PCAN-ISO-TP API

API Implementation of the ISO-TP Standard
(ISO 15765-2)

Documentation

Document version 1.6.2 (2019-01-10)

PEAK System

PEAK-System Technik GmbH
Otto-Roehm-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 1.6.2 (2019-01-10)

Contents

1 PCAN-ISO-TP API

Welcome to the documentation of PCAN-ISO-TP API, a PEAK CAN API that implements ISO 15765-2, or ISO-TP, an international standard for sending data packets over a CAN bus.

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

2 Introduction

PCAN-ISO-TP, is a simple programming interface that allows the communication between Windows applications and Electronic Control Units (ECU) over a CAN bus and more specifically to transmit and receive bigger data packets than the limited 8 bytes of the CAN standard.

2.1 Understanding PCAN-ISO-TP

ISO-TP is an international standard for sending data packets over a CAN bus. The protocol defines data segmentation that allows to transmit messages which cannot be transmitted with a single CAN frame, the maximum data length that can be transmitted in a single data block is 4095 bytes.

The exchange of data between nodes (e.g. from Electronic Control Units (ECU) to ECU, or between external test equipment and an ECU) is supported by different addressing formats. Each of them requires a different number of CAN frame data bytes to encapsulate the addressing information associated with the data to be exchanged.

This protocol is fully described in the norm ISO 15765-2 and is required by UDS, Unified Diagnostic Services. This later protocol is a communication protocol of the automotive industry and is described in the norm ISO 14229-1.

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

2.2 Using PCAN-ISO-TP

Since PCAN-ISO-TP API is built on top of the PCAN-Basic API, most of its functions are similar. It offers the possibility to use several PCANTP channels within the same application in an easy way. The communication process is divided in three phases: initialization, interaction, and finalization of a PCANTP channel.

Initialization: In order to do CANTP communication (i.e. CAN communication with ISO-TP support) using a channel, it is necessary to initialize it first. This is done by making a call to the function CANTP_Initialize (**class method version**: Initialize). Depending on the message addressing format, it may be necessary to define mappings between CAN Identifier and ISO-TP network addressing information through the function CANTP_AddMapping (**class method version**: AddMapping).

Interaction: After a successful initialization, a channel is ready to communicate with the connected CAN bus. Further configuration is not needed. The functions CANTP_Read and CANTP_Write (**class method versions**: Read and Write) can be then used to read and write CAN messages that supports ISO-TP. If desired, extra configuration can be made to improve a communication session, like changing the time between transmissions of fragmented CAN messages.

Finalization: When the communication is finished, the function CANTP_Uninitialize (**class method version**: Uninitialize) should be called in order to release the PCANTP 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 License Regulations

The interface DLLs of this API, PCAN-Basic, device drivers, and further files needed for linking are property of the PEAK-System Technik GmbH and may be used only in connection with a hardware component purchased from PEAK-System or one of its partners. If a CAN hardware component of third-party suppliers should be compatible to one of PEAK-System, then you are not allowed to use or to pass on the APIs and driver software of PEAK-System.

If a third-party supplier develops software based on the PCAN-ISO-TP API and problems occur during the use of this software, consult the software provider.

2.4 Features

- Implementation of the ISO-TP protocol (ISO 15765-2) for the transfer of data packages up to 4095 bytes via the CAN bus
- Windows DLLs for the development of 32-bit and 64-bit applications
- Thread-safe API
- Physical communication via CAN using a CAN or CAN FD interface of the PCAN series
- Uses the PCAN-Basic programming interface to access the CAN or CAN FD hardware in the computer

2.5 System Requirements

- Windows 10, 8.1, 7 (32/64-bit)
- At least 2 GB RAM and 1.5 GHz CPU
- For the CAN bus connection: PC CAN or CAN FD interface from PEAK-System
- PCAN-Basic API

2.6 Scope of Supply

- Interface DLLs for Windows (32/64-bit)
- Examples and header files for all common programming languages
- Documentation in PDF format

3 DLL API Reference

This section contains information about the data types (classes, structures, types, defines, enumerations) and API functions which are contained in the PCAN-ISO-TP 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-OBD-2 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.IsoTp

The Peak.Can.IsoTp namespace contains types and classes to use the PCAN-ISO-TP 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 PCANTP 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
	TPCANTPHandle	Represents a PCAN-ISO-TP channel handle.

Classes

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

Structures

	Class	Description
	TPCANTPMsg	Defines a CAN ISO-TP message. The members of this structure are sequentially byte aligned.
	TPCANTPTimestamp	Defines a time-stamp of a CAN ISO-TP message.

Enumerations

	Name	Description
	TPCANTPStatus	Represents a PCAN-ISO-TP status/error code.
	TPCANTPBaudrate	Represents a PCAN Baud rate register value.
	TPCANTPHWType	Represents the type of PCAN hardware to be initialized.
	TPCANTPMessageType	Represents the type of a PCAN-ISO-TP message.
	TPCANTPIdType	Represents the CAN ID type of a PCAN-ISO-TP message.
	TPCANTPFormatType	Represents the type of format of a PCAN-ISO-TP message.
	TPCANTPAddressingType	Represents the type of message addressing of a PCAN-ISO-TP message.
	TPCANTPConfirmation	Represents the network status of a communicated PCAN ISO-TP message.
	TPCANTPParameter	Represents a PCAN-ISO-TP parameter to be read or set.
	TPCANTPBitrateFD	Represents a bit rate string with flexible data rate (FD).

3.2 Units

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

Namespaces

	Alias	Description
{}	PCANTP Unit	Delphi unit for using the PCAN-ISO-TP API from PEAK-System.

3.2.1 PCANTP Unit

The PCANTP Unit contains types and classes to use the PCAN-ISO-TP 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.IsoTp 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
	TPCANTPHandle	Represents a PCAN-ISO-TP channel handle.

Classes

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

Structures

	Class	Description
	TPCANTPMsg	Defines a CAN ISO-TP message. The members of this structure are sequentially byte aligned.
	TPCANTPTimestamp	Defines a time-stamp of a CAN ISO-TP message.

Enumerations

	Name	Description
	TPCANTPStatus	Represents a PCAN-ISO-TP status/error code.
	TPCANTPBaudrate	Represents a PCAN Baud rate register value.
	TPCANTPHWType	Represents the type of PCAN hardware to be initialized.
	TPCANTPMessageType	Represents the type of a PCAN-ISO-TP message.
	TPCANTPIdType	Represents the CAN ID type of a PCAN-ISO-TP message.
	TPCANTPFormatType	Represents the type of format of a PCAN-ISO-TP message.
	TPCANTPAddressingType	Represents the type of message addressing of a PCAN-ISO-TP message.
	TPCANTPConfirmation	Represents the network status of a communicated PCAN ISO-TP message.
	TPCANTPParameter	Represents a PCAN-ISO-TP parameter to be read or set.
	TPCANTPBitrateFD	Represents a bit rate string with flexible data rate (FD).

3.3 Classes

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

Classes

	Class	Description
	CanTpApi	Defines a class to use the PCAN-ISO-TP API within the Microsoft's .NET Framework programming environment.
	TCanTpApi	Defines a class to use the PCAN-ISO-TP API within the Delphi programming environment.

3.3.1 CanTpApi

Defines a class which represents the PCAN-ISO-TP API for using within the Microsoft's .NET Framework.

Syntax

C#

```
public static class CanTpApi
```

C++ / CLR

```
public ref class CanTpApi abstract sealed
```

Visual Basic

```
Public NotInheritable Class CanTpApi
```

Remarks

The CanTpApi class collects and implements the PCAN-ISO-TP API functions. Each method is called just like the API function with the exception that the prefix "CANTP_" is not used. The structure and functionality of the methods and API functions are the same.

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

```
TPCANTPStatus res;
// Static use without any instance.
//
res = CanTpApi.Initialize(CanTpApi.PCANTP_USBBUS1, TPCANTPBaudrate.PCANTP_BAUD_500K);
```

Note: This class under Delphi is called TCanTpApi.

See also: Methods on page 39, Definitions on page 106.

3.3.2 TCanTpApi

Defines a class which represents the PCAN-ISO-TP API for using within the Delphi programming environment.

Syntax

Pascal OO

```
TCanTpApi = class
```

Remarks

TCanTpApi is a class containing only class method versions 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-ISO-TP API functions. Each method is called just like the API function with the exception that the prefix "CANTP_" is not used. The structure and functionality of the methods and API functions are the same.

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

See also: Methods on page 39, Definitions on page 106.

3.4 Structures

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

Name	Description
TPCANTPMsg	Defines a CAN ISO-TP message.
TPCANTPTimestamp	Defines a time-stamp of a CAN ISO-TP message.

3.4.1 TPCANTPMsg

Defines a CAN ISO-TP message.

Syntax

C++

```
typedef struct
{
  BYTE SA;
  BYTE TA;
  TPCANTPAddressingType TA_TYPE;
  BYTE RA;
  TPCANTPIdType IDTYPE;
  TPCANTPMessageType MSGTYPE;
  TPCANTPFormatType FORMAT;
  BYTE DATA[4095];
  WORD LEN;
  TPCANTPConfirmation RESULT;
} TPCANMsg;
```

Pascal OO

```
TPCANMsg = record
    SA: Byte;
    TA: Byte;
    TA_TYPE: TPCANTPAddressingType;
    RA: Byte;

    IDTYPE: TPCANTPIdType;
    MSGTYPE: TPCANTPMessageType;
    FORMAT: TPCANTPFormatType;

    DATA: array[0..4094] of Byte;
    LEN: Word;
    RESULT: TPCANTPConfirmation;
end;
```

C#

```
public struct TPCANMsg
{
  public byte SA;
  public byte TA;
  [MarshalAs(UnmanagedType.U1)]
```

```
    public TPCANTPAddressingType TA_TYPE;
    public byte RA;

    [MarshalAs(UnmanagedType.U1)]
    public TPCANTPIdType IDTYPE;
    [MarshalAs(UnmanagedType.U1)]
    public TPCANTPMessageType MSGTYPE;
    [MarshalAs(UnmanagedType.U1)]
    public TPCANTPFormatType FORMAT;

    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4095)]
    public byte[] DATA;
    public ushort LEN;
    [MarshalAs(UnmanagedType.U1)]
    public TPCANTPConfirmation RESULT;
}
```

C++ / CLR

```
public value struct TPCANMsg
{
    Byte SA;
    Byte TA;
    [MarshalAs(UnmanagedType::U1)]
    TPCANTPAddressingType TA_TYPE;
    Byte RA;

    [MarshalAs(UnmanagedType::U1)]
    TPCANTPIdType IDTYPE;
    [MarshalAs(UnmanagedType::U1)]
    TPCANTPMessageType MSGTYPE;
    [MarshalAs(UnmanagedType::U1)]
    TPCANTPFormatType FORMAT;

    [MarshalAs(UnmanagedType::ByValArray, SizeConst = 4095)]
    array<Byte>^ DATA;
    unsigned short LEN;
    [MarshalAs(UnmanagedType::U1)]
    TPCANTPConfirmation RESULT;
}
```

Visual Basic

```
Public Structure TPCANMsg
        Public SA As Byte
        Public TA As Byte
        Public TA_TYPE As TPCANTPAddressingType
        Public RA As Byte

       <MarshalAs(UnmanagedType.U1)> _
       Public IDTYPE As TPCANTPIdType
       <MarshalAs(UnmanagedType.U1)> _
       Public MSGTYPE As TPCANTPMessageType
```

```
    <MarshalAs(UnmanagedType.U1)> _
    Public FORMAT As TPCANTPFormatType

    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4095)> _
    Public DATA As Byte()
    Public LEN As UShort
    <MarshalAs(UnmanagedType.U1)> _
    Public RESULT As TPCANTPConfirmation
End Structure
```

Fields

Name	Description
SA	Source Address
TA	Target Address
TA_TYPE	Target Address Type
RA	Remote Address
IDTYPE	CAN ID configuration
MSGTYPE	Type of the message
FORMAT	Addressing format
DATA	Raw data of the message
LEN	Data Length Code (DLC) of the message (0 – 4095)
RESULT	Network status result

See also:

CANTP_Read on page 101, class method version: Read on page 77.

CANTP_Write on page 103, class method version: Write on page 84.

3.4.2 TPCANTPTimestamp

Defines a time-stamp of a CAN ISO-TP message. The time-stamp of a CAN message contains the number of microseconds since the start of Windows.

Syntax

C++

```
typedef struct
{
    DWORD  millis;
    WORD   millis_overflow;
    WORD   micros;
} TPCANTPTimestamp;
```

Pascal OO

```
TPCANTPTimestamp = record
    millis: Longword;
    millis_overflow: Word;
    micros: Word;
 end;
PTPCANTPTimestamp = ^TPCANTPTimestamp;
```

C#

```
public value struct TPCANTPTimestamp
{
   public uint millis;
   public ushort millis_overflow;
   public ushort micros;
};
```

C++/CLR

```
public value struct TPCANTPTimestamp
{
      UInt32 millis;
      UInt16 millis_overflow;
      UInt16 micros;
};
```

Visual Basic

```
Public Structure TPCANTPTimestamp
      Public millis As UInt32
      Public millis_overflow As UInt16
      Public micros As UInt16
End Structure
```

Remarks

Calculation of total of microseconds: **micros + 1000 * millis + 0x100000000 * 1000 * millis_overflow**.

Fields

Name	Description
millis	Base-value: milliseconds: 0.. 2^32-1
millis_overflow	Roll-arounds of millis
micros	Microseconds: 0.. 999

See also: CANTP_Read on page 101, class method version: Read on page 77.

3.5 Types

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

Name	Description
TPCANTPHandle	Represents a PCAN-ISO-TP hardware channel handle.
TPCANTPStatus	Represents a PCAN-ISO-TP status/error code.
TPCANTPBaudrate	Represents a PCAN Baud rate register value.
TPCANTPHWType	Represents the type of PCAN hardware to be initialized.
TPCANTPMessageType	Represents the type of a PCAN-ISO-TP message.
TPCANTPIdType	Represents the CAN ID type of a PCAN-ISO-TP message.
TPCANTPFormatType	Represents the type of format of a PCAN-ISO-TP message.
TPCANTPAddressingType	Represents the type of message addressing of a PCAN-ISO-TP message.
TPCANTPConfirmation	Represents the network status of a communicated PCAN ISO-TP message.
TPCANTPParameter	Represents a PCAN-ISO-TP parameter to be read or set.
TPCANTPBitrateFD	Represents a bit rate string with flexible data rate (FD).

3.5.1 TPCANTPHandle

Represents a PCAN-ISO-TP channel handle.

Syntax

C++

```
#define TPCANTPHandle WORD
```

C++ / CLR

```
#define TPCANTPHandle System::Int16
```

C#

```
using TPCANTPHandle = System.Int16;
```

Visual Basic

```
Imports TPCANTPHandle = System.Int16
```

Remarks

TPCANTPHandle is defined for the ISO-TP API but it is identical to a 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 a homogeneity between all programming languages listed above.

Aliases are defined in the Peak.Can.IsoTp 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.IsoTp Namespace. Otherwise, just use the native type, which in this case is a byte.

C#

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

Visual Basic

```
Imports System
Imports Peak.Can.IsoTp
Imports TPCANTPHandle = System.Int16 ' Alias' declaration for System.Byte
```

See also: PCAN-ISO-TP Handle Definitions on page 106.

