complete CAN environment on a PC running Windows and work as interfaces between CAN software and PC-based CAN hardware. The drivers manage the entire data flow of every CAN device connected to the PC.

A user or administrator of a CAN installation gets access via the PCAN-Clients (short: Clients). Several parameters of processes can be visualized and changed with their help. The drivers allow the connection of several Clients at the same time.



Furthermore, several hardware components based on the SJA1000 CAN controller are supported by a PCAN driver. So-called Nets provide the logical structure for CAN busses, which are virtually extended into the PC. On the hardware side, several Clients can be connected, too. The following figures demonstrate different possibilities of Net configurations (also realizable at the same time).

Following rules apply to PCAN clients, nets and hardware:

- One Client can be connected to several Nets
- One Net provides several Clients
- One piece of hardware belongs to one Net
- One Net can include none or one piece of hardware
- A message from a transmitting Client is carried on to every other connected Client, and to the external bus via the connected CAN hardware
- A message received by the CAN hardware is received by every connected Client. However, Clients react only on those messages that pass their acceptance filter

Users of PCAN-View 3 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.

See also: PCAN-Basic below, ISO-TP Network Addressing Format on page 342.

4.2 PCAN-Basic

PCAN-Basic is an Application Programming Interface for the use of a collection of Windows Device Drivers from PEAK-System, which allow the real-time connection of Windows applications to all CAN busses physically connected to a PC.

PCAN-Basic principal characteristics are:

- Information about the receive time of a CAN message
- Easy switching between different PCAN-Channels (PCAN-PC hardware)
- The possibility to control some parameters in the hardware, eg. "Listen-Only" mode, automatic reset of the CAN controller, etc
- The use of event notifications, for faster processing of incomming CAN messages
- An improved system for debugging operations
- The use of only one Dynamic Link Library (PCANBasic.DLL) for all supported hardware
- The possibility to connect more than 2 channels per PCAN-Device. The following list shows the PCAN-Channels that can be connected per PCAN-Device:

	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

The PCAN-basic offers the possibility to use several PCAN-Channels within the same application in an easy way. The communication process is divided in 3 phases: initialization, interaction and finalization of a PCAN-Channel.

Initialization: In order to do CAN communication using a channel, it is necessary to first initialize it. This is done making a call to the function CAN_Initialize (**class-method**: Initialize) or CAN_InitializeFD (**class-method**: InitializeFD) in case FD communication is desired.

Interaction: After a successful initialization, a channel is ready to communicate with the connected CAN bus. Further configuration is not needed. The functions CAN_Read and CAN_Write (**class-methods**: Read and Write) can be then used to read and write CAN messages. If the channel being used is FD capable and it was initialized using CAN_InitializedFD, then the functions to use are CAN_ReadFD and CAN_WriteFD (**class-methods**: ReadFD and WriteFD). If desired, extra configuration can be made to improve a 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 PCAN-Channel and the resources allocated for it. In this way the channel is marked as "Free" and can be used from 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-cPCI	Yes	Pcan_pci.sys
PCAN-miniPCI	Yes	Pcan_pci.sys
PCAN-PC/104-Plus	Yes	Pcan_pci.sys
PCAN-USB	Yes	Pcan_usb.sys
PCAN-USB Pro	Yes	Pcan_usb.sys
PCAN-USB Pro FD	Yes	Pcan_usb.sys
PCAN-PC Card	Yes	Pcan_pcc.sys
PCAN-Ethernet Gateway DR	Yes	Pcan_lan.sys
PCAN-Wireless DR	Yes	Pcan_lan.sys
PCAN-Wireless Gateway	Yes	Pcan_lan.sys
PCAN-Wireless Automotive Gateway	Yes	Pcan_lan.sys

See also: PCAN Fundamentals on page 336, ISO-TP Network Addressing Format on page 342.