3.5.2 TPCANTPStatus

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

Syntax

C++

```
#define PCANTP_ERROR_OK 0x00000
#define PCANTP_ERROR_NOT_INITIALIZED 0x00001
#define PCANTP_ERROR_ALREADY_INITIALIZED 0x00002
#define PCANTP_ERROR_NO_MEMORY 0x00003
#define PCANTP_ERROR_OVERFLOW 0x00004
#define PCANTP_ERROR_TIMEOUT 0x00006
#define PCANTP_ERROR_NO_MESSAGE 0x00007
#define PCANTP_ERROR_WRONG_PARAM 0x00008
#define PCANTP_ERROR_BUSLIGHT 0x00009
#define PCANTP_ERROR_BUSHEAVY 0x0000A
#define PCANTP_ERROR_BUSOFF 0x0000B
#define PCANTP_ERROR_CAN_ERROR 0x80000000
```

C++ / CLR

```
public enum TPCANTPStatus : unsigned int
{
        PCANTP_ERROR_OK = 0x00000,
        PCANTP_ERROR_NOT_INITIALIZED = 0x00001,
        PCANTP_ERROR_ALREADY_INITIALIZED = 0x00002,
        PCANTP_ERROR_NO_MEMORY = 0x00003,
        PCANTP_ERROR_OVERFLOW = 0x00004,
        PCANTP_ERROR_TIMEOUT = 0x00006,
        PCANTP_ERROR_NO_MESSAGE = 0x00007,
        PCANTP_ERROR_WRONG_PARAM = 0x00008,
        PCANTP_ERROR_BUSLIGHT = 0x00009,
        PCANTP_ERROR_BUSHEAVY = 0x0000A,
        PCANTP_ERROR_BUSOFF = 0x0000B,
        PCANTP_ERROR_CAN_ERROR = 0x80000000,
};
```

C#

```
public enum TPCANTPStatus : uint
{
        PCANTP_ERROR_OK = 0x00000,
        PCANTP_ERROR_NOT_INITIALIZED = 0x00001,
        PCANTP_ERROR_ALREADY_INITIALIZED = 0x00002,
        PCANTP_ERROR_NO_MEMORY = 0x00003,
        PCANTP_ERROR_OVERFLOW = 0x00004,
        PCANTP_ERROR_TIMEOUT = 0x00006,
        PCANTP_ERROR_NO_MESSAGE = 0x00007,
        PCANTP_ERROR_WRONG_PARAM = 0x00008,
        PCANTP_ERROR_BUSLIGHT = 0x00009,
        PCANTP_ERROR_BUSHEAVY = 0x0000A,
        PCANTP_ERROR_BUSOFF = 0x0000B,
        PCANTP_ERROR_CAN_ERROR = 0x80000000,
};
```

Pascal OO

```
TPCANTPStatus = (
        PCANTP_ERROR_OK = $00000
        PCANTP_ERROR_NOT_INITIALIZED = $00001
        PCANTP_ERROR_ALREADY_INITIALIZED = $00002
        PCANTP_ERROR_NO_MEMORY = $00003
        PCANTP_ERROR_OVERFLOW = $00004
        PCANTP_ERROR_TIMEOUT = $00006
        PCANTP_ERROR_NO_MESSAGE = $00007
        PCANTP_ERROR_WRONG_PARAM = $00008
        PCANTP_ERROR_BUSLIGHT = $00009
        PCANTP_ERROR_BUSHEAVY = $0000A
        PCANTP_ERROR_BUSOFF = $0000B
        PCANTP_ERROR_CAN_ERROR = LongWord($80000000)
);
```

Visual Basic

```
Public Enum TPCANTPStatus As Integer
        PCANTP_ERROR_OK = &H0
        PCANTP_ERROR_NOT_INITIALIZED = &H1
        PCANTP_ERROR_ALREADY_INITIALIZED = &H2
        PCANTP_ERROR_NO_MEMORY = &H3
        PCANTP_ERROR_OVERFLOW = &H4
        PCANTP_ERROR_TIMEOUT = &H6
        PCANTP_ERROR_NO_MESSAGE = &H7
        PCANTP_ERROR_WRONG_PARAM = &H8
        PCANTP_ERROR_BUSLIGHT = &H9
        PCANTP_ERROR_BUSHEAVY = &HA
        PCANTP_ERROR_BUSOFF = &HB
        PCANTP_ERROR_CAN_ERROR = &H80000000
End Enum
```

Remarks

The PCANTP_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-ISO-TP API). When a PCAN-Basic error occurs, the API performs a bitwise combination of the PCANTP_ERROR_CAN_ERROR and the PCAN-Basic (TPCANStatus) error.

Values

Name	Value	Description
PCANTP_ERROR_OK	0x00000 (000000)	No error. Success
PCANTP_ERROR_NOT_INITIALIZED	0x00001 (000001)	Not initialized (ex. a non-initialized CANTP channel or CAN ID mapping).
PCANTP_ERROR_ALREADY_INITIALIZED	0x00002 (000002)	Already initialized (used by CANTP channel or CAN ID mapping).
PCANTP_ERROR_NO_MEMORY	0x00003 (000003)	Failed to allocate memory.
PCANTP_ERROR_OVERFLOW	0x00004 (000004)	Buffer overflow occurred (too many channels initialized or too many messages in queue).
PCANTP_ERROR_TIMEOUT	0x00006 (000006)	Timeout while trying to access the PCAN-ISO-TP API.
PCANTP_ERROR_NO_MESSAGE	0x00007 (000007)	No message available.
PCANTP_ERROR_WRONG_PARAM	0x00008 (000008)	Invalid parameter
PCANTP_ERROR_BUSLIGHT	0x00009 (000009)	Bus error: an error counter reached the 'light' limit.
PCANTP_ERROR_BUSHEAVY	0x0000A (000010)	Bus error: an error counter reached the 'heavy' limit.
PCANTP_ERROR_BUSOFF	0x0000B (000011)	Bus error: the CAN controller is in bus-off state.
PCANTP_ERROR_CAN_ERROR	0x80000000 (2147483648)	PCAN-Basic error flag (remove the flag to get a TPCANStatus error code)

3.5.3 TPCANTPBaudrate

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 TPCANBaudrate WORD

#define PCANTP_BAUD_1M 0x0014
#define PCANTP_BAUD_800K 0x0016
#define PCANTP_BAUD_500K 0x001C
#define PCANTP_BAUD_250K 0x011C
#define PCANTP_BAUD_125K 0x031C
#define PCANTP_BAUD_100K 0x432F
#define PCANTP_BAUD_95K 0xC34E
#define PCANTP_BAUD_83K 0x852B
#define PCANTP_BAUD_50K 0x472F
#define PCANTP_BAUD_47K 0x1414
#define PCANTP_BAUD_33K 0x8B2F
#define PCANTP_BAUD_20K 0x532F
#define PCANTP_BAUD_10K 0x672F
#define PCANTP_BAUD_5K 0x7F7F
```

Pascal OO

```
{$Z2}
TPCANBaudrate = (
```

```
  PCANTP_BAUD_1M = $0014,
  PCANTP_BAUD_800K = $0016,
  PCANTP_BAUD_500K = $001C,
  PCANTP_BAUD_250K = $011C,
  PCANTP_BAUD_125K = $031C,
  PCANTP_BAUD_100K = $432F,
  PCANTP_BAUD_95K = $C34E,
  PCANTP_BAUD_83K = $852B,
  PCANTP_BAUD_50K = $472F,
  PCANTP_BAUD_47K = $1414,
  PCANTP_BAUD_33K = $8B2F,
  PCANTP_BAUD_20K = $532F,
  PCANTP_BAUD_10K = $672F,
  PCANTP_BAUD_5K = $7F7F
);
```

C#

```
public enum TPCANBaudrate : ushort
{
  PCANTP_BAUD_1M = 0x0014,
  PCANTP_BAUD_800K = 0x0016,
  PCANTP_BAUD_500K = 0x001C,
  PCANTP_BAUD_250K = 0x011C,
  PCANTP_BAUD_125K = 0x031C,
  PCANTP_BAUD_100K = 0x432F,
  PCANTP_BAUD_95K = 0xC34E,
  PCANTP_BAUD_83K = 0x852B,
  PCANTP_BAUD_50K = 0x472F,
  PCANTP_BAUD_47K = 0x1414,
  PCANTP_BAUD_33K = 0x8B2F,
  PCANTP_BAUD_20K = 0x532F,
  PCANTP_BAUD_10K = 0x672F,
  PCANTP_BAUD_5K = 0x7F7F,
}
```

C++ / CLR

```
public enum class TPCANTPBaudrate : UInt16
{
  PCANTP_BAUD_1M = 0x0014,
  PCANTP_BAUD_800K = 0x0016,
  PCANTP_BAUD_500K = 0x001C,
  PCANTP_BAUD_250K = 0x011C,
  PCANTP_BAUD_125K = 0x031C,
  PCANTP_BAUD_100K = 0x432F,
  PCANTP_BAUD_95K = 0xC34E,
  PCANTP_BAUD_83K = 0x852B,
  PCANTP_BAUD_50K = 0x472F,
  PCANTP_BAUD_47K = 0x1414,
  PCANTP_BAUD_33K = 0x8B2F,
  PCANTP_BAUD_20K = 0x532F,
```

```
  PCANTP_BAUD_10K = 0x672F,
  PCANTP_BAUD_5K = 0x7F7F,
};
```

Visual Basic

```
Public Enum TPCANTPBaudrate As UInt16
  PCANTP_BAUD_1M = &H14
  PCANTP_BAUD_800K = &H16
  PCANTP_BAUD_500K = &H1C
  PCANTP_BAUD_250K = &H11C
  PCANTP_BAUD_125K = &H31C
  PCANTP_BAUD_100K = &H432F
  PCANTP_BAUD_95K = &C34E
  PCANTP_BAUD_83K = &852B
  PCANTP_BAUD_50K = &H472F
  PCANTP_BAUD_47K = &1414
  PCANTP_BAUD_33K = &8B2F
  PCANTP_BAUD_20K = &H532F
  PCANTP_BAUD_10K = &H672F
  PCANTP_BAUD_5K = &H7F7F
End Enum
```

Values

Name	Value	Description
PCANTP_BAUD_1M	20	1 MBit/s
PCANTP_BAUD_800K	22	800 kBit/s
PCANTP_BAUD_500K	28	500 kBit/s
PCANTP_BAUD_250K	284	250 kBit/s
PCANTP_BAUD_125K	796	125 kBit/s
PCANTP_BAUD_100K	17199	100 kBit/s
PCANTP_BAUD_95K	49998	95,238 kBit/s
PCANTP_BAUD_83K	34091	83,333 kBit/s
PCANTP_BAUD_50K	18223	50 kBit/s
PCANTP_BAUD_47K	5140	47,619 kBit/s
PCANTP_BAUD_33K	35631	33,333 kBit/s
PCANTP_BAUD_20K	21295	20 kBit/s
PCANTP_BAUD_10K	26415	10 kBit/s
PCANTP_BAUD_5K	32639	5 kBit/s

See also: CANTP_Initialize on page 90, class method version: Initialize on page 40.

3.5.4 TPCANTPBitrateFD

Represents a bit rate string with flexible data rate (FD).

Syntax

C++

```
#define TPCANTPBitrateFD LPSTR
```

Pascal OO

```
TPCANTPBitrateFD = String;
```

C#

```
using TPCANTPBitrateFD = System.String;
```

C++ / CLR

```
#define TPCANTPBitrateFD System::String^
```

Visual Basic

```
Imports TPCANTPBitrateFD = System.String
```

Remarks

.NET Framework programming languages:

An alias is used to represent a flexible data rate under Microsoft .NET in order to originate a homogeneity between all programming languages listed above.

Aliases are defined in the Peak.Can.IsoTp 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.IsoTp Namespace. Otherwise, just use the native type, which in this case is a string.

C#

```
using System;
using Peak.CAN.ISoTp;
using TPCANBitrateFD = System.String; // Alias' declaration for System.String
```

Visual Basic

```
Imports System
Imports Peak.CAN.ISoTp
Imports TPCANBitrateFD = System.String ' Alias declaration for System.String
```

See Also: FD Bit Rate Parameter Definitions on page 108.

3.5.5 TPCANTPHWType

Represents the type of PCAN (non-Plug and 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 TPCANTPHWType BYTE

#define PCANTP_TYPE_ISA 0x01
#define PCANTP_TYPE_ISA_SJA 0x09
#define PCANTP_TYPE_ISA_PHYTEC 0x04
#define PCANTP_TYPE_DNG 0x02
#define PCANTP_TYPE_DNG_EPP 0x03
```

```
#define PCANTP_TYPE_DNG_SJA 0x05
#define PCANTP_TYPE_DNG_SJA_EPP 0x06
```

Pascal OO

```
{$Z1}
TPCANTPHWType = (
  PCANTP_TYPE_ISA = $01,
  PCANTP_TYPE_ISA_SJA = $09,
  PCANTP_TYPE_ISA_PHYTEC = $04,
  PCANTP_TYPE_DNG = $02,
  PCANTP_TYPE_DNG_EPP = $03,
  PCANTP_TYPE_DNG_SJA = $05,
  PCANTP_TYPE_DNG_SJA_EPP = $06
);
```

C#

```
public enum TPCANTPHWType : byte
{
  PCANTP_TYPE_ISA = 0x01,
  PCANTP_TYPE_ISA_SJA = 0x09,
  PCANTP_TYPE_ISA_PHYTEC = 0x04,
  PCANTP_TYPE_DNG = 0x02,
  PCANTP_TYPE_DNG_EPP = 0x03,
  PCANTP_TYPE_DNG_SJA = 0x05,
  PCANTP_TYPE_DNG_SJA_EPP = 0x06,
}
```

C++ / CLR

```
public enum class TPCANTPHWType : Byte
{
  PCANTP_TYPE_ISA = 0x01,
  PCANTP_TYPE_ISA_SJA = 0x09,
  PCANTP_TYPE_ISA_PHYTEC = 0x04,
  PCANTP_TYPE_DNG = 0x02,
  PCANTP_TYPE_DNG_EPP = 0x03,
  PCANTP_TYPE_DNG_SJA = 0x05,
  PCANTP_TYPE_DNG_SJA_EPP = 0x06,
};
```

Visual Basic

```
Public Enum TPCANTPHWType As Byte
  PCANTP_TYPE_ISA = &H1
  PCANTP_TYPE_ISA_SJA = &H9
  PCANTP_TYPE_ISA_PHYTEC = &H4
  PCANTP_TYPE_DNG = &H2
  PCANTP_TYPE_DNG_EPP = &H3
  PCANTP_TYPE_DNG_SJA = &H5
  PCANTP_TYPE_DNG_SJA_EPP = &H6
End Enum
```

Values

Name	Value	Description
PCANTP_TYPE_ISA	1	PCAN-ISA 82C200
PCANTP_TYPE_ISA_SJA	9	PCAN-ISA SJA1000
PCANTP_TYPE_ISA_PHYTEC	4	PHYTEC ISA
PCANTP_TYPE_DNG	2	PCAN-Dongle 82C200
PCANTP_TYPE_DNG_EPP	3	PCAN-Dongle EPP 82C200
PCANTP_TYPE_DNG_SJA	5	PCAN-Dongle SJA1000
PCANTP_TYPE_DNG_SJA_EPP	6	PCAN-Dongle EPP SJA1000

See also: PCANTP_Initialize, class method version: initialize.

3.5.6 TPCANTPMessageType

Represents the type of PCAN ISO-TP messages. ISO-TP defines 2 types of message (diagnostic and remote diagnostic), the API adds two more for information purpose.

Syntax

C++

```
#define TPCANTPMessageType BYTE

#define PCANTP_MESSAGE_UNKNOWN 0x00
#define PCANTP_MESSAGE_DIAGNOSTIC 0x01
#define PCANTP_MESSAGE_REMOTE_DIAGNOSTIC 0x02
#define PCANTP_MESSAGE_REQUEST_CONFIRMATION 0x03
#define PCANTP_MESSAGE_INDICATION 0x04
```

Pascal OO

```
{$Z1}
TPCANTPMessageType = (
   PCANTP_MESSAGE_UNKNOWN = $00,
   PCANTP_MESSAGE_DIAGNOSTIC = $01,
   PCANTP_MESSAGE_REMOTE_DIAGNOSTIC = $02,
   PCANTP_MESSAGE_REQUEST_CONFIRMATION = $03,
   PCANTP_MESSAGE_INDICATION = $04
);
```

C#

```
public enum TPCANTPMessageType : byte
{
   PCANTP_MESSAGE_UNKNOWN = 0x00,
   PCANTP_MESSAGE_DIAGNOSTIC = 0x01,
   PCANTP_MESSAGE_REMOTE_DIAGNOSTIC = 0x02,
   PCANTP_MESSAGE_REQUEST_CONFIRMATION = 0x03,
   PCANTP_MESSAGE_INDICATION = 0x04,
}
```

C++ / CLR

```
public enum class TPCANTPMessageType : Byte
{
  PCANTP_MESSAGE_UNKNOWN = 0x00,
  PCANTP_MESSAGE_DIAGNOSTIC = 0x01,
  PCANTP_MESSAGE_REMOTE_DIAGNOSTIC = 0x02,
  PCANTP_MESSAGE_REQUEST_CONFIRMATION = 0x03,
  PCANTP_MESSAGE_INDICATION = 0x04,
};
```

Visual Basic

```
Public Enum TPCANTPMessageType As Byte
  PCANTP_MESSAGE_UNKNOWN = &H0
  PCANTP_MESSAGE_DIAGNOSTIC = &H1
  PCANTP_MESSAGE_REMOTE_DIAGNOSTIC = &H2
  PCANTP_MESSAGE_REQUEST_CONFIRMATION = &H3
  PCANTP_MESSAGE_INDICATION = &H4
End Enum
```

Values

Name	Value	Description
PCANTP_MESSAGE_UNKNOWN	0	Unknown (non-ISO-TP) message
PCANTP_MESSAGE_DIAGNOSTIC	1	Diagnostic message
PCANTP_MESSAGE_REMOTE_DIAGNOSTIC	2	Remote Diagnostic message (ie. message uses the Remote Address (RA) field)
PCANTP_MESSAGE_REQUEST_CONFIRMATION	3	Notification: message has been sent (successfully or not).
PCANTP_MESSAGE_INDICATION	4	Notification: multi-Frame message is being received.
PCANTP_MESSAGE_INDICATION_TX	5	Notification: multi-Frame message is being transmitted.

See also: TPCANTPMsg on page 12, CANTP_AddMapping on page 96, Primary Language ID.

3.5.7 TPCANTPIdType

Represents the ID type of CAN messages (standard/11bits or extended/29bits).