4.3 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-UDS 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`):
  - CAN ID `0x7DF` (`PUDS_ISO_15765_4_CAN_ID_FUNCTIONAL_REQUEST`) from Source `0xF1` to Target `0x33`
- 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):
  - ECU #1
    1. Request: CAN ID `0x7E8` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_1`) from Source `0xF1` to Target `0x01`
    2. Response: CAN ID `0x7E0` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_1`) from Source `0x01` to Target `0xF1`
  - ECU #2:
    1. Request: CAN ID `0x7E9` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_2`) from Source `0xF1` to Target `0x01`
    2. Response: CAN ID `0x7E1` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_2`) from Source `0x01` to Target `0xF1`
  - ECU #3:
    1. Request: CAN ID `0x7EA` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_3`) from Source `0xF1` to Target `0x01`
    2. Response: CAN ID `0x7E2` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_3`) from Source `0x01` to Target `0xF1`
  - ECU #4:
    1. Request: CAN ID `0x7EB` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_4`) from Source `0xF1` to Target `0x01`
    2. Response: CAN ID `0x7E3` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_4`) from Source `0x01` to Target `0xF1`
  - ECU #5:
    1. Request: CAN ID `0x7EC` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_5`) from Source `0xF1` to Target `0x01`
    2. Response: CAN ID `0x7E4` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_5`) from Source `0x01` to Target `0xF1`
  - ECU #6:
    1. Request: CAN ID `0x7ED` (`PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_6`) from Source `0xF1` to Target `0x01`

2. Response: CAN ID 0x7E5 (PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_6) from Source 0x01 to Target 0xF1

- ECU #7:
  1. Request: CAN ID 0x7EE (PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_7) from Source 0xF1 to Target 0x01
  2. Response: CAN ID 0x7E6 (PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_7) from Source 0x01 to Target 0xF1
- ECU #8:
  1. Request: CAN ID 0x7EF (PUDS_ISO_15765_4_CAN_ID_PHYSICAL_REQUEST_8) from Source 0xF1 to Target 0x01
  2. Response: CAN ID 0x7E7 (PUDS_ISO_15765_4_CAN_ID_PHYSICAL_RESPONSE_8) from Source 0x01 to Target 0xF1

- Communications with 29 bits CAN identifier and FIXED NORMAL addressing format (where XX is Target Address and YY Source address and respectively physical/functional addressing):
  1. CAN ID 0x00DAXXYY/0x00DBXXYY (data link layer priority 0)
  2. CAN ID 0x04DAXXYY/0x04DBXXYY (data link layer priority 1)
  3. CAN ID 0x08DAXXYY/0x08DBXXYY (data link layer priority 2)
  4. CAN ID 0x0CDAXXYY/0x0CDBXXYY (data link layer priority 3)
  5. CAN ID 0x10DAXXYY/0x10DBXXYY (data link layer priority 4)
  6. CAN ID 0x14DAXXYY/0x14DBXXYY (data link layer priority 5)
  7. CAN ID 0x18DAXXYY/0x18DBXXYY (data link layer priority 6)
  8. CAN ID 0x1CDAXXYY/0x1CDBXXYY (data link layer priority 7)
- Communications with 29 bits CAN identifier and MIXED addressing format (where XX is Target Address and YY Source address and respectively physical/functional addressing):
  1. CAN ID 0x00CEXXYY/0x00CDXXYY (data link layer priority 0)
  2. CAN ID 0x04CEXXYY/0x04CDXXYY (data link layer priority 1)
  3. CAN ID 0x08CEXXYY/0x08CDXXYY (data link layer priority 2)
  4. CAN ID 0x0CCEXXYY/0x0CCDXXYY (data link layer priority 3)
  5. CAN ID 0x10CEXXYY/0x10CDXXYY (data link layer priority 4)
  6. CAN ID 0x14CEXXYY/0x14CDXXYY (data link layer priority 5)
  7. CAN ID 0x18CEXXYY/0x18CDXXYY (data link layer priority 6)
  8. CAN ID 0x1CCEXXYY/0x1CCDXXYY (data link layer priority 7)

- Communications with 29 bits CAN identifier and ENHANCED addressing format (where YYY is Target Address and XXX Source address, addresses are encoded on 11 bits):

    1. CAN ID 0x03XXXYYY (data link layer priority 0)
    2. CAN ID 0x07XXXYYY (data link layer priority 1)
    3. CAN ID 0x0BXXXYYY (data link layer priority 2)
    4. CAN ID 0x0FXXXYYY (data link layer priority 3)
    5. CAN ID 0x13XXXYYY (data link layer priority 4)
    6. CAN ID 0x17XXXYYY (data link layer priority 5)
    7. CAN ID 0x1BXXXYYY (data link layer priority 6)
    8. CAN ID 0x1FXXXYYY (data link layer priority 7)

If an application requires other communication settings, it will have to be set with through the parameters `PUDS_PARAM_MAPPING_ADD` and `PUDS_PARAM_MAPPING_REMOVE`. For a complete example, see §4.3.3 PCAN-UDS Example C:\Users\Christoph\Documents\Doku 2019\PCAN-UDS-API\Phase 2\PCAN-UDS-API_UserMan_eng.doc - _PCAN-UDS_example#_PCAN-UDS_exampleon page 343.

Alternatively, it is also possible to directly use PCAN-ISO-TP API: although PCAN-UDS and PCAN-ISO-TP define differents types for CAN channels (respectively `TPUDSCANHandle` and `TPCANTPHandle`), they are both the same type. Once a PCAN-UDS channel is initialized, PCAN-ISO-TP specific functions (like `CANTP_AddMapping`) can be called with this PCAN-UDS channel.

4.3.1 Usage in a Non-Standardized Context

Default source address

When a UDS channel is initialized, the default source address for this new node is the standardized “Test Equipment” address: 0xF1. This means that all UDS messages received by this node whose target address does not match this source address will be discarded. If your application makes communications with a different source address, you need to specify that address to the API by using the parameter `PUDS_PARAM_SERVER_ADDRESS`:

```
TPUDSCANHandle Channel;
...
// Define server address
BYTE param = 0xA1;
TPUDSStatus Status = UDS_SetValue(Channel, PUDS_PARAM_SERVER_ADDRESS, &param,
sizeof(param));
// check status and proceed…
```

Alternatively, it is possible to listen to multiple addresses via the parameter `PUDS_PARAM_SERVER_Filter`:

```
TPUDSCANHandle Channel;
...
// listen to address 0x11
WORD param = (PUDS_SERVER_FILTER_LISTEN | 0x11);
```

```
TPUDSStatus Status = UDS_SetValue(Channel, PUDS_PARAM_SERVER_Filter, &param,
sizeof(param));
// check status and proceed…
```

Removing Default Mappings

- If you need to set different ISO-TP Network Addressing Information to the already defined mappings (for instance to use the standardized CAN IDs with a different Source Address), you first need to remove the existing mapping(s) by calling the UDS_SetValue with the parameter PUDS_PARAM_MAPPING_REMOVE.
- Standardized 29 bits CAN IDs do not use mappings, if you want to override those CAN IDs, simply use that value when adding a maping.

```
TPUDSCANHandle Channel;
TPUDSMsg Message;
TPUDSStatus Status;

// Note: Channel is propoerly initialized with UDS_Initialize(..)
// [...]

// Remove mapping matching ID 0x7DF for functional request from External_Equipment
//  - any protocol value except 0 will try to remove matching mappings.
Message.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;
Message.LEN = 4;
// data[0..3] holds CAN ID
Message.DATA.RAW[0] = 0x00;
Message.DATA.RAW[1] = 0x00;
Message.DATA.RAW[2] = 0x07;
Message.DATA.RAW[3] = 0xDF;
status = UDS_SetValue(Channel, PUDS_PARAM_MAPPING_REMOVE, &Message, sizeof(Message));
```

4.3.2 ISO-TP Network Addressing Format

ISO-TP specifies three addressing formats to exchange data: normal, extended and mixed addressing. Each addressing requires a different number of CAN frame data bytes to encapsulate the addressing information associated with the data to be exchanged.