Syntax

C++

```
#define TPCANTPIdType BYTE

#define PCANTP_ID_CAN_11BIT 0x01
#define PCANTP_ID_CAN_29BIT 0x02

#define PCANTP_ID_CAN_FD    0x04
#define PCANTP_ID_CAN_BRS   0x08
```

Pascal OO

```
{$Z1}
TPCANTPIdType = (
  PCANTP_ID_CAN_11BIT = $01,
  PCANTP_ID_CAN_29BIT = $02,
);
```

C#

```
public enum TPCANTPIdType : byte
{
   PCANTP_ID_CAN_11BIT = 0x01,
   PCANTP_ID_CAN_29BIT = 0x02,
}
```

C++ / CLR

```
public enum class TPCANTPIdType : Byte
{
   PCANTP_ID_CAN_11BIT = 0x01,
   PCANTP_ID_CAN_29BIT = 0x02,
};
```

Visual Basic

```
Public Enum TPCANTPIdType As Byte
   PCANTP_ID_CAN_11BIT = &H1
   PCANTP_ID_CAN_29BIT = &H2
End Enum
```

Values

Name	Value	Description
PCANTP_ID_CAN_11BIT	1	Standard CAN ID (11-bit identifier)
PCANTP_ID_CAN_29BIT	2	Extended CAN ID (29-bit identifier)

Note: Since version 1.4.3.40, IDTYPE can include the priority parameter for ISO-TP messages compliant with J1939 data link layer. This applies for a 29-bit CAN ID messages with normal fixed, mixed, or enhanced addressing.

The following shows how the IDTYPE member is encoded:

Bits	7..5	4	3..0
Variable	Priority value	Priority flag	CAN ID type

- CAN ID Type (bits 0 to 3) contains the usual value for a TPCANTPIdType (PCANTP_ID_CAN_11BIT or PCANTP_ID_CAN_29BIT).
- The bit #4 „Priority flag“ states whether the priority value is set or not.
- Priority value (bits 5 to 7) holds the 3 bits priority according to SAE J1939.

The API provides several macros or functions to manage the priority value:

- PCANTP_ID_CAN_GET_29B(j1939_priority: Byte): returns the IDTYPE value for a a 29-bit CAN ID with the specified priority.
- PCANTP_ID_CAN_GET_PRIORIY(id_type: Byte): reads and returns the priority from an IDTYPE value.
- PCANTP_ID_CAN_IS_EXTENDED(id_type: Byte): states if the IDTYPE value corresponds to a a 29-bit CAN ID.
- PCANTP_ID_CAN_HAS_PRIORITY(id_type: Byte) : states if the IDTYPE value corresponds to a a 29-bit CAN ID with a defined priority.

Syntax

C++

```
#define PCANTP_ID_CAN_MASK                0x0F
#define PCANTP_ID_CAN_IS_PRIORITY_MASK    0x10
#define PCANTP_ID_CAN_PRIORITY_MASK       0xE0

#define PCANTP_ID_CAN_GET_29B(j1939_priority)   ((BYTE)((j1939_priority << 5) |
PCANTP_ID_CAN_IS_PRIORITY_MASK | (PCANTP_ID_CAN_29BIT & PCANTP_ID_CAN_MASK)))

#define PCANTP_ID_CAN_GET_PRIORIY(id_type)      ((BYTE)((id_type & PCANTP_ID_CAN_PRIORITY_MASK)
>> 5))

#define PCANTP_ID_CAN_IS_EXTENDED(id_type)      ((id_type & PCANTP_ID_CAN_MASK) ==
PCANTP_ID_CAN_29BIT)
#define  PCANTP_ID_CAN_HAS_PRIORITY(id_type) (PCANTP_ID_CAN_IS_EXTENDED(id_type) && ((id_type &
PCANTP_ID_CAN_IS_PRIORITY_MASK) == PCANTP_ID_CAN_IS_PRIORITY_MASK))
```

Pascal OO

```
class function PCANTP_ID_CAN_GET_29B(j1939_priority: Byte): TPCANTPIdType;

class function PCANTP_ID_CAN_GET_PRIORIY(id_type: Byte): Byte;

class function PCANTP_ID_CAN_IS_EXTENDED(id_type: Byte): Boolean;

class function PCANTP_ID_CAN_HAS_PRIORITY(id_type: Byte): Boolean;
```

C#

```
public static class TPCANTPIdTypePriority
{
    public static TPCANTPIdType PCANTP_ID_CAN_GET_29B(byte j1939_priority) ;
    public static byte PCANTP_ID_CAN_GET_PRIORIY(byte id_type) ;
    public static bool PCANTP_ID_CAN_IS_EXTENDED(byte id_type) ;
    public static bool PCANTP_ID_CAN_HAS_PRIORITY(byte id_type) ;
}
```

C++ / CLR

```
public ref class TPCANTPIdTypePriority
{
public:
       static TPCANTPIdType PCANTP_ID_CAN_GET_29B(Byte j1939_priority);
       static Byte PCANTP_ID_CAN_GET_PRIORIY(Byte id_type);
       static bool PCANTP_ID_CAN_IS_EXTENDED(Byte id_type);
       static bool PCANTP_ID_CAN_HAS_PRIORITY(Byte id_type);
};
```

Visual Basic

```
Public Class TPCANTPIdTypePriority
    Public Shared Function PCANTP_ID_CAN_GET_29B(ByVal j1939_priority As Byte) As TPCANTPIdType
    End Function
    Public Shared Function PCANTP_ID_CAN_GET_PRIORIY(ByVal id_type As Byte) As Byte
    End Function
    Public Shared Function PCANTP_ID_CAN_IS_EXTENDED(ByVal id_type As Byte) As Boolean
    End Function
    Public Shared Function PCANTP_ID_CAN_HAS_PRIORITY(ByVal id_type As Byte) As Boolean
    End Function
End Class
```

Note: Since version 1.4.3.54, IDTYPE can include flags that define if the message is CAN FD and use the Bitrate Switch:

- PCANTP_ID_CAN_FD: This flag defines that the message is FD capable.
- PCANTP_ID_CAN_BRS: This flag is only valid when the previous flag is set, it enables the "Data Bitrate Switch" meaning that the data of the message will be transmitted at the data speed rate.

The following example defines a 29-bit ISO-TP message with FD and BRS flags:

```
TPCANTPMsg Message;
memset(&Message, 0, sizeof(TPCANTPMsg));
Message.IDTYPE = PCANTP_ID_CAN_29BIT;
Message.IDTYPE |= PCANTP_ID_CAN_FD | PCANTP_ID_CAN_BRS;
```

See also: TPCANTPMsg on page 24, CANTP_AddMapping on page 96.

3.5.8 TPCANTPFormatType

Represents the format addressing type of CAN ISO-TP messages.

Syntax

C++

```
#define TPCANTPFormatType BYTE

#define PCANTP_FORMAT_UNKNOWN 0xFF
#define PCANTP_FORMAT_NONE 0x00
#define PCANTP_FORMAT_NORMAL 0x01
#define PCANTP_FORMAT_FIXED_NORMAL 0x02
#define PCANTP_FORMAT_EXTENDED 0x03
#define PCANTP_FORMAT_MIXED 0x04
#define PCANTP_FORMAT_ENHANCED 0x05
```

Pascal OO

```
{$Z1}
TPCANTPFormatType = (
  PCANTP_FORMAT_UNKNOWN = $FF
  PCANTP_FORMAT_NONE = $00,
  PCANTP_FORMAT_NORMAL = $01,
  PCANTP_FORMAT_FIXED_NORMAL = $02,
  PCANTP_FORMAT_EXTENDED = $03,
  PCANTP_FORMAT_MIXED = $04,
  PCANTP_FORMAT_ENHANCED = $05,
);
```

C#

```
public enum TPCANTPFormatType : byte
{
   PCANTP_FORMAT_UNKNOWN = 0xFF,
   PCANTP_FORMAT_NONE = 0x00,
   PCANTP_FORMAT_NORMAL = 0x01,
   PCANTP_FORMAT_FIXED_NORMAL = 0x02,
   PCANTP_FORMAT_EXTENDED = 0x03,
   PCANTP_FORMAT_MIXED = 0x04,
   PCANTP_FORMAT_ENHANCED = 0x05,
}
```

C++ / CLR

```
public enum class TPCANTPFormatType : Byte
{
   PCANTP_FORMAT_UNKNOWN = 0xFF,
   PCANTP_FORMAT_NONE = 0x00,
   PCANTP_FORMAT_NORMAL = 0x01,
   PCANTP_FORMAT_FIXED_NORMAL = 0x02,
   PCANTP_FORMAT_EXTENDED = 0x03,
   PCANTP_FORMAT_MIXED = 0x04,
   PCANTP_FORMAT_ENHANCED = 0x05,
};
```

Visual Basic

```
Public Enum TPCANTPFormatType As Byte
   PCANTP_FORMAT_UNKNOWN = &HFF
   PCANTP_FORMAT_NONE = &H00
   PCANTP_FORMAT_NORMAL = &H01
   PCANTP_FORMAT_FIXED_NORMAL = &H02
   PCANTP_FORMAT_EXTENDED = &H03
   PCANTP_FORMAT_MIXED = &H04
   PCANTP_FORMAT_ENHANCED = &H05
End Enum
```

Values

Name	Value	Description
PCANTP_FORMAT_UNKNOWN	255	Unknown addressing format
PCANTP_FORMAT_NONE	0	Unsegmented CAN frame (non ISO-TP)
PCANTP_FORMAT_NORMAL	1	Normal addressing format
PCANTP_FORMAT_FIXED_NORMAL	2	Fixed Normal addressing format
PCANTP_FORMAT_EXTENDED	3	Extended addressing format
PCANTP_FORMAT_MIXED	4	Mixed Addressing format
PCANTP_FORMAT_ENHANCED	5	Enhanced addressing format (defined in ISO-15765-3)

Note: To send an unsegmented CAN message with PCAN-ISO-TP API, the FORMAT member of a TPCANTPMsg must be set with the value PCANTP_FORMAT_NONE.

Since the ISO-TP message structure doesn't contain a field to define the CAN ID, it must be stored in the four first bytes of the DATA structure (Fourth byte corresponds to the Least Significant Byte).

The following C++ example shows how to transmit a standard CAN message with ISO-TP:

```
TPCANTPMsg msg;
int i, idx;

memset(&msg, 0, sizeof(msg));
msg.FORMAT = PCANTP_FORMAT_NONE;
msg.MSGTYPE = PCANTP_MESSAGE_UNKNOWN;
msg.TA_TYPE = PCANTP_ADDRESSING_UNKNOWN;
msg.IDTYPE = PCANTP_ID_CAN_29BIT;

// Sending CAN frame with CAN ID : 0x3FAB0123.
idx = 0;
msg.DATA[idx++] = 0x3F;
msg.DATA[idx++] = 0xAB;
msg.DATA[idx++] = 0x01;
msg.DATA[idx++] = 0x23;
// Initialize CAN DATA.
msg.LEN = idx + 8;
for (i = idx; i < msg.LEN; i++) {
      msg.DATA[i] = 0xA1 + i;
}
// Transmit message.
sts = CANTP_Write(channel, &msg);
if (sts == PCANTP_ERROR_OK) {
      // Message added to transmit queue.
}
```

See also: TPCANTPMsg on page 24, CANTP_AddMapping on page 96.

3.5.9 TPCANTPAddressingType

Represents the format addressing type of CAN ISO-TP messages.

Syntax

C++

```
#define TPCANTPAddressingType BYTE

#define PCANTP_ADDRESSING_UNKNOWN 0x00
#define PCANTP_ADDRESSING_PHYSICAL 0x01
#define PCANTP_ADDRESSING_FUNCTIONAL 0x02
```

Pascal OO

```
{$Z1}
TPCANTPAddressingType = (
  PCANTP_ADDRESSING_UNKNOWN = $00
  PCANTP_ADDRESSING_PHYSICAL = $01,
  PCANTP_ADDRESSING_FUNCTIONAL = $02,
);
```

C#

```
public enum TPCANTPAddressingType : byte
{
```

```
  PCANTP_ADDRESSING_UNKNOWN = 0x00,
  PCANTP_ADDRESSING_PHYSICAL = 0x01,
  PCANTP_ADDRESSING_FUNCTIONAL = 0x02,
}
```

C++ / CLR

```
public enum class TPCANTPAddressingType : Byte
{
  PCANTP_ADDRESSING_UNKNOWN = 0x00,
  PCANTP_ADDRESSING_PHYSICAL = 0x01,
  PCANTP_ADDRESSING_FUNCTIONAL = 0x02,
};
```

Visual Basic

```
Public Enum TPCANTPAddressingType As Byte
  PCANTP_ADDRESSING_UNKNOWN = &H00
  PCANTP_ADDRESSING_PHYSICAL = &H01
  PCANTP_ADDRESSING_FUNCTIONAL = &H02
End Enum
```

Values

Name	Value	Description
PCANTP_ADDRESSING_UNKNOWN	0	Unknown addressing
PCANTP_ADDRESSING_PHYSICAL	1	Physical addressing (used for communication between 2 nodes)
PCANTP_ADDRESSING_FUNCTIONAL	2	Functional addressing (used to broadcast messages on the CAN bus)

See also: TPCANTPMsg on page 12, Primary Language ID, CANTP_AddMapping on page 96.

3.5.10 TPCANTPConfirmation

Represents the network status of a communicated CAN ISO-TP message.

Syntax

C++

```
#define TPCANTPConfirmation BYTE

#define PCANTP_N_OK 0x00
#define PCANTP_N_TIMEOUT_A 0x01
#define PCANTP_N_TIMEOUT_Bs 0x02
#define PCANTP_N_TIMEOUT_Cr 0x03
#define PCANTP_N_WRONG_SN 0x04
#define PCANTP_N_INVALID_FS 0x05
#define PCANTP_N_UNEXP_PDU 0x06
#define PCANTP_N_WFT_OVRN 0x07
#define PCANTP_N_BUFFER_OVFLW 0x08
#define PCANTP_N_ERROR 0x09
```

Pascal OO

```
{$Z1}
TPCANTPConfirmation = (
   PCANTP_N_OK = $00,
   PCANTP_N_TIMEOUT_A = $01,
   PCANTP_N_TIMEOUT_Bs = $02,
   PCANTP_N_TIMEOUT_Cr = $03,
   PCANTP_N_WRONG_SN = $04,
   PCANTP_N_INVALID_FS = $05,
   PCANTP_N_UNEXP_PDU = $06,
   PCANTP_N_WFT_OVRN = $07,
   PCANTP_N_BUFFER_OVFLW = $08,
   PCANTP_N_ERROR = $09
);
```

C#

```
public enum TPCANTPConfirmation : byte
{
   PCANTP_N_OK = 0x00,
   PCANTP_N_TIMEOUT_A = 0x01,
   PCANTP_N_TIMEOUT_Bs = 0x02,
   PCANTP_N_TIMEOUT_Cr = 0x03,
   PCANTP_N_WRONG_SN = 0x04,
   PCANTP_N_INVALID_FS = 0x05,
   PCANTP_N_UNEXP_PDU = 0x06,
   PCANTP_N_WFT_OVRN = 0x07,
   PCANTP_N_BUFFER_OVFLW = 0x08,
   PCANTP_N_ERROR = 0x09,
}
```

C++ / CLR

```
public enum class TPCANTPConfirmation : Byte
{
   PCANTP_N_OK = 0x00,
   PCANTP_N_TIMEOUT_A = 0x01,
   PCANTP_N_TIMEOUT_Bs = 0x02,
   PCANTP_N_TIMEOUT_Cr = 0x03,
   PCANTP_N_WRONG_SN = 0x04,
   PCANTP_N_INVALID_FS = 0x05,
   PCANTP_N_UNEXP_PDU = 0x06,
   PCANTP_N_WFT_OVRN = 0x07,
   PCANTP_N_BUFFER_OVFLW = 0x08,
   PCANTP_N_ERROR = 0x09,
};
```

Visual Basic

```
Public Enum TPCANTPConfirmation As Byte
   PCANTP_N_OK = 0x00,
   PCANTP_N_TIMEOUT_A = &H01,
```

```
  PCANTP_N_TIMEOUT_Bs = &H02
  PCANTP_N_TIMEOUT_Cr = &H03
  PCANTP_N_WRONG_SN = &H04
  PCANTP_N_INVALID_FS = &H05
  PCANTP_N_UNEXP_PDU = &H06
  PCANTP_N_WFT_OVRN = &H07
  PCANTP_N_BUFFER_OVFLW = &H08
  PCANTP_N_ERROR = &H09
End Enum
```

Values

Name	Value	Description
PCANTP_N_OK	0	No network error
PCANTP_N_TIMEOUT_A	1	Timeout occured between 2 frames transmission (sender and receiver side).
PCANTP_N_TIMEOUT_Bs	2	Sender side timeout while waiting for flow control frame.
PCANTP_N_TIMEOUT_Cr	3	Receiver side timeout while waiting for consecutive frame.
PCANTP_N_WRONG_SN	4	Unexpected sequence number
PCANTP_N_INVALID_FS	5	Invalid or unknown FlowStatus
PCANTP_N_UNEXP_PDU	6	Unexpected protocol data unit
PCANTP_N_WFT_OVRN	7	Reception of flow control WAIT frame that exceeds the maximum counter defined by PCANTP_PARAM_WFT_MAX.
PCANTP_N_BUFFER_OVFLW	8	Buffer on the receiver side cannot store the data length (server side only).
PCANTP_N_ERROR	9	General error

See also: TPCANTPMsg on page 12.

3.5.11 TPCANTPParameter

Represents a PCAN-ISO-TP parameter or a PCAN-ISO-TP 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#

```
public enum TPCANTPParameter : byte
{
  PCANTP_PARAM_BLOCK_SIZE = 0xE1,
  PCANTP_PARAM_SEPERATION_TIME = 0xE2,
  PCANTP_PARAM_DEBUG = 0xE3,
  PCANTP_PARAM_CHANNEL_CONDITION = 0xE4,
  PCANTP_PARAM_WFT_MAX = 0xE5,
  PCANTP_PARAM_MSG_PENDING = 0xE6,
  PCANTP_PARAM_API_VERSION = 0xE7,
  PCANTP_PARAM_CAN_DATA_PADDING = 0xE8,
  PCANTP_PARAM_CAN_UNSEGMENTED = 0xE9,
  PCANTP_PARAM_RECEIVE_EVENT = 0xEA,
  PCANTP_PARAM_PADDING_VALUE = 0xED,
  PCANTP_PARAM_J1939_PRIORITY = 0xEE,
  PCANTP_PARAM_CAN_TX_DL = 0xEF,
  PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION = 0xF0,
```

```
}
```

C++ / CLR

```
public enum TPCANTPParameter : Byte
{
   PCANTP_PARAM_BLOCK_SIZE = 0xE1,
   PCANTP_PARAM_SEPERATION_TIME = 0xE2,
   PCANTP_PARAM_DEBUG = 0xE3,
   PCANTP_PARAM_CHANNEL_CONDITION = 0xE4,
   PCANTP_PARAM_WFT_MAX = 0xE5,
   PCANTP_PARAM_MSG_PENDING = 0xE6,
   PCANTP_PARAM_API_VERSION = 0xE7,
   PCANTP_PARAM_CAN_DATA_PADDING = 0xE8,
   PCANTP_PARAM_CAN_UNSEGMENTED = 0xE9,
   PCANTP_PARAM_RECEIVE_EVENT = 0xEA,
   PCANTP_PARAM_PADDING_VALUE = 0xED,
   PCANTP_PARAM_J1939_PRIORITY = 0xEE,
   PCANTP_PARAM_CAN_TX_DL = 0xEF,
   PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION = 0xF0,
}
```

C++

```
#define PCANTP_PARAM_BLOCK_SIZE 0xE1
#define PCANTP_PARAM_SEPERATION_TIME 0xE2
#define PCANTP_PARAM_DEBUG 0xE3
#define PCANTP_PARAM_CHANNEL_CONDITION 0xE4
#define PCANTP_PARAM_WFT_MAX 0xE5
#define PCANTP_PARAM_MSG_PENDING 0xE6
#define PCANTP_PARAM_API_VERSION 0xE7
#define PCANTP_PARAM_CAN_DATA_PADDING 0xE8
#define PCANTP_PARAM_CAN_UNSEGMENTED 0xE9
#define PCANTP_PARAM_RECEIVE_EVENT 0xEA
#define PCANTP_PARAM_PADDING_VALUE 0xED
#define PCANTP_PARAM_J1939_PRIORITY 0xEE
#define PCANTP_PARAM_CAN_TX_DL 0xEF
#define PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION 0xF0
```

Pascal OO

```
{$Z1}
TPCANTPParameter = (
   PCANTP_PARAM_BLOCK_SIZE = $E1,
   PCANTP_PARAM_SEPERATION_TIME = $E2,
   PCANTP_PARAM_DEBUG = $E3,
   PCANTP_PARAM_CHANNEL_CONDITION = $E4,
   PCANTP_PARAM_WFT_MAX = $E5,
   PCANTP_PARAM_MSG_PENDING = $E6,
   PCANTP_PARAM_API_VERSION = $E7,
   PCANTP_PARAM_CAN_DATA_PADDING = $E8,
   PCANTP_PARAM_UNSEGMENTED = $E9
   PCANTP_PARAM_RECEIVE_EVENT = $EA,
```

```
    PCANTP_PARAM_PADDING_VALUE = $ED,
    PCANTP_PARAM_J1939_PRIORITY = $EE
    PCANTP_PARAM_CAN_TX_DL = $EF
    PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION = $F0
);
```

Visual Basic

```
Public Enum TPCANTPParameter As Byte
    PCANTP_PARAM_BLOCK_SIZE = &HE1
    PCANTP_PARAM_SEPERATION_TIME = &HE2
    PCANTP_PARAM_DEBUG = &HE3
    PCANTP_PARAM_CHANNEL_CONDITION = &HE4
    PCANTP_PARAM_WFT_MAX = &HE5
    PCANTP_PARAM_MSG_PENDING = &HE6
    PCANTP_PARAM_API_VERSION = &HE7
    PCANTP_PARAM_CAN_DATA_PADDING = &HE8
    PCANTP_PARAM_UNSEGMENTED = &HE9
    PCANTP_PARAM_RECEIVE_EVENT = &HEA
    PCANTP_PARAM_PADDING_VALUE = &HED
    PCANTP_PARAM_J1939_PRIORITY = &HEE
    PCANTP_PARAM_CAN_TX_DL = &HEF
    PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION = &HF0
End Enum
```

Values

Name	Value	Data type	Description
PCANTP_PARAM_BLOCK_SIZE	0xE1	Byte	ISO-TP "BlockSize" (BS) parameter
PCANTP_PARAM_SEPARATION_TIME	0xE2	Byte	ISO-TP "SeparationTime" (STmin) parameter
PCANTP_PARAM_DEBUG	0xE3	Byte	Debug mode
PCANTP_PARAM_CHANNEL_CONDITION	0xE4	Byte	PCAN-ISO-TP channel condition
PCANTP_PARAM_WFT_MAX	0xE5	Int32	ISO-TP "N_WFTmax" parameter
PCANTP_PARAM_MSG_PENDING	0xE6	Byte	Filter for message indication
PCANTP_PARAM_API_VERSION	0xE7	String	API version
PCANTP_PARAM_CAN_DATA_PADDING	0xE8	Byte	ISO-TP CAN frame data handling mode
PCANTP_PARAM_UNSEGMENTED	0xE9	Byte	Handling of non ISO-TP message
PCANTP_PARAM_RECEIVE_EVENT	0xEA	Byte	PCAN-ISO-TP receive event handler parameter
PCANTP_PARAM_PADDING_VALUE	0xED	Byte	Value used when CAN Data padding is enabled.
PCANTP_PARAM_J1939_PRIORITY	0xEE	Byte	Priority value used when ISO-TP J1939 compliant messages are sent.
PCANTP_PARAM_CAN_TX_DL	0xEF	Byte	The Data Length Code (DLC) of the fragmented frames used when transmitting FD messages.
PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION	0xF0	Byte	Optimization of the Minimum SeparationTime latency

Characteristics

PCANTP_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 PCANTP_NONEBUS channel).

PCANTP_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 to 0x7F (range from 0 to 127 ms) and 0xF1 to 0xF9 (range from 100 to 900 µs).

Note: Values between 0xF1 to 0xF3 should define a minimum time of 100 to 300 µs, but in practice the time to transmit effectively a frame takes about 300 µs (which is to send the message to the CAN controller and to assert that the message is physically emitted on the CAN bus). Other values than the ones stated above are ISO reserved.

Default value: 10ms.

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

PCANTP_PARAM_DEBUG

Access: R W

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

Possible values: PCANTP_DEBUG_NONE disables debug mode and PCANTP_DEBUG_CAN enables it.

Default value: PCANTP_DEBUG_NONE.

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

PCANTP_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: PCANTP_CHANNEL_UNAVAILABLE, PCANTP_CHANNEL_AVAILABLE, PCANTP_CHANNEL_OCCUPIED.

Default value: NA.

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

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

PCANTP_PARAM_WFT_MAX

Access: R W

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

Possible value: Any positive number.

Default value: PCANTP_WFT_MAX_DEFAULT (0x10).

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

PCANTP_PARAM_MSG_PENDING

Access: R W

Description: This parameter is used to define if the API should filter notifications of pending CAN-TP messages: fragmented CAN frames (either during reception and transmission) are notified by the API with a CAN-TP message with the type PCANTP_MESSAGE_INDICATION. If enabled, the function CANTP_Read will also return messages with this type.

Possible values: PCANTP_MSG_PENDING_HIDE enables message indication filtering, while PCANTP_MSG_PENDING_SHOW disables it.

Default value: PCANTP_MSG_PENDING_HIDE.

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

PCANTP_PARAM_API_VERSION

Access: R

Description: This parameter is used to get information about the PCAN-ISO-TP 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 PCANTP_NONEBUS channel.

PCANTP_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 the padding value.

Possible values: PCANTP_CAN_DATA_PADDING_NONE disables data padding (enabling CAN data optimization) and PCANTP_CAN_DATA_PADDING_ON enables data padding.

Default value: PCANTP_CAN_DATA_PADDING_ON since ECUs that do not support CAN data optimization may not respond to CAN-TP messages.

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

PCANTP_PARAM_RECEIVE_EVENT

Access: R W

Description: This parameter is used to let the PCAN-ISO-TP API notify an application when ISO-TP 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.

Default value: Disabled (0).

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

PCANTP_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 (PCANTP_CAN_DATA_PADDING_VALUE).

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

PCANTP_PARAM_J1939_PRIORITY

Access: R W

Description: This parameter is used to define the default priority for ISO-TP messages compliant with SAE J1939 data link layer (i.e. 29-bit CAN ID messages with normal fixed, mixed, or enhanced addressing).

Possible values: Any value from 0x00 to 0x07.

Default value: 0x06 (PCANTP_J1939_PRIORITY_DEFAULT).

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

PCANTP_PARAM_CAN_UNSEGMENTED

Access: R W

Description: This parameter is used to define how the API will handle non ISO-TP messages. The default behavior is PCANTP_CAN_UNSEGMENTED_OFF: all non ISO-TP CAN message will be discarded. If PCANTP_CAN_UNSEGMENTED_ON is set then non ISO-TP messages can be read with the ISO-TP read function.

Finally, if the parameter is set to PCANTP_CAN_UNSEGMENTED_ALL_FRAMES then the receive queue will contain ISO-TP messages, non ISO-TP messages, and all the segmented frames of an ISO-TP message, it is recommended to use this mode for debugging purpose only.

Possible values: PCANTP_CAN_UNSEGMENTED_OFF, PCANTP_CAN_UNSEGMENTED_ON, PCANTP_CAN_UNSEGMENTED_ALL_FRAMES.

Default value: 0x00 (PCANTP_CAN_UNSEGMENTED_OFF).

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

See also: Parameter Value Defintions on page 107.

PCANTP_PARAM_CAN_TX_DL

Access: R W

Description: This parameter is used to define the default Data Length Code (DLC) used when transmitting CAN FD messages: the fragmented frames composing the full CAN ISO-TP message will have a length corresponding to that DLC.

Possible values: 8 (0x08) to 15 (0x0F).

Default value: 8 (0x08).

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

See also: Parameter Value Defintions on page 107.

PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION

Access: R W

Description: This parameter is used to define the optimization mode of the Minimum Separation Time latency. The default is to use the optimized mode in order to speed up the transmission of ISO-TP messages, however some fragmented frames may be sent a few microseconds before the STMin value (experiment shows an average below 100µs, but on rare occasion it can be up to 1ms). If the ECU does not support STMin violation, set the value to 0.

Possible values: 0 or 1.

Default value: 1 (0x01).

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

See also: Parameter Value Defintions on page 107.

3.6 Methods

The methods defined for the classes CanTpApi (on page 10) and TCanTpApi (on page 11) are divided in 4 groups of functionality.

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

Connection

	Function	Description
S	Initialize	Initializes a PCANTP channel.
S	Uninitialize	Uninitializes a PCANTP channel.

Configuration

	Function	Description
S	SetValue	Sets a configuration or information value within a PCANTP channel.
S	AddMapping	Configures the ISO-TP mapping between a CAN ID and an ISO-TP network addressing information.
S	RemoveMapping	Removes a previously configured ISO-TP mapping.

Information

	Function	Description
	GetValue	Retrieves information from a PCANTP channel.
	GetStatus	Retrieves the current bus status of a PCANTP channel.
	GetErrorText	Gets a descriptive text for an error code.

Communication

	Function	Description
	Read	Reads a CAN message from the receive queue of a PCANTP channel.
	Write	Transmits a CAN message using a connected PCANTP channel.
	Reset	Resets the receive and transmit queues of a PCANTP channel.

3.6.1 Initialize

Initializes a PCANTP channel.

Overloads

	Function	Description
	Initialize(TPCANTPHandle, TPCANTPBaudrate)	Initializes a Plug and Play PCANTP channel.
	Initialize(TPCANTPHandle, TPCANTPBaudrate, TPCANTPHWType, UInt32, UInt16)	Initializes a Non-Plug-and Play PCANTP channel.

3.6.2 Initialize (TPCANTPHandle, TPCANTPBaudrate)

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

Syntax

Pascal OO

```
class function Initialize(
  CanChannel: TPCANTPHandle;
  Baudrate: TPCANTPBaudrate
  ): TPCANTPStatus; overload;
```

C#

```
public static TPCANTPStatus Initialize(
        [MarshalAs (UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        TPCANTPBaudrate Baudrate);
```

C++ / CLR

```
static TPCANTPStatus Initialize(
  [MarshalAs(UnmanagedType::U2)]
  TPCANTPHandle CanChannel,
  [MarshalAs(UnmanagedType::U2)]
  TPCANTPBaudrate Baudrate);
```

Visual Basic

```
Public Shared Function Initialize( _
   <MarshalAs(UnmanagedType.U2)> _
   ByVal CanChannel As TPCANTPHandle, _
   <MarshalAs(UnmanagedType.U2)> _
   ByVal Baudrate As TPCANTPBaudrate) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CANChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Baudrate	The speed for the communication (see TPCANTPBaudrate on page 19)

Returns

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

PCANTP_ERROR_ALREADY_INITIALIZED	Indicates that the desired PCANTP 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

The Initialize method initiates a PCANTP channel, preparing it for communication within the CAN bus connected to it. Calls to the other methods will fail, if they are used with a channel handle, different than PCANTP_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCANTP 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-Basic API, hence registering/connecting the Hardware denoted by the channel handle.
- To set-up the default values of the different parameters (see GetValue).

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

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

Example

The following example shows the initialize and uninitialize processes for a Plug and Play channel (channel 2 of a PCAN-PCI hardware). In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

C#

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.Initialize(CanTpApi.PCANTP_PCIBUS2,
TPCANTPBaudrate.PCANTP_BAUD_500K);
```

```
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released.
CanTpApi.Uninitialize(CanTpApi.PCANTP_NONEBUS);
```

C++/CLR

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi::Initialize(CanTpApi::PCANTP_PCIBUS2, PCANTP_BAUD_500K);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released.
CanTpApi::Uninitialize(CanTpApi::PCANTP_NONEBUS);
```

Visual Basic

```
Dim result As TPCANTPStatus

' The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.Initialize(CanTpApi.PCANTP_PCIBUS2,
TPCANTPBaudrate.PCANTP_BAUD_500K)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
End If

' All initialized channels are released.
CanTpApi.Uninitialize(CanTpApi.PCANTP_NONEBUS)
```

Pascal OO

```
var
  result: TPCANTPStatus;

begin
  // The Plug and Play channel (PCAN-PCI) is initialized.
  result := TCanTpApi.Initialize(TCanTpApi.PCANTP_PCIBUS2, PCANTP_BAUD_500K);
  if (result <> PCANTP_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.
  TCanTpApi.Uninitialize(TCanTpApi.PCANTP_NONEBUS);
end;
```

See also: InitializeFD on page 45, GetValue on page 67, Understanding PCAN-ISO-TP on page 6.

Plain function version: CANTP_Initialize on page 90.

3.6.3 Initialize (TPCANTPHandle, TPCANTPBaudrate, TPCANTPHWType, UInt32, UInt16)

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

Syntax

Pascal OO