The following table sums up the mandatory configuration to the ISO-TP API for each addressing format:

Addressing format	CAN ID length	Mandatory configuration steps
Normal addressing PCANTP_FORMAT_NORMAL	11 bits	Define mappings with CANTP_AddMapping
	29 bits	Define mappings with CANTP_AddMapping
Normal fixed addressing PCANTP_FORMAT_FIXED_NORMAL	11 bits	Addressing is invalid
	29 bits	-
Extended addressing PCANTP_FORMAT_EXTENDED	11 bits	Define mappings with CANTP_AddMapping
	29 bits	Define mappings with CANTP_AddMapping
Mixed addressing PCANTP_FORMAT_MIXED	11 bits	Define mappings with CANTP_AddMapping
	29 bits	-
Enhanced addressing PCANTP_FORMAT_ENHANCED	11 bits	Addressing is invalid
	29 bits	-

A mapping allows an ISO-TP node to identify and decode CAN Identifiers, it binds a CAN ID to an ISO-TP network address information. CAN messages that cannot be identified are ignored by the API.

Mappings involving physically addressed communication are most usually defined in pairs: the first mapping defines outgoing communication (i.e. request messages from node A to node B) and the second to match incoming communication (i.e. responses from node B to node A).

Functionally addressed communication requires one mapping to transmit functionally addressed messages (i.e. request messages from node A to any node) and as many mappings as responding nodes (i.e. responses from nodes B, C, etc. to node A).

4.3.3 PCAN-UDS Example

Configuration of Mappings

The following C++ example shows how to define 4 mappings with the PCAN-UDS API on the Tester Client side in order to:

1. Transmit physical message to ECU #1 with CAN ID 0x326
2. Receive physical message from ECU #1 with CAN ID 0x626
3. Transmit functional message on CAN ID 0x200
4. Receive UUDT message from the ECU #1 with CAN ID 0x526

```
TPUDSCANHandle Channel;
TPUDSMsg Message;
TPUDSStatus Status;

// Note: Channel is properly initialized with UDS_Initialize(..)
// [...]

// Add mapping for physical request from External_Equipment to ECU_#X : ID=0x326
Message.NETADDRINFO.SA = 0xF1;
Message.NETADDRINFO.TA = 0x01;
Message.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
Message.NETADDRINFO.RA = 0x00;
Message.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;
Message.LEN = 8;
// data[0..3] holds CAN ID
Message.DATA.RAW[0] = 0x00;
Message.DATA.RAW[1] = 0x00;
Message.DATA.RAW[2] = 0x03;
Message.DATA.RAW[3] = 0x26;
// data[4..7] holds CAN ID Response (for Flow Control) (-1 if none)
Message.DATA.RAW[4] = 0x00;
Message.DATA.RAW[5] = 0x00;
Message.DATA.RAW[6] = 0x06;
Message.DATA.RAW[7] = 0x26;
status = UDS_SetValue(Channel, PUDS_PARAM_MAPPING_ADD, &Message, sizeof(Message));

// Add mapping for physical response from ECU_#X to External_Equipment : ID=0x626
Message.NETADDRINFO.SA = 0x01;
Message.NETADDRINFO.TA = 0xF1;
Message.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
Message.NETADDRINFO.RA = 0x00;
Message.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;
Message.LEN = 8;
// data[0..3] holds CAN ID
Message.DATA.RAW[0] = 0x00;
Message.DATA.RAW[1] = 0x00;
Message.DATA.RAW[2] = 0x06;
Message.DATA.RAW[3] = 0x26;
// data[4..7] holds CAN ID Response (for Flow Control) (-1 if none)
```

```
        Message.DATA.RAW[4] = 0x00;
        Message.DATA.RAW[5] = 0x00;
        Message.DATA.RAW[6] = 0x03;
        Message.DATA.RAW[7] = 0x26;
        status = UDS_SetValue(Channel, PUDS_PARAM_MAPPING_ADD, &Message, sizeof(Message));

        // Add mapping for functional request from External_Equipment : ID=0x200
        Message.NETADDRINFO.SA = 0xF1;
        Message.NETADDRINFO.TA = 0x33;
        Message.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_FUNCTIONAL;
        Message.NETADDRINFO.RA = 0x00;
        Message.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_ISO_15765_2_11B;
        Message.LEN = 8;
        // data[0..3] holds CAN ID
        Message.DATA.RAW[0] = 0x00;
        Message.DATA.RAW[1] = 0x00;
        Message.DATA.RAW[2] = 0x02;
        Message.DATA.RAW[3] = 0x00;
        // data[4..7] holds CAN ID Response (for Flow Control) (-1 if none)
        Message.DATA.RAW[4] = 0xFF;
        Message.DATA.RAW[5] = 0xFF;
        Message.DATA.RAW[6] = 0xFF;
        Message.DATA.RAW[7] = 0xFF;
        status = UDS_SetValue(Channel, PUDS_PARAM_MAPPING_ADD, &Message, sizeof(Message));

        ////// Unacknowledge Unsegmented Data Transfert (UUDT) support:
        ////// standard CAN ID without UDS Protocol Data Unit can be sent by ECU with service
readDataByPeriodicdataIdentifier
        // Add mapping for UUDT physical response from ECU_#X to External_Equipment : ID=0x526
        Message.NETADDRINFO.SA = 0x01;
        Message.NETADDRINFO.TA = 0xF1;
        Message.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
        Message.NETADDRINFO.RA = 0x00;
        Message.NETADDRINFO.PROTOCOL = 0;
        Message.LEN = 4;
        // data[0..3] holds CAN ID
        Message.DATA.RAW[0] = 0x00;
        Message.DATA.RAW[1] = 0x00;
        Message.DATA.RAW[2] = 0x05;
        Message.DATA.RAW[3] = 0x26;
        status = UDS_SetValue(Channel, PUDS_PARAM_MAPPING_ADD, &Message, sizeof(Message));
```

UUDT Read/Write example

The following C++ example shows Unacknowledge Unsegmented Data Transfer, it writes from a UDS channel and reads the message from another UDS channel.

```
void sample_rw_uudt(TPUDSCANHandle channelTx, TPUDSCANHandle channelRx) {
        TPUDSStatus status;
        int iBuffer;
        TPUDSMsg Message;
        int count;

        // Initializes UDS Communication for the transmitting channel
        status = UDS_Initialize(channelTx, PUDS_BAUD_500K, 0, 0, 0);
        printf("Initialize UDS: %i\n", (int)status);
        // Initializes UDS Communication for the receiving channel
        status = UDS_Initialize(channelRx, PUDS_BAUD_500K, 0, 0, 0);
        printf("Initialize ChannelRx: %i\n", (int)status);

        // Define "channelTx" Address as ECU #9
        iBuffer = 0xF9;
        status = UDS_SetValue(channelTx, PUDS_PARAM_SERVER_ADDRESS, &iBuffer, 1);
        printf("  Set ServerAddress: %i (0x%02x)\n", (int)status, iBuffer);
        // Define "channelRx" Address as External equipement
```

```
        iBuffer = 0xF1;
        status = UDS_SetValue(channelRx, PUDS_PARAM_SERVER_ADDRESS, &iBuffer, 1);
        printf("  Set ServerAddress: %i (0x%02x)\n", (int)status, iBuffer);

        // Prepare mapping configuration:
        //  UUDT physical response from ECU_#9 to External_Equipment : ID=0x526
        Message.NETADDRINFO.SA = 0xF9;
        Message.NETADDRINFO.TA = 0xF1;
        Message.NETADDRINFO.TA_TYPE = PUDS_ADDRESSING_PHYSICAL;
        Message.NETADDRINFO.RA = 0x00;
        Message.NETADDRINFO.PROTOCOL = PUDS_PROTOCOL_NONE;
        Message.LEN = 4;
        // data[0..3] holds CAN ID
        Message.DATA.RAW[0] = 0x00;
        Message.DATA.RAW[1] = 0x00;
        Message.DATA.RAW[2] = 0x05;
        Message.DATA.RAW[3] = 0x26;

        // Add "channelTx" mapping (in order to send message)
        //  for UUDT physical response from ECU_#9 to External_Equipment : ID=0x526
        status = UDS_SetValue(channelTx, PUDS_PARAM_MAPPING_ADD, &Message, sizeof(Message));
        printf("  Add UUDT mapping: 0x%04x\n", status);
        // Add "channelRx" mapping (in order to receive message)
        //  for UUDT physical response from ECU_#9 to External_Equipment : ID=0x526
        status = UDS_SetValue(channelRx, PUDS_PARAM_MAPPING_ADD, &Message, sizeof(Message));
        printf("  Add ChannelRx UUDT mapping: 0x%04x\n", status);

        // Write a message from "channelTx" to "channelRx"
        Message.LEN = 6;
        Message.DATA.RAW[4] = 0xCA;
        Message.DATA.RAW[5] = 0xB1;
        status = UDS_Write(channelTx, &Message);
        printf("  UDS_Write UUDT: 0x%04x\n", status);

        // Read message on channel Rx
        printf("  Reading message on ChannelRx...\n");
        memset(&Message, 0, sizeof(Message));
        count = 0;
        do {
                count++;
                Sleep(100);
                status = UDS_Read(channelRx, &Message);
        } while (status == PUDS_ERROR_NO_MESSAGE && count < 10);
        if (status == PUDS_ERROR_NO_MESSAGE)
                printf("Failed to read message on Channel RX !");
        else {
                // Received message will hold Network Address Information
                //  as defined by the mapping.
                // The CAN ID information is removed from the DATA.RAW field.
                displayMessage(NULL, &Message);
        }

        UDS_Uninitialize(channelTx);

}
```