```
class function Initialize(
   CanChannel: TPCANTPHandle;
   Baudrate: TPCANTPBaudrate;
   HwType: TPCANTPHWType;
   IOPort: LongWord;
   Interrupt: Word
   ): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Initialize")]
public static extern TPCANTPStatus Initialize(
   [MarshalAs(UnmanagedType.U2)]
   TPCANTPHandle CanChannel,
   [MarshalAs(UnmanagedType.U2)]
   TPCANTPBaudrate Baudrate,
   [MarshalAs(UnmanagedType.U1)]
   TPCANTPHWType HwType,
   UInt32 IOPort,
   UInt16 Interrupt);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Initialize")]
static TPCANTPStatus Initialize(
   [MarshalAs(UnmanagedType::U2)]
   TPCANTPHandle CanChannel,
   [MarshalAs(UnmanagedType::U2)]
   TPCANTPBaudrate Baudrate,
   [MarshalAs(UnmanagedType::U1)]
   TPCANTPHWType HwType,
   UInt32 IOPort,
   UInt16 Interrupt);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Initialize")> _
Public Shared Function Initialize( _
   <MarshalAs(UnmanagedType.U2)> _
   ByVal CanChannel As TPCANTPHandle, _
   <MarshalAs(UnmanagedType.U2)> _
   ByVal Baudrate As TPCANTPBaudrate, _
   <MarshalAs(UnmanagedType.U1)> _
   ByVal HwType As TPCANTPHWType, _
   ByVal IOPort As UInt32, _
   ByVal Interrupt As UInt16) As TPCANTPStatus
```

End Function

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Baudrate	The speed for the communication (see TPCANTPBaudrate on page 19)
HwType	The speed for the communication (see TPCANTPHWType on page 22)
IOPort	The I/O address for the parallel port
Interrupt	Interrupt number of the parallel port

Returns

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

PCANTP_ERROR_ALREADY_INITIALIZED	Indicates that the desired PCANTP 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

The Initialize method initiates a PCANTP channel, preparing it for communicate within the CAN bus connected to it. Calls to the other methods will fail, if they are used with a channel handle, different than PCANTP_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCANTP 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-Basic API, hence registering/connecting the hardware denoted by the channel handle.
- To set up the default values of the different parameters (see GetValue).

The Initialization process will fail, if an application tries to initialize a PCANTP channel that has already been initialized within the same process. Take into consideration, that initializing a channel causes a reset of the CAN hardware. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT, are removed.

Example

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

C#

```
TPCANTPStatus result;

// The Non-Plug and Play channel (PCAN-DNG) is initialized.
result = CanTpApi.Initialize(CanTpApi.PCANTP_DNGBUS1,
TPCANTPBaudrate.PCANTP_BAUD_500K, TPCANTPHWType.PCANTP_TYPE_DNG_SJA, 0x378, 7);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released.
CanTpApi.Uninitialize(CanTpApi.PCANTP_NONEBUS);
```

C++/CLR

```
TPCANTPStatus result;

// The non-Plug and Play channel (PCAN-DNG) is initialized.
result = CanTpApi::Initialize(CanTpApi::PCANTP_DNGBUS1, PCANTP_BAUD_500K,
PCANTP_TYPE_DNG_SJA, 0x378, 7);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released.
CanTpApi::Uninitialize(CanTpApi::PCANTP_NONEBUS);
```

Visual Basic

```
Dim result As TPCANTPStatus

' The non-Plug and Play channel (PCAN-DNG) is initialized.
result = CanTpApi.Initialize(CanTpApi.PCANTP_DNGBUS1,
TPCANTPBaudrate.PCANTP_BAUD_500K, TPCANTPHWType.PCANTP_TYPE_DNG_SJA, &H378, 7)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
End If

' All initialized channels are released.
CanTpApi.Uninitialize(CanTpApi.PCANTP_NONEBUS)
```

Pascal OO

```
var
  result: TPCANTPStatus;

begin
  // The non-Plug and Play channel (PCAN-DNG) is initialized.
  result := TCanTpApi.Initialize(TCanTpApi.PCANTP_DNGBUS1, PCANTP_BAUD_500K, PCANTP_TYPE_DNG_SJA,
$378, 7);
  if (result <> PCANTP_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.
  TCanTpApi.Uninitialize(TCanTpApi.PCANTP_NONEBUS);
end;
```

See also: CANTP_InitializeFD below, GetValue on page 67, Understanding PCAN-ISO-TP on page 6.

Plain function version: CANTP_Uninitialize on page 93.

3.6.4 InitializeFD

Initializes a FD capable PCANTP channel.

Syntax

Pascal OO

```
class function InitializeFD(
  CanChannel: TPCANTPHandle;
  Bitrate: TPCANTPBitrateFD
  ): TPCANTPStatus; overload;
```

C#

```
public static TPCANTPStatus InitializeFD(
        TPCANTPHandle CanChannel,
        TPCANTPBitrateFD Bitrate);
```

C++ / CLR

```
static TPCANTPStatus InitializeFD(
  [MarshalAs(UnmanagedType::U1)]
  TPCANTPHandle CanChannel,
  [MarshalAs(UnmanagedType::U2)]
  TPCANTPBitrateFD Bitrate);
```

Visual Basic

```
Public Shared Function InitializeFD( _
  <MarshalAs(UnmanagedType.U1)> _
  ByVal CanChannel As TPCANTPHandle, _
  <MarshalAs(UnmanagedType.U2)> _
  ByVal Bitrate As TPCANTPBitrateFD) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle)
Bitrate	The speed for the communication (see TPCANTPBitrateFD)

Returns

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

PCANTP_ERROR_ALREADY_INITIALIZED	Indicates that the desired PCANTP 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

The InitializeFD method initiates a FD capable PCANTP channel, preparing it for communication within the CAN bus connected to it. Calls to the other methods will fail, if they are used with a channel handle, different than PCANTP_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCANTP 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-Basic API, hence registering/connecting the Hardware denoted by the channel handle.

- To set up the default values of the different parameters (see GetValue on page 67).

The Initialization process will fail, if an application tries to initialize a PCANTP channel that has already been initialized within the same process. Take into consideration, that initializing a channel causes a reset of the CAN hardware. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT, are removed.

Example

The following example shows the initialize and uninitialize processes (with FD support) for a Plug and Play channel (channel 2 of a PCAN-USB hardware). In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French, or Spanish, and it will be shown to the user.

C#

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.InitializeFD(CanTpApi.PCANTP_PCIBUS2, "f_clock=80000000,
nom_brp=10, nom_tseg1=12, nom_tseg2=3, nom_sjw=1, data_brp=4, data_tseg1=7,
data_tseg2=2, data_sjw=1");
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released.
CanTpApi.Uninitialize(CanTpApi.PCANTP_NONEBUS);
```

C++/CLR

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi::InitializeFD(CanTpApi::PCANTP_PCIBUS2, "f_clock=80000000,
nom_brp=10, nom_tseg1=12, nom_tseg2=3, nom_sjw=1, data_brp=4, data_tseg1=7,
data_tseg2=2, data_sjw=1");
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");

// All initialized channels are released.
CanTpApi::Uninitialize(CanTpApi::PCANTP_NONEBUS);
```

Visual Basic

```
Dim result As TPCANTPStatus

' The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.InitializeFD(CanTpApi.PCANTP_PCIBUS2, "f_clock=80000000,
nom_brp=10, nom_tseg1=12, nom_tseg2=3, nom_sjw=1, data_brp=4, data_tseg1=7,
data_tseg2=2, data_sjw=1")
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
End If

' All initialized channels are released.
CanTpApi.Uninitialize(CanTpApi.PCANTP_NONEBUS)
```

Pascal OO

```
var
   result: TPCANTPStatus;

begin
   // The Plug and Play channel (PCAN-PCI) is initialized.
   result := TCanTpApi.InitializeFD(TCanTpApi.PCANTP_PCIBUS2, "f_clock=80000000, nom_brp=10, nom_tseg1=12, nom_tseg2=3, nom_sjw=1, data_brp=4, data_tseg1=7, data_tseg2=2, data_sjw=1");
   if (result <> PCANTP_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.
   TCanTpApi.Uninitialize(TCanTpApi.PCANTP_NONEBUS);
end;
```

See Also: Uninitialize below, GetValue on page 67, Understanding PCAN-ISO-TP on page 6, FD Bit Rate Parameter Definitions on page 108.

Plain function Version: CANTP_InitializeFD on page 92.

3.6.5 Uninitialize

Uninitializes a PCANTP channel.

Syntax

Pascal OO

```
class function Uninitialize(
   CanChannel: TPCANTPHandle
   ): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Uninitialize")]
public static TPCANTPStatus Uninitialize(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel);
```

C++ / CLR

```
static TPCANTPStatus Uninitialize (
   [MarshalAs(UnmanagedType::U2)]
   TPCANTPHandle CanChannel);
```

Visual Basic

```
Public Shared Function Uninitialize( _
   <MarshalAs(UnmanagedType.U2)> _
   ByVal CanChannel As TPCANTPHandle) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)

Returns

The return value is a TPCANTPStatus 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 PCAN channel can be released using one of these possibilities:

- **Single-Release:** Given a handle of a PCANTP 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 PCAN_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#

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.Initialize(CanTpApi.PCANTP_PCIBUS2,
TPCANTPBaudrate.PCANTP_BAUD_500K);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Initialization failed");
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized");

// Release channel.
CanTpApi.Uninitialize(CanTpApi.PCANTP_PCIBUS2);
```

C++/CLR

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi::Initialize(CanTpApi::PCANTP_PCIBUS2, PCANTP_BAUD_500K);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Initialization failed");
else
      MessageBox::Show("PCAN-PCI (Ch-2) was initialized");

// Release channel.
CanTpApi::Uninitialize(CanTpApi::PCANTP_PCIBUS2);
```

Visual Basic

```
Dim result As TPCANTPStatus

' The Plug and Play channel (PCAN-PCI) is initialized.
```

```
result = CanTpApi.Initialize(CanTpApi.PCANTP_PCIBUS2,
TPCANTPBaudrate.PCANTP_BAUD_500K)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Initialization failed")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized")
End If

' Release channel.
CanTpApi.Uninitialize(CanTpApi.PCANTP_PCIBUS2)
```

Pascal OO

```
var
  result: TPCANTPStatus;

begin
  // The Plug and Play channel (PCAN-PCI) is initialized.
  result := TCanTpApi.Initialize(TCanTpApi.PCANTP_PCIBUS2, PCANTP_BAUD_500K);
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Initialization failed', 'Error', MB_OK)
  else
    MessageBox(0, 'PCAN-PCI (Ch-2) was initialized', 'Error', MB_OK);

  // Release channel
  TCanTpApi.Uninitialize(TCanTpApi.PCANTP_PCIBUS2);
end;
```

See also: Initialize on page 40.

Plain function version: CANTP_Uninitialize on page 93.

3.6.6 SetValue

Sets a configuration or information value within a PCANTP channel.

Overloads

	Function	Description
	SetValue(TPCANTPHandle, TPCANTPParameter, UInt32, UInt32);	Sets a configuration or information numeric value within a PCANTP channel.
	SetValue(TPCANTPHandle, TPCANTPParameter, String, UInt32);	Sets a configuration or information string value within a PCANTP channel.
	SetValue(TPCANTPHandle, TPCANTPParameter, Byte[], UInt32);	Sets a configuration or information with an array of bytes within a PCANTP channel.

3.6.7 SetValue (TPCANTPHandle, TPCANTPParameter, UInt32, Uint32)

Sets a configuration or information numeric value within a PCANTP channel.

Syntax

Pascal OO

```
class function SetValue(
        CanChannel: TPCANTPHandle;
        Parameter: TPCANTPParameter;
        NumericBuffer: PLongWord;
        BufferLength: LongWord
        ): TPCANTPStatus; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue")]
public static extern TPCANTPStatus SetValue(
  [MarshalAs(UnmanagedType.U2)]
  TPCANTPHandle CanChannel,
  [MarshalAs(UnmanagedType.U1)]
  TPCANTPParameter Parameter,
  ref UInt32 NumericBuffer,
  UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue")]
static TPCANTPStatus SetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPParameter Parameter,
        UInt32% NumericBuffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPCANTPParameter, _
        ByRef NumericBuffer As UInt32, _
        ByVal BufferLength As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to be set (see TPCANTPParameter on page 33)
NumericBuffer	The buffer containing the numeric value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is aTPCANTPStatus 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.
PCANTP_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 PCANTP channel.

Note: Any calls with non ISO-TP parameters (ie. TPCANTPParameter) will be forwarded to PCAN-Basic API.

More information about the parameters and values that can be set can be found in Parameter Value Defintions. Since most of the ISO-TP 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 PCANTP_PCIBUS2 to enable debug mode.

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

C#

```
TPCANTPStatus result;
UInt32 iBuffer = 0;

// Enables CAN DEBUG mode.
iBuffer = CanTpApi.PCANTP_DEBUG_CAN;
result = CanTpApi.SetValue(CanTpApi.PCANTP_PCIBUS2,
TPCANTPParameter.PCANTP_PARAM_DEBUG, ref iBuffer, sizeof(UInt32));
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to set value");
else
    MessageBox.Show("Value changed successfully ");
```

C++/CLR

```
TPCANTPStatus result;
UInt32 iBuffer = 0;

// Enables CAN DEBUG mode.
iBuffer = CanTpApi::PCANTP_DEBUG_CAN;
result = CanTpApi::SetValue(CanTpApi::PCANTP_PCIBUS2, PCANTP_PARAM_DEBUG, iBuffer,
sizeof(UInt32));
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to set value");
else
      MessageBox::Show("Value changed successfully ");
```

Visual Basic

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

' Enables CAN DEBUG mode.
iBuffer = CanTpApi.PCANTP_DEBUG_CAN
result = CanTpApi.SetValue(CanTpApi.PCANTP_PCIBUS2,
TPCANTPParameter.PCANTP_PARAM_DEBUG, iBuffer, Convert.ToUInt32(Len(iBuffer)))
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
```

```
    MessageBox.Show("Failed to set value")
Else
    MessageBox.Show("Value changed successfully ")
End If
```

Pascal OO

```
var
  result: TPCANTPStatus;
  iBuffer: UINT;

begin
  // Enables CAN DEBUG mode.
  iBuffer := TCanTpApi.PCANTP_DEBUG_CAN;
  result := TCanTpApi.SetValue(TCanTpApi.PCANTP_PCIBUS2, PCANTP_PARAM_DEBUG,
        PLongWord(@iBuffer), sizeof(iBuffer));
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Failed to set value', 'Error', MB_OK)
  else
    MessageBox(0, 'Value changed successfully ', 'Error', MB_OK);
end;
```

See also: GetValue on page 67, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_GetValue on page 99.

3.6.8 SetValue (TPCANTPHandle, TPCANTPParameter, StringBuffer, Uint32)

Sets a configuration or information string value within a PCANTP channel.

Syntax

Pascal OO

```
class function SetValue(
        CanChannel: TPCANTPHandle;
        Parameter: TPCANTPParameter;
        StringBuffer: PAnsiChar;
        BufferLength: LongWord
        ): TPCANTPStatus; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue")]
public static extern TPCANTPStatus SetValue(
  [MarshalAs(UnmanagedType.U2)]
  TPCANTPHandle CanChannel,
  [MarshalAs(UnmanagedType.U1)]
  TPCANTPParameter Parameter,
  [MarshalAs(UnmanagedType.LPStr, SizeParamIndex = 3)]
  string StringBuffer,
  UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue")]
```

```
static TPCANTPStatus SetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPParameter Parameter,
        [MarshalAs(UnmanagedType::LPStr, SizeParamIndex = 3)]
        String^ StringBuffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPCANTPParameter, _
        <MarshalAs(UnmanagedType.LPStr, SizeParamIndex:=3)> _
        ByVal StringBuffer As String, _
        ByVal BufferLength As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to be set (see TPCANTPParameter on page 33).
NumericBuffer	The buffer containing the string value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is aTPCANTPStatus 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.
PCANTP_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

This override is only defined for users who wishes to configure PCAN-Basic API through the ISO-TP API.

See also: GetValue on page 67, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_SetValue on page 94.

3.6.9 SetValue (TPCANTPHandle, TPCANTPParameter, byte(), Uint32)

Sets a configuration or information value as a byte array within a PCANTP channel.

Syntax

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue")]
public static extern TPCANTPStatus SetValue(
```

```
[MarshalAs(UnmanagedType.U2)]
TPCANTPHandle CanChannel,
[MarshalAs(UnmanagedType.U1)]
TPCANTPParameter Parameter,
[MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
        Byte[] Buffer,
UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue")]
static TPCANTPStatus SetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPParameter Parameter,
        [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 3)]
        array<Byte>^ Buffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_SetValue")> _
Public Shared Function SetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPCANTPParameter, _
        ByVal Buffer As Byte(), _
        ByVal BufferLength As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16).
Parameter	The code of the value to be set (see TPCANTPParameter on page 33).
NumericBuffer	The buffer containing the array value to be set.
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:

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.
PCANTP_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 PCANTP channel.

Note: Any calls with non ISO-TP parameters (ie. TPCANTPParameter) will be forwarded to PCAN-Basic API.

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

Example

The following example shows the use of the method SetValue on the channel PCANTP_USBBUS1 to define the use of unlimited block size during ISO-TP communication.

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

C#

```
TPCANTPStatus result;

// Defines unlimited blocksize.
result = CanTpApi.SetValue(CanTpApi.PCANTP_USBBUS1,
TPCANTPParameter.PCANTP_PARAM_BLOCK_SIZE, new byte[] { 0 }, 1);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to set value");
else
    MessageBox.Show("Value changed successfully ");
```

C++/CLR

```
TPCANTPStatus result;

// Defines unlimited blocksize.
result = CanTpApi::SetValue(CanTpApi::PCANTP_USBBUS1, PCANTP_PARAM_BLOCK_SIZE,
gcnew array<Byte> { 0 }, 1);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to set value");
else
      MessageBox::Show("Value changed successfully ");
```

Visual Basic

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

' Defines unlimited blocksize.
bufferArray(0) = 0
result = CanTpApi.SetValue(CanTpApi.PCANTP_USBBUS1,
TPCANTPParameter.PCANTP_PARAM_BLOCK_SIZE, bufferArray,
Convert.ToUInt32(bufferArray.Length))
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to set value")
Else
    MessageBox.Show("Value changed successfully ")
End If
```

Pascal OO

```
var
  result: TPCANTPStatus;
  bufferArray: array[0..2] of Byte;
```

```
begin
  // Defines unlimited blocksize.
  bufferArray[0] := 0;
  result := TCanTpApi.SetValue(TCanTpApi.PCANTP_USBBUS1, PCANTP_PARAM_BLOCK_SIZE,
PLongWord(@bufferArray), Length(bufferArray));
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Failed to set value', 'Error', MB_OK)
  else
    MessageBox(0, 'Value changed successfully ', 'Error', MB_OK);
end;
```

See also: GetValue on page 67, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_SetValue on page 94.

3.6.10 GetErrorText(TPCANStatus, UInt16, StringBuilder)

Gets a descriptive text for an error code.

Syntax

Pascal OO

```
class function GetErrorText(
        Error: TPCANTPStatus;
        Language: Word;
        StringBuffer: PAnsiChar
        ): TPCANTPStatus;
```

C#

```
[DllImport ("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetErrorText")]
public static extern TPCANTPStatus GetErrorText(
        [MarshalAs(UnmanagedType.U4)]
        TPCANTPStatus Error,
        UInt16 Language,
        StringBuilder StringBuffer);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetErrorText")]
static TPCANTPStatus GetErrorText(
        [MarshalAs(UnmanagedType::U4)]
        TPCANTPStatus Error,
        UInt16 Language,
        StringBuilder^ StringBuffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetErrorText")> _
Public Shared Function GetErrorText( _
        <MarshalAs(UnmanagedType.U4)> _
        ByVal anError As TPCANTPStatus, _
        ByVal Language As UInt16, _
```

```
        ByVal StringBuffer As StringBuilder) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
Error	A TPCANTPStatus error code
Language	Indicates a "Primary Language ID".
StringBuffer	A buffer for a null-terminated char array.

Returns

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

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

Remarks

The Primary Language IDs are codes used by Windows OS from Microsoft, to identify a human language. The PCAN-Basic API currently supports the following languages:

Language	Primary Language ID
Neutral (System dependant)	00h (0)
English	09h (9)
German	07h (7)
French	0Ch (12)
Italian	10h (16)
Spanish	0Ah (10)

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

Example

The following example shows the use of the method GetErrorText to get the description of an error. The language of the description's text will be the same used by the operating system (if its language is supported; otherwise English is used).

C#

```
TPCANTPStatus result;
StringBuilder strMsg;

strMsg = new StringBuilder(256);

// Gets the description text for PCANTP_ERROR_ALREADY_INITIALIZED using the Neutral
language.
result = CanTpApi.GetErrorText(
      TPCANTPStatus.PCANTP_ERROR_ALREADY_INITIALIZED, 0, strMsg);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
{
      // An error occurred.
      MessageBox.Show("An error occured");
}
else
```

```
      MessageBox.Show(strMsg.ToString());
```

C++ / CLR

```
TPCANTPStatus result;
StringBuilder^ strMsg;

strMsg = gcnew StringBuilder(256);

// Gets the description text for PCANTP_ERROR_ALREADY_INITIALIZED using the Neutral
language.
result = CanTpApi::GetErrorText(PCANTP_ERROR_ALREADY_INITIALIZED, 0, strMsg);
if (result != PCANTP_ERROR_OK)
{
      // An error occurred.
      MessageBox::Show("An error occured");
}
else
      MessageBox::Show(strMsg->ToString());
```

Visual Basic

```
Dim result As TPCANTPStatus
Dim strMsg As StringBuilder

strMsg = New StringBuilder(256)

' Gets the description text for PCANTP_ERROR_ALREADY_INITIALIZED using the Neutral
language.
result = CanTpApi.GetErrorText( _
      TPCANTPStatus.PCANTP_ERROR_ALREADY_INITIALIZED, 0, strMsg)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
      ' An error occurred.
      MessageBox.Show("An error occured")
Else
      MessageBox.Show(strMsg.ToString())
End If
```

Pascal OO

```
var
  result: TPCANTPStatus;
  strMsg: array [0..256] of Char;

begin
  // Gets the description text for PCANTP_ERROR_ALREADY_INITIALIZED using the Neutral language.
  result := TCanTpApi.GetErrorText(PCANTP_ERROR_ALREADY_INITIALIZED, 0, strMsg);
  if (result <> PCANTP_ERROR_OK) then
    // An error occurred.
    MessageBox(0, 'An error occured', 'Error', MB_OK)
  else
    MessageBox(0, strMsg, 'Success', MB_OK);
end;
```

See also: Primary Language ID

3.6.11 AddMapping

Adds a mapping between a CAN ID and an ISO-TP network addressing information within a PCANTP channel.

Syntax

Pascal OO

```
class function AddMapping(
        CanChannel: TPCANTPHandle;
        canID: LongWord;
        canIDResponse: LongWord;
        canIdType: TPCANTPIdType;
        formatType: TPCANTPFormatType;
        msgType: TPCANTPMessageType ;
        sourceAddr: Byte;
        targetAddr: Byte;
        targetType: TPCANTPAddressingType;
        remoteAddr: Byte
): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_AddMapping")]
public static extern TPCANTPStatus AddMapping(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        uint canID,
        uint canIDResponse,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPIdType canIdType,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPFormatType formatType,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPMessageType msgType,
        byte sourceAddr,
        byte targetAddr,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPAddressingType targetType,
        byte remoteAddr);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_AddMapping")]
static TPCANTPStatus AddMapping(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        UInt32 canID,
        UInt32 canIDResponse,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPIdType canIdType,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPFormatType formatType,
        [MarshalAs(UnmanagedType::U1)]
```

```
        TPCANTPMessageType msgType,
        Byte sourceAddr,
        Byte targetAddr,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPAddressingType targetType,
        Byte remoteAddr);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_AddMapping")> _
Public Shared Function AddMapping( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        ByVal canID As UInt32, _
        ByVal canIDResponse As UInt32, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal canIdType As TPCANTPIdType, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal formatType As TPCANTPFormatType, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal msgType As TPCANTPMessageType, _
        ByVal sourceAddr As Byte, _
        ByVal targetAddr As Byte, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal targetType As TPCANTPAddressingType, _
        ByVal remoteAddr As Byte) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
canID	The CAN Identifier to be mapped.
canIdReponse	The CAN Identifier that is used to respond to a request with the CAN Id 'canId'.
canIdType	Type of CAN identifier
formatType	Format type of the ISO-TP network addressing information.
msgType	ISO-TP message type
sourceAddr	Source address
targetAddr	Target address
targetType	Type of the target
remoteAddr	Remote address

Returns

The return value is aTPCANTPStatus 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.
PCANTP_ERROR_ALREADY_INITIALIZED	A mapping with the same CAN ID already exists.
PCANTP_ERROR_WRONG_PARAM	Mapping is not valid in regards to ISO-TP standard.
PCANTP_ERROR_NO_MEMORY	Failed to allocate memory to define mapping.

Remarks

The following table summarizes requirements to get a valid mapping based on the addressing format type:

FormatType parameter	Valid canIdType parameter	Valid msgType parameter	Valid targetType parameter	Valid remoteAddr parameter
PCANTP_FORMAT_NORMAL	Any	PCANTP_MESSAGE_DIAGNOSTIC	Any values	0x00 (value is ignored)
PCANTP_FORMAT_EXTENDED	Any	Any	Any	Any
PCANTP_FORMAT_MIXED	PCANTP_ID_CAN_11BIT	PCANTP_MESSAGE_REMOTE_DIAGNOSTIC	Any	Any

When target type is functional addressing there is no need to define a CAN ID response, since responses from functional adressing will be physically adressed. The definition value CAN_ID_NO_MAPPING can be used to fill in the canIdResponse parameter in those cases.

Note: The formats PCANTP_FORMAT_FIXED_NORMAL and PCANTP_FORMAT_ENHANCED requires a 29-bit CAN ID and do not need mappings to be defined, see ISO-TP Network Addressing for more information.

Example

The following example defines two CAN ID mappings in order to receive and transmit ISO-TP messages using 11-bit CAN Identifiers with “MIXED” format addressing.

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

C#

```
TPCANTPHandle CanChannel = CanTpApi.PCANTP_USBBUS1;
TPCANTPStatus result;
byte canId = 0xD1;
byte canIdResponse = 0xD2;
byte N_SA = 0xF1;
byte N_TA = 0x13;
byte N_RA = 0x52;

// Defines a first mapping to allow communication from Source 0xF1 to Destination
0x13.
result = CanTpApi.AddMapping(CanChannel, canId, canIdResponse,
TPCANTPIdType.PCANTP_ID_CAN_11BIT, TPCANTPFormatType.PCANTP_FORMAT_MIXED,
TPCANTPMessageType.PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_SA, N_TA,
TPCANTPAddressingType.PCANTP_ADDRESSING_PHYSICAL, N_RA);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to add first mapping.");
// Defines a second mapping to allow communication from Destination 0x13 to Source
0xF1.
result = CanTpApi.AddMapping(CanChannel, canIdResponse, canId,
TPCANTPIdType.PCANTP_ID_CAN_11BIT, TPCANTPFormatType.PCANTP_FORMAT_MIXED,
TPCANTPMessageType.PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_TA, N_SA,
TPCANTPAddressingType.PCANTP_ADDRESSING_PHYSICAL, N_RA);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to add second mapping.");
```

C++ / CLR

```
TPCANTPHandle CanChannel = CanTpApi::PCANTP_USBBUS1;
TPCANTPStatus result;
```

```
Byte canId = 0xD1;
Byte canIdResponse = 0xD2;
Byte N_SA = 0xF1;
Byte N_TA = 0x13;
Byte N_RA = 0x52;

// Defines a first mapping to allow communication from Source 0xF1 to Destination
0x13.
result = CanTpApi::AddMapping(CanChannel, canId, canIdResponse,
PCANTP_ID_CAN_11BIT, PCANTP_FORMAT_MIXED, PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_SA,
N_TA, PCANTP_ADDRESSING_PHYSICAL, N_RA);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to add first mapping.");
// Defines a second mapping to allow communication from Destination 0x13 to Source
0xF1.
result = CanTpApi::AddMapping(CanChannel, canIdResponse, canId,
PCANTP_ID_CAN_11BIT, PCANTP_FORMAT_MIXED, PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_TA,
N_SA, PCANTP_ADDRESSING_PHYSICAL, N_RA);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to add second mapping.");
```

Visual Basic

```
Dim CanChannel As TPCANTPHandle = CanTpApi.PCANTP_USBBUS1
Dim result As TPCANTPStatus
Dim canId As Byte = &HD1
Dim canIdResponse As Byte = &HD2
Dim N_SA As Byte = &HF1
Dim N_TA As Byte = &H13
Dim N_RA As Byte = &H52

' Defines a first mapping to allow communication from Source &HF1 to Destination
&H13.
result = CanTpApi.AddMapping(CanChannel, canId, canIdResponse,
TPCANTPIdType.PCANTP_ID_CAN_11BIT, TPCANTPFormatType.PCANTP_FORMAT_MIXED,
TPCANTPMessageType.PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_SA, N_TA,
TPCANTPAddressingType.PCANTP_ADDRESSING_PHYSICAL, N_RA)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to add first mapping.")
End If
' Defines a second mapping to allow communication from Destination &H13 to Source
&HF1.
result = CanTpApi.AddMapping(CanChannel, canIdResponse, canId,
TPCANTPIdType.PCANTP_ID_CAN_11BIT, TPCANTPFormatType.PCANTP_FORMAT_MIXED,
TPCANTPMessageType.PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_TA, N_SA,
TPCANTPAddressingType.PCANTP_ADDRESSING_PHYSICAL, N_RA)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to add second mapping.")
End If
```

Pascal OO

```
var
  CanChannel: TPCANTPHandle;
  result: TPCANTPStatus;
  canId: Byte;
  canIdResponse: Byte;
  N_SA: Byte;
  N_TA: Byte;
```

```
  N_RA: Byte;

begin
  CanChannel := TCanTpApi.PCANTP_USBBUS1;
  canId := $D1;
  canIdResponse := $D2;
  N_SA := $F1;
  N_TA := $13;
  N_RA := $52;

  // Defines a first mapping to allow communication from Source $F1 to Destination $13.
  result := TCanTpApi.AddMapping(CanChannel, canId, canIdResponse,
    PCANTP_ID_CAN_11BIT, PCANTP_FORMAT_MIXED, PCANTP_MESSAGE_REMOTE_DIAGNOSTIC,
    N_SA, N_TA, PCANTP_ADDRESSING_PHYSICAL, N_RA);
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Failed to add first mapping.', 'Error', MB_OK);
  // Defines a second mapping to allow communication from Destination $13 to Source $F1.
  result := TCanTpApi.AddMapping(CanChannel, canIdResponse, canId,
    PCANTP_ID_CAN_11BIT, PCANTP_FORMAT_MIXED, PCANTP_MESSAGE_REMOTE_DIAGNOSTIC,
    N_TA, N_SA, PCANTP_ADDRESSING_PHYSICAL, N_RA);
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Failed to add second mapping.', 'Error', MB_OK);
```

See also: RemoveMapping below.

Plain function version: CANTP_Remove Mapping on page 98.

3.6.12 RemoveMapping

Removes a previously defined CAN ID mapping.

Syntax

Pascal OO

```
class function RemoveMapping(
        CanChannel: TPCANTPHandle;
        canID: LongWord
): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveMapping")]
public static extern TPCANTPStatus RemoveMapping(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        uint canID);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveMapping")]
static TPCANTPStatus RemoveMapping(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
```

```
        UInt32 canID);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_RemoveMapping")> _
Public Shared Function RemoveMapping( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        ByVal canID As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
canID	The CAN Identifier that identifies the mapping to remove.

Returns

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

PCANTP_ERROR_NOT_INITIALIZED	Indicates that the given PCANTP channel cannot be uninitialized or the mapping was not found.

Example

The following example shows the definition and removal of a CAN ID mapping used for functional addressing with NORMAL addressing.

C#

```
TPCANTPHandle CanChannel = CanTpApi.PCANTP_USBBUS1;
TPCANTPStatus result;
byte canId = 0xD1;
byte N_SA = 0xF1;
byte N_TA = 0x30;
byte N_RA = 0x00;

// Adds a mapping to transmit functionally addressed messages.
result = CanTpApi.AddMapping(CanChannel,
canId, CanTpApi.CAN_ID_NO_MAPPING, TPCANTPIdType.PCANTP_ID_CAN_11BIT,
TPCANTPFormatType.PCANTP_FORMAT_NORMAL,
TPCANTPMessageType.PCANTP_MESSAGE_DIAGNOSTIC,
N_SA, N_TA, TPCANTPAddressingType.PCANTP_ADDRESSING_FUNCTIONAL, N_RA);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to add mapping.");

...

// Removes the mapping.
result = CanTpApi.RemoveMapping(CanChannel, canId);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to remove mapping.");
```

C++ / CLR

```
TPCANTPHandle CanChannel = CanTpApi::PCANTP_USBBUS1;
TPCANTPStatus result;
Byte canId = 0xD1;
```

```
Byte N_SA = 0xF1;
Byte N_TA = 0x30;
Byte N_RA = 0x00;

// Adds a mapping to transmit functionally addressed messages.
result = CanTpApi::AddMapping(CanChannel,
      canId, CanTpApi::CAN_ID_NO_MAPPING, PCANTP_ID_CAN_11BIT,
      PCANTP_FORMAT_NORMAL,
      PCANTP_MESSAGE_DIAGNOSTIC,
      N_SA, N_TA, PCANTP_ADDRESSING_FUNCTIONAL, N_RA);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to add mapping.");

// ...

// Removes the mapping.
result = CanTpApi::RemoveMapping(CanChannel, canId);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to add second mapping.");
```

Visual Basic

```
Dim CanChannel As TPCANTPHandle = CanTpApi.PCANTP_USBBUS1
Dim result As TPCANTPStatus
Dim canId As Byte = &HD1
Dim N_SA As Byte = &HF1
Dim N_TA As Byte = &H30
Dim N_RA As Byte = &H0

' Adds a mapping to transmit functionally addressed messages.
result = CanTpApi.AddMapping(CanChannel, _
    canId, CanTpApi.CAN_ID_NO_MAPPING, TPCANTPIdType.PCANTP_ID_CAN_11BIT, _
    TPCANTPFormatType.PCANTP_FORMAT_NORMAL, _
    TPCANTPMessageType.PCANTP_MESSAGE_DIAGNOSTIC, _
    N_SA, N_TA, TPCANTPAddressingType.PCANTP_ADDRESSING_FUNCTIONAL, N_RA)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to add mapping.")
End If

' ...

' Removes the mapping.
result = CanTpApi.RemoveMapping(CanChannel, canId)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to add second mapping.")
End If
```

Pascal OO

```
var
  CanChannel: TPCANTPHandle;
  result: TPCANTPStatus;
  canId: Byte;
  N_SA: Byte;
  N_TA: Byte;
  N_RA: Byte;

begin
  CanChannel := TCanTpApi.PCANTP_USBBUS1;
```

```
  canId := $D1;
  N_SA := $F1;
  N_TA := $30;
  N_RA := $00;

  // Adds a mapping to transmit functionally addressed messages.
  result := TCanTpApi.AddMapping(CanChannel,
    canId, TCanTpApi.CAN_ID_NO_MAPPING, PCANTP_ID_CAN_11BIT,
    PCANTP_FORMAT_NORMAL,
    PCANTP_MESSAGE_DIAGNOSTIC,
    N_SA, N_TA, PCANTP_ADDRESSING_FUNCTIONAL, N_RA);
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Failed to add mapping.', 'Error', MB_OK);

  // ...

  // Removes the mapping.
  result := TCanTpApi.RemoveMapping(CanChannel, canId);
  if (result <> PCANTP_ERROR_OK) then
    MessageBox(0, 'Failed to add second mapping.', 'Error', MB_OK);
end;
```

See also: AddMapping on page 60.