4.4 Using Events

Event objects can be used to automatically notify a client on reception of a UDS message. This has following advantages:

- The client program doesn't need to check periodically for received messages any longer
- The response time on received messages is reduced

To use events, the client application must call the UDS_SetValue function (class-method: SetValue) to set the parameter PUDS_PARAM_RECEIVE_EVENT. This parameter sets the handle for the event object. When receiving a message, the API sets this event to the "Signaled" state.

Another thread must be started in the client application, which waits for the event to be signaled, using one of the Win32 synchronization functions (e.g. WaitForSingleObject) without increasing the processor load. After the event is signaled, available messages can be read with the UDS_Read function (class method: Read), and the UDS messages can be processed.

Remarks

Be careful, it is not recommended to use both event-handler (with a reading thread) and the UDS „WaitFor“ functions:

- WaitForSingleMessage,
- WaitForMultipleMessage,
- WaitForService,
- WaitForServiceFunctional.

Indeed both mechanisms would read messages at the same time, the result is that one will not receive any (or some) messages. If one of the previous functions is called and a thread is waiting for events to call the read UDS message function, then the user will have to temporarily prevent the thread from reading messages.

Tips for the creation of the event object:

- Creation of the event as "auto-reset"
  - Trigger mode "set" (default): After the first waiting thread has been released, the event object's state changes to non-signaled. Other waiting threads are not released. If no threads are waiting, the event object's state remains signaled
  - Trigger mode "pulse": After the first waiting thread has been released, the event object's state changes to non-signaled. Other waiting threads are not released. If no threads are waiting, or if no thread can be released immediately, the event object's state is simply set to non-signaled
- Creation of the event as "manual-reset"
  - Trigger mode "set" (default): The state of the event object remains signaled until it is set explicitly to the non-signaled state by the Win32 ResetEvent function. Any number of waiting threads, or threads that subsequently begin wait operations, can be released while the object's state remains signaled
  - Trigger mode "pulse": All waiting threads that can be released immediately are released. The event object's state is then reset to the non-signaled state. If no threads are waiting, or if no thread can be released immediately, the event object's state is simply set to non-signaled

See also: UDS_SetValue (class-method: SetValue), UDS_Read (class-method: Read)

5 License Information

The APIs PCAN-UDS, PCAN-ISO-TP, and PCAN-Basic 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 CAN hardware of third-party suppliers should be compatible to that of PEAK-System, then you are not allowed to use the mentioned APIs with those components.

If a third-party supplier develops software based on the mentioned APIs and problems occur during the use of this software, consult that third-party supplier.