Plain function version: CANTP_Remove Mapping on page 98.

3.6.13 GetValue

Retrieves information from a PCANTP channel.

Overloads

	Function	Description
	GetValue(TPCANTPHandle, TPCANTPParameter, UInt32, UInt32);	Retrieves information from a PCANTP channel in numeric form.
	GetValue(TPCANTPHandle, TPCANTPParameter, String, UInt32);	Retrieves information from a PCANTP channel in text form.
	GetValue(TPCANTPHandle, TPCANTPParameter, Byte[], UInt32)	Retrieves information from a PCANTP channel in byte array form.

3.6.14 GetValue (TPCANTPHandle, TPCANTPParameter, StringBuilder, Uint32)

Retrieves information from a PCANTP channel in text form.

Syntax

Pascal OO

```
class function GetValue(
        CanChannel: TPCANTPHandle;
        Parameter: TPCANTPParameter;
        StringBuffer: PAnsiChar;
        BufferLength: LongWord
): TPCANTPStatus; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue")]
public static extern TPCANTPStatus GetValue(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPParameter Parameter,
        StringBuilder StringBuffer,
        UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue")]
static TPCANTPStatus GetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPParameter Parameter,
        StringBuilder^ StringBuffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetValue")> _
Public Shared Function GetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPCANTPParameter, _
        ByVal StringBuffer As StringBuilder, _
        ByVal BufferLength As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to retrieve (see TPCANTPParameter on page 33)
StringBuffer	The buffer to return the required string value.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is aTPCANTPStatus 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 was not found in the list of initialized channels of the calling application.
PCANTP_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#

```
TPCANTPStatus result;
StringBuilder BufferString;

// Get API version.
BufferString = new StringBuilder(255);
result = CanTpApi.GetValue(CanTpApi.PCANTP_NONEBUS,
TPCANTPParameter.PCANTP_PARAM_API_VERSION, BufferString, 255);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to get value");
else
    MessageBox.Show(BufferString.ToString());
```

C++ / CLR

```
TPCANTPStatus result;
StringBuilder^ BufferString;

// Get API version.
BufferString = gcnew StringBuilder(255);
result = CanTpApi::GetValue(CanTpApi::PCANTP_NONEBUS,
      PCANTP_PARAM_API_VERSION, BufferString, 255);
if (result != PCANTP_ERROR_OK)
      MessageBox::Show("Failed to get value");
else
      MessageBox::Show(BufferString->ToString());
```

Visual Basic

```
Dim result As TPCANTPStatus
Dim BufferString As StringBuilder

' Get API version.
BufferString = New StringBuilder(255)
result = CanTpApi.GetValue(CanTpApi.PCANTP_NONEBUS, _
    TPCANTPParameter.PCANTP_PARAM_API_VERSION, BufferString, 255)
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(BufferString.ToString())
End If
```

Pascal OO

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

begin

  // Get API version.
  result := TCanTpApi.GetValue(TCanTpApi.PCANTP_NONEBUS, PCANTP_PARAM_API_VERSION, BufferString,
255);
  if (result <> PCANTP_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 50, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_GetValue on page 99.

3.6.15 GetValue (TPCANTPHandle, TPCANTPParameter, UInt32, UInt32)

Retrieves information from a PCAN channel in numeric form.

Syntax

Pascal OO

```
class function GetValue(
        CanChannel: TPCANTPHandle;
        Parameter: TPCANTPParameter;
        NumericBuffer: PLongWord;
        BufferLength: LongWord
): TPCANTPStatus; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue")]
public static extern TPCANTPStatus GetValue(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPParameter Parameter,
        out UInt32 NumericBuffer,
        UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue")]
static TPCANTPStatus GetValue(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPParameter Parameter,
        UInt32% NumericBuffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetValue")> _
Public Shared Function GetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPCANTPParameter, _
        ByRef NumericBuffer As UInt32, _
        ByVal BufferLength As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to retrieve (see TPCANTPParameter on page 33)
NumericBuffer	The buffer to return the required numeric value.
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:

PCANTP_ERROR_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_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 PCANTP_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 = CanTpApi.PCANTP_USBBUS1;
TPCANTPStatus result;
UInt32 iBuffer = 0;

// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue(CanChannel,
TPCANTPParameter.PCANTP_PARAM_SEPERATION_TIME,
out iBuffer, sizeof(UInt32));
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to get value");
else
    MessageBox.Show(iBuffer.ToString());
```

C++ / CLR

```
TPCANTPHandle CanChannel = CanTpApi::PCANTP_USBBUS1;
TPCANTPStatus result;
UInt32 iBuffer = 0;

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

Visual Basic

```
Dim CanChannel As TPCANTPHandle = CanTpApi.PCANTP_USBBUS1
Dim result As TPCANTPStatus
Dim iBuffer As UInt32 = 0
```

```
' Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue(CanChannel,
TPCANTPParameter.PCANTP_PARAM_SEPERATION_TIME, _
    iBuffer, Convert.ToUInt32(Len(iBuffer)))
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(iBuffer.ToString())
End If
```

Pascal OO

```
var
  CanChannel: TPCANTPHandle;
  result: TPCANTPStatus;
  iBuffer: UINT;

begin
  CanChannel := TCanTpApi.PCANTP_USBBUS1;
  // Gets the value of the ISO-TP Separation Time (STmin) parameter
  result := TCanTpApi.GetValue(CanChannel, PCANTP_PARAM_SEPERATION_TIME, PLongWord(@iBuffer),
sizeof(iBuffer));
  if (result <> PCANTP_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 50, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_GetValue on page 99.

3.6.16 GetValue (TPCANTPHandle, TPCANTPParameter, byte(), UInt32)

Retrieves information from a PCAN channel in a byte array.

Syntax

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue")]
public static extern TPCANTPStatus GetValue(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType.U1)]
        TPCANTPParameter Parameter,
        [MarshalAs(UnmanagedType.LPArray)]
        [Out] Byte[] Buffer,
        UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue")]
static TPCANTPStatus GetValue(
        [MarshalAs(UnmanagedType::U2)]
```

```
        TPCANTPHandle CanChannel,
        [MarshalAs(UnmanagedType::U1)]
        TPCANTPParameter Parameter,
        [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 3)]
        array<Byte>^ Buffer,
        UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetValue")> _
Public Shared Function GetValue( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Parameter As TPCANTPParameter, _
        ByVal Buffer As Byte(), _
        ByVal BufferLength As UInt32) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to be set (see TPCANTPParameter on page 33)
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:

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.
PCANTP _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 PCANTP_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 = CanTpApi.PCANTP_USBBUS1;
TPCANTPStatus result;
uint bufferLength = 2;
byte[] bufferArray = new byte[bufferLength];

// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue(CanChannel,
TPCANTPParameter.PCANTP_PARAM_SEPERATION_TIME,
bufferArray, sizeof(byte) * bufferLength);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
    MessageBox.Show("Failed to get value");
```

```
else
    MessageBox.Show(bufferArray[0].ToString());
```

C++ / CLR

```
TPCANTPHandle CanChannel = CanTpApi::PCANTP_USBBUS1;
TPCANTPStatus result;
UInt32 bufferLength = 2;
array<Byte>^ bufferArray = gcnew array<Byte>(bufferLength);

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

Visual Basic

```
Dim CanChannel As TPCANTPHandle = CanTpApi.PCANTP_USBBUS1
Dim result As TPCANTPStatus
Dim bufferLength As UInt32 = 2
Dim bufferArray(bufferLength) As Byte

' Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue(CanChannel,
TPCANTPParameter.PCANTP_PARAM_SEPERATION_TIME, _
    bufferArray, Convert.ToUInt32(bufferLength))
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(bufferArray(0).ToString())
End If
```

Pascal OO

```
var
  CanChannel: TPCANTPHandle;
  result: TPCANTPStatus;
  bufferArray: array [0..1] of Byte;

begin
  CanChannel := TCanTpApi.PCANTP_USBBUS1;

  // Gets the value of the ISO-TP Separation Time (STmin) parameter.
  result := TCanTpApi.GetValue(CanChannel, PCANTP_PARAM_SEPERATION_TIME, PLongWord(@bufferArray),
Length(bufferArray));
  if (result <> PCANTP_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 50, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_GetValue on page 99.

3.6.17 GetStatus

Gets the current BUS status of a PCANTP channel.

Syntax

Pascal OO

```
class function GetStatus(
CanChannel: TPCANTPHandle
): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetStatus")]
public static extern TPCANTPStatus GetStatus(
[MarshalAs(UnmanagedType.U2)]
TPCANTPHandle CanChannel);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetStatus")]
static TPCANTPStatus GetStatus(
[MarshalAs(UnmanagedType::U2)]
TPCANTPHandle CanChannel);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetStatus")> _
Public Shared Function GetStatus( _
<MarshalAs(UnmanagedType.U2)> _
ByVal CanChannel As TPCANTPHandle) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)

Returns

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

PCANTP_ERROR_OK	Indicates that the status of the given PCANTP channel is OK.
PCANTP_ERROR_BUSLIGHT	Indicates a bus error within the given PCANTP channel. The hardware is in bus-light status.
PCANTP_ERROR_BUSHEAVY:	Indicates a bus error within the given PCANTP channel. The hardware is in bus-heavy status.
PCANTP_ERROR_BUSOFF:	Indicates a bus error within the given PCANTP channel. The hardware is in bus-off status.
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

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 PCANTP_PCIBUS1. 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;

// Checks the status of the PCI channel.
result = CanTpApi.GetStatus(CanTpApi.PCANTP_PCIBUS1);
switch (result)
{
    case TPCANTPStatus.PCANTP_ERROR_BUSLIGHT:
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...");
        break;
    case TPCANTPStatus.PCANTP_ERROR_BUSHEAVY:
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...");
        break;
    case TPCANTPStatus.PCANTP_ERROR_BUSOFF:
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...");
        break;
    case TPCANTPStatus.PCANTP_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

```
TPCANTPStatus result;

// Checks the status of the PCI channel.
result = CanTpApi::GetStatus(CanTpApi::PCANTP_PCIBUS1);
switch (result)
{
case PCANTP_ERROR_BUSLIGHT:
      MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...");
      break;
case PCANTP_ERROR_BUSHEAVY:
      MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...");
      break;
case PCANTP_ERROR_BUSOFF:
      MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...");
      break;
case PCANTP_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 TPCANTPStatus

' Checks the status of the PCI channel.
result = CanTpApi.GetStatus(CanTpApi.PCANTP_PCIBUS1)
Select Case (result)
    Case TPCANTPStatus.PCANTP_ERROR_BUSLIGHT
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...")
    Case TPCANTPStatus.PCANTP_ERROR_BUSHEAVY
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...")
    Case TPCANTPStatus.PCANTP_ERROR_BUSOFF
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...")
    Case TPCANTPStatus.PCANTP_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: TPCANTPStatus;

begin

  // Checks the status of the PCI channel.
  result := TCanTpApi.GetStatus(TCanTpApi.PCANTP_PCIBUS1);
  Case (result) of
    PCANTP_ERROR_BUSLIGHT:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...', 'Error', MB_OK);
    PCANTP_ERROR_BUSHEAVY:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...', 'Error', MB_OK);
    PCANTP_ERROR_BUSOFF:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-OFF status...', 'Error', MB_OK);
    PCANTP_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: TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Plain function version: CANTP_GetStatus on page 100.

3.6.18 Read

Reads a CAN ISO-TP message from the receive queue of a PCANTP channel.

Overloads

	Function	Description
	Read(TPCANTPHandle, TPCANTPMsg)	Reads a CAN ISO-TP message from the receive queue of a PCANTP channel.

	Function	Description
	Read(TPCANTPHandle, TPCANTPMsg, TPCANTPTimestamp)	Reads a CAN ISO-TP message and its timestamp from the receive queue of a PCANTP channel.

3.6.19 Read(TPCANTPHandle, TPCANTPMsg)

Reads a CAN ISO-TP message from the receive queue of a PCANTP channel.

Syntax

Pascal OO

```
class function Read(
        CanChannel: TPCANTPHandle;
        var MessageBuffer: TPCANTPMsg
): TPCANTPStatus;overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Read")]
public static extern TPCANTPStatus Read(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        out TPCANTPMsg MessageBuffer);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Read")]
static TPCANTPStatus Read(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        TPCANTPMsg %MessageBuffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Read")> _
Public Shared Function Read( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        ByRef MessageBuffer As TPCANTPMsg) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Message Buffer	A TPCANTPMsg buffer to store the CAN ISO-TP message.

Returns

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

PCANTP_ERROR_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
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

The message type (see TPCANTPMessageType) of a CAN ISO-TP message indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic), 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.

If the time when the message was received is needed, use the overloaded Read(TPCANTPHandle, TPCANTPMsg, TPCANTPTimestamp) method.

Example

The following example shows the use of the method "Read" on the channel PCANTP_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
{
    // Reads the first message in the queue.
    result = CanTpApi.Read(CanTpApi.PCANTP_USBBUS1, out msg);
    if (result == TPCANTPStatus.PCANTP_ERROR_OK)
    {
        // Processes 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

```
TPCANTPStatus result;
TPCANTPMsg msg;
bool bStop = false;

do
{
      // Reads the first message in the queue.
      result = CanTpApi::Read(CanTpApi::PCANTP_USBBUS1, msg);
      if (result == PCANTP_ERROR_OK)
      {
            // Processes 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 TPCANTPStatus
Dim msg As TPCANTPMsg
Dim bStop As Boolean = False

Do
    ' Reads the first message in the queue.
    msg = New TPCANTPMsg()
    result = CanTpApi.Read(CanTpApi.PCANTP_USBBUS1, msg)
    If result = TPCANTPStatus.PCANTP_ERROR_OK Then
        ' Processes 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: TPCANTPStatus;
  msg: TPCANTPMsg;
  bStop: Boolean;

begin
  bStop := False;
  repeat
    // Reads the first message in the queue.
    result := TCanTpApi.Read(TCanTpApi.PCANTP_USBBUS1, msg);
    if (result = PCANTP_ERROR_OK) then
    begin
      // Processes 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 on page 84.

Plain function version: CANTP_Read on page 101.

3.6.20 Read(TPCANTPHandle, TPCANTPMsg, TPCANTPTimestamp)

Reads a CAN ISO-TP message and its timestamp from the receive queue of a PCANTP channel.

Syntax

Pascal OO

```
class function Read(
        CanChannel: TPCANTPHandle;
        var MessageBuffer: TPCANTPMsg;
        TimestampBuffer: PTPCANTPTimestamp
): TPCANTPStatus; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Read")]
public static extern TPCANTPStatus Read(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        out TPCANTPMsg MessageBuffer
        out TPCANTPTimestamp TimestampBuffer);
```

C++/CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Read")]
static TPCANTPStatus Read(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        TPCANTPMsg %MessageBuffer,
        TPCANTPTimestamp % TimestampBuffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Read")> _
Public Shared Function Read( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        ByRef MessageBuffer As TPCANTPMsg, _
        ByRef TimestampBuffer As TPCANTPTimestamp) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Message Buffer	A TPCANTPMsg buffer to store the CAN ISO-TP message.
TimestampBuffer	A TPCANTimestamp buffer to get the reception time of the message.

Returns

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

PCANTP_ERROR_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
PCANTP_ERROR_NOT_INITIALIZED	Indicates that the given PCANTP channel cannot be used because it was not found in the list of reserved channels of the calling application.

Remarks

The message type (see TPCANTPMessageType) of a CAN ISO-TP message indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic), 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.

Example

The following example shows the use of the method “Read” on the channel PCANTP_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;
TPCANTPTimestamp ts;
bool bStop = false;

do
{
    // Reads the first message in the queue.
    result = CanTpApi.Read(CanTpApi.PCANTP_USBBUS1, out msg, out ts);
    if (result == TPCANTPStatus.PCANTP_ERROR_OK)
    {
        // Processes 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

```
TPCANTPStatus result;
TPCANTPMsg msg;
TPCANTPTimestamp ts;
bool bStop = false;

do
{
      // Reads the first message in the queue.
      result = CanTpApi::Read(CanTpApi::PCANTP_USBBUS1, msg, ts);
      if (result == PCANTP_ERROR_OK)
      {
            // Processes 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 TPCANTPStatus
Dim msg As TPCANTPMsg
Dim ts As TPCANTPTimestamp
Dim bStop As Boolean = False

Do
    ' Reads the first message in the queue.
    msg = New TPCANTPMsg()
    result = CanTpApi.Read(CanTpApi.PCANTP_USBBUS1, msg, ts)
    If result = TPCANTPStatus.PCANTP_ERROR_OK Then
        ' Processes 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: TPCANTPStatus;
  msg: TPCANTPMsg;
  ts: TPCANTPTimestamp;
  bStop: Boolean;

begin
  bStop := False;
```

```
  repeat
    // Reads the first message in the queue.
    result := TCanTpApi.Read(TCanTpApi.PCANTP_USBBUS1, msg, PTPCANTPTimestamp(@ts));
    if (result = PCANTP_ERROR_OK) then
    begin
      // Processes 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: CANTP_Read on page 101.

3.6.21 Write

Transmits a CAN ISO-TP message.

Syntax

Pascal OO

```
class function Write(
        CanChannel: TPCANTPHandle;
        var MessageBuffer: TPCANTPMsg
): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Write")]
public static extern TPCANTPStatus Write(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel,
        ref TPCANTPMsg MessageBuffer);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Write")]
static TPCANTPStatus Write(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel,
        TPCANTPMsg %MessageBuffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Write")> _
Public Shared Function Write( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle, _
        ByRef MessageBuffer As TPCANTPMsg) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Message Buffer	A TPCANTPMsg buffer containing the CAN ISO-TP message to be sent.

Returns

The return value is a TPCANTPStatus 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 was not found in the list of initialized channels of the calling application or that a required CAN ID mapping was not found.
PCANTP_ERROR_WRONG_PARAM	The network addressing information of the message is not valid.
PCANTP_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue.

Remarks

The “Write” function do not actually send the ISO-TP 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 TPCANTPMsg.

Example

The following example shows the use of the method Write on the channel PCANTP_USBBUS1. It 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 was configured.

C#

```
// Prepare an 11-bit CAN ID, physically addressed using extended format and
extended addressing ISO-TP message containing 4095 bytes of data.
TPCANTPMsg request = new TPCANTPMsg();
request.DATA = new byte[4095];
request.LEN = (ushort)request.DATA.Length;
request.MSGTYPE = TPCANTPMessageType.PCANTP_MESSAGE_DIAGNOSTIC;
request.IDTYPE = TPCANTPIdType.PCANTP_ID_CAN_11BIT;
request.FORMAT = TPCANTPFormatType.PCANTP_FORMAT_EXTENDED;

request.SA = 0xf1;
request.TA = 0x7e;
request.TA_TYPE = TPCANTPAddressingType.PCANTP_ADDRESSING_PHYSICAL;
request.RA = 0x00;

// The message is sent using the PCAN-USB.
TPCANTPStatus result = CanTpApi.Write(CanTpApi.PCANTP_USBBUS1, ref request);
if (result == TPCANTPStatus.PCANTP_ERROR_OK)
{
    // Loop until the transmission confirmation is received.
    do
```

```
    {
        result = CanTpApi.Read(CanTpApi.PCANTP_USBBUS1, out request);
        MessageBox.Show(String.Format("Read = {0}, type={1}, result={2}", result,
request.MSGTYPE, request.RESULT));
    } while (result == TPCANTPStatus.PCANTP_ERROR_NO_MESSAGE);
}
else
{
    // An error occurred.
    MessageBox.Show("Error occured: " + result.ToString());
}
```

C++ / CLR

```
TPCANTPStatus result;
// Prepare an 11-bit CAN ID, physically addressed using extended format and
extended addressing ISO-TP message containing 4095 bytes of data.
TPCANTPMsg^ request = gcnew TPCANTPMsg();
request->DATA = gcnew array<Byte>(4095);
request->LEN = (unsigned short)request->DATA->Length;
request->MSGTYPE = PCANTP_MESSAGE_DIAGNOSTIC;
request->IDTYPE = PCANTP_ID_CAN_11BIT;
request->FORMAT = PCANTP_FORMAT_EXTENDED;
request->SA = 0xf1;
request->TA = 0x7e;
request->TA_TYPE = PCANTP_ADDRESSING_PHYSICAL;
request->RA = 0x00;

// The message is sent using the PCAN-USB.
result = CanTpApi::Write(CanTpApi::PCANTP_USBBUS1, *request);
if (result == PCANTP_ERROR_OK)
{
      // Loop until the transmission confirmation is received.
      do
      {
            result = CanTpApi::Read(CanTpApi::PCANTP_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 == PCANTP_ERROR_NO_MESSAGE);
}
else
{
      // An error occurred.
      MessageBox::Show(String::Format("Error occured: {0}", (int)result));
}
```

Visual Basic

```
' Prepare an 11-bit CAN ID, physically addressed using extended format and extended
addressing ISO-TP message containing 4095 bytes of data.
Dim request As TPCANTPMsg = New TPCANTPMsg()
request.DATA = New Byte(4095) {}
request.LEN = request.DATA.Length
request.MSGTYPE = TPCANTPMessageType.PCANTP_MESSAGE_DIAGNOSTIC
request.IDTYPE = TPCANTPIdType.PCANTP_ID_CAN_11BIT
request.FORMAT = TPCANTPFormatType.PCANTP_FORMAT_EXTENDED

request.SA = &HF1
request.TA = &H7E
request.TA_TYPE = TPCANTPAddressingType.PCANTP_ADDRESSING_PHYSICAL
```

```
request.RA = &H0

' The message is sent using the PCAN-USB.
Dim result As TPCANTPStatus = CanTpApi.Write(CanTpApi.PCANTP_USBBUS1, request)
If result = TPCANTPStatus.PCANTP_ERROR_OK Then
    ' Loop until the transmission confirmation is received.
    Do
        result = CanTpApi.Read(CanTpApi.PCANTP_USBBUS1, request)
        MessageBox.Show(String.Format("Read = {0}, type={1}, result={2}", result,
request.MSGTYPE, request.RESULT))
    Loop While result = TPCANTPStatus.PCANTP_ERROR_NO_MESSAGE
Else
    ' An error occurred.
    MessageBox.Show("Error occured: " + result.ToString())
End If
```

Pascal OO

```
var
  // Prepares an 11-bit CAN ID, physically addressed using extended format and extended addressing ISO-TP
message containing 4095 bytes of data-
  request: TPCANTPMsg;
  result: TPCANTPStatus;

begin
  request.LEN := Length(request.DATA);
  request.MSGTYPE := PCANTP_MESSAGE_DIAGNOSTIC;
  request.IDTYPE := PCANTP_ID_CAN_11BIT;
  request.FORMAT := PCANTP_FORMAT_EXTENDED;

  request.SA := $f1;
  request.TA := $7e;
  request.TA_TYPE := PCANTP_ADDRESSING_PHYSICAL;
  request.RA := $00;

  // The message is sent using the PCAN-USB.
  result := TCanTpApi.Write(TCanTpApi.PCANTP_USBBUS1, request);
  if (result = PCANTP_ERROR_OK) then
  begin
    // Loop until the transmission confirmation is received.
    repeat
      result := TCanTpApi.Read(TCanTpApi.PCANTP_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 = PCANTP_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 77.

Plain function version: CANTP_Read on page 101.

3.6.22 Reset

Resets the receive and transmit queues of a PCANTP channel.

Syntax

Pascal OO

```
class function Reset(
        CanChannel: TPCANTPHandle
): TPCANTPStatus;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Reset")]
public static extern TPCANTPStatus Reset(
        [MarshalAs(UnmanagedType.U2)]
        TPCANTPHandle CanChannel);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Reset")]
static TPCANTPStatus Reset(
        [MarshalAs(UnmanagedType::U2)]
        TPCANTPHandle CanChannel);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Reset")> _
Public Shared Function Reset( _
        <MarshalAs(UnmanagedType.U2)> _
        ByVal CanChannel As TPCANTPHandle) As TPCANTPStatus
End Function
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)

Returns

The return value is a TPCANTPStatus 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 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 PCANTP_PCIBUS1. 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;

// The PCI channel is reset.
result = CanTpApi.Reset(CanTpApi.PCANTP_PCIBUS1);
if (result != TPCANTPStatus.PCANTP_ERROR_OK)
{
    // An error occurred.
    MessageBox.Show("An error occured");
}
else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset");
```

C++ / CLR

```
TPCANTPStatus result;

// The PCI channel is reset.
result = CanTpApi::Reset(CanTpApi::PCANTP_PCIBUS1);
if (result != PCANTP_ERROR_OK)
{
      // An error occurred.
      MessageBox::Show("An error occured");
}
else
      MessageBox::Show("PCAN-PCI (Ch-1) was reset");
```

Visual Basic

```
Dim result As TPCANTPStatus

' The PCI channel is reset.
result = CanTpApi.Reset(CanTpApi.PCANTP_PCIBUS1)
If result <> TPCANTPStatus.PCANTP_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: TPCANTPStatus;

begin
  // The PCI channel is reset.
  result := TCanTpApi.Reset(TCanTpApi.PCANTP_PCIBUS1);
  if (result <> PCANTP_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 48.

Plain function version: CANTP_Reset on page 104.

3.7 Functions

The functions of the PCAN ISO-TP API are divided in 4 groups of functionality.

Connection

	Function	Description
	CANTP_Initialize	Initializes a PCANTP channel.
	CANTP_Uninitialize	Uninitializes a PCANTP channel.

Configuration

	Function	Description
	CANTP_SetValue	Sets a configuration or information value within a PCANTP channel.
	CANTP_AddMapping	Configures the ISO-TP mapping between a CAN ID and an ISO-TP network addressing information.
	CANTP_RemoveMapping	Removes a previously configured ISO-TP mapping.

Information

	Function	Description
	CANTP_GetValue	Retrieves information from a PCANTP channel.
	CANTP_GetStatus	Retrieves the current BUS status of a PCANTP channel.
	CANTP_GetErrorText	Gets a descriptive text for an error code.

Communication

	Function	Description
	CANTP_Read	Reads a CAN message from the receive queue of a PCANTP channel.
	CANTP_Write	Transmits a CAN message using a connected PCANTP channel.
	CANTP_Reset	Resets the receive and transmit queues of a PCANTP channel.

3.7.1 CANTP_Initialize

Initializes a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_Initialize(
        TPCANTPHandle channel,
        TPCANTPBaudrate Baudrate,
        TPCANTPHWType HwType = 0,
        DWORD IOPort = 0,
        WORD Interrupt = 0);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Baudrate	The speed for the communication (see TPCANTPBaudrate on page 19)
HwType	The type of hardware (TPCANTPHWType)
IOPort	The I/O address for the parallel port.
Interrupt	Interrupt number of the parallel port.

Returns

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

PCANTP_ERROR_ALREADY_INITIALIZED	Indicates that the desired PCANTP 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

The “initialize” method initiates a PCANTP channel, preparing it for communicate within the CAN bus connected to it. Calls to the other methods will fail, if they are used with a channel handle, different than PCANTP_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCANTP 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-Basic API, hence registering/connecting the Hardware denoted by the channel handle.
- To set-up the default values of the different parameters (see CANTP_SetValue).

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

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

The PCAN-ISO-TP API uses the same function for initializations of both, Plug and Play, and non-Plug and Play hardware. The CANTP_Initialize function has three additional parameters that are only for the connection of Non-Plug and Play hardware. With Plug and Play hardware, however, only two parameters are to be supplied. The remaining three are not evaluated.

Example

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

C++

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CANTP_Initialize(PCANTP_PCIBUS2, PCANTP_BAUD_500K);
if (result != PCANTP_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
CANTP_Uninitialize(PCANTP_NONEBUS);
```

See also: CANTP_Uninitialize on page 93, CANTP_GetValue on page 99, Understanding PCAN-ISO-TP on page 6.

Class method version: Initialize on page 40.

3.7.2 CANTP_InitializeFD

Initializes a FD capable PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_InitializeFD(
        TPCANTPHandle channel,
        TPCANTPBitrateFD BitrateFD
);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
BitrateFD	The speed for the communication (see TPCANTPBitrateFD on page 21)

Returns

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

PCANTP_ERROR_ALREADY_INITIALIZED	Indicates that the desired PCANTP 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

The InitializeFD method initiates a FD capable PCANTP channel, preparing it for communicate within the CAN bus connected to it. Calls to the other methods will fail, if they are used with a channel handle, different than PCANTP_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCANTP 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-Basic API, hence registering/connecting the Hardware denoted by the channel handle.
- To set up the default values of the different parameters (see CANTP_SetValue).

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

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

Example

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

C++

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-USB) is initialized @500kbps/2Mbps.
result = CANTP_InitializeFD(PCANTP_USBBUS2, "f_clock=80000000, nom_brp=10,
nom_tseg1=12, nom_tseg2=3, nom_sjw=1, data_brp=4, data_tseg1=7, data_tseg2=2,
data_sjw=1");
if (result != PCANTP_ERROR_OK)
```

```
      MessageBox(NULL, "Initialization failed", "Error", MB_OK);
else
      MessageBox(NULL, "PCAN-USB (Ch-2) was initialized", "Success", MB_OK);

// All initialized channels are released.
CANTP_Uninitialize(PCANTP_NONEBUS);
```

See also: CANTP_Uninitialize below, CANTP_GetValue on page 99, Understanding PCAN-ISO-TP on page 6, FD Bit Rate Parameter Definitions, FD Bit Rate Parameter Definitions on page 108.

Class method version: InitializeFD on page 45.

3.7.3 CANTP_Uninitialize

Uninitializes a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_Uninitialize(
        TPCANTPHandle CanChannel);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)

Returns

The return value is a TPCANTPStatus 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 PCAN channel can be released using one of these possibilities:

- **Single-Release**: Given a handle of a PCANTP 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 PCAN_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++

```
TPCANTPStatus result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CANTP_Initialize(PCANTP_PCIBUS2, PCANTP_BAUD_500K);
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Initialization failed", "Error", MB_OK);
else
      MessageBox(NULL, "PCAN-PCI (Ch-2) was initialized", "Success", MB_OK);
```

```
// Release channel
CANTP_Uninitialize(PCANTP_PCIBUS2);
```

See also: CANTP_Initialize on page 90.

Class method version: Uninitialize on page 48.

3.7.4 CANTP_SetValue

Sets a configuration or information value within a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_SetValue(
        TPCANTPHandle CanChannel,
        TPCANTPParameter Parameter,
        void* Buffer,
            DWORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to be set (see TPCANTPParameter on page 33)
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 TPCANTPStatus 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.
PCANTP_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 PCANTP channel.

Note: Any calls with non ISO-TP parameters (i.e. TPCANTPParameter) will be forwarded to PCAN-Basic API.

More information about the parameters and values can be found in Parameter Value Defintions. Since most of the ISO-TP 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 function CANTP_SetValue on the channel PCANTP_PCIBUS2 to enable debug mode.

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

C++

```
TPCANTPStatus result;
unsigned int iBuffer = 0;

// Enable CAN DEBUG mode.
iBuffer = PCANTP_DEBUG_CAN;
result     =     CANTP_SetValue(PCANTP_PCIBUS2,     PCANTP_PARAM_DEBUG,     &iBuffer,
sizeof(unsigned int));
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Failed to set value", "Error", MB_OK);
else
      MessageBox(NULL, "Value changed successfully ", "Success", MB_OK);
```

See also: CANTP_GetValue on page 99, TPCANTPParameter on page 33.

Class method version: GetValue on page 67.

3.7.5 CANTP_GetErrorText

Gets a descriptive text for an error code.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_GetErrorText(
    TPCANTPStatus Error,
    WORD Language,
    LPSTR Buffer);
```

Parameters

Parameters	Description
Error	A TPCANTPStatus error code)
Language	Indicates a "Primary Language ID".
StringBuffer	A buffer for a null-terminated char array

Returns

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

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

Remarks

The Primary Language IDs are codes used by Windows OS from Microsoft, to identify a human language. The PCAN-Basic API currently supports the following languages:

Language	Primary Language ID
Neutral (System dependant)	00h (0)
English	09h (9)
German	07h (7)
French	0Ch (12)
Italian	10h (16)
Spanish	0Ah (10)

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

Example

The following example shows the use of the method GetErrorText to get the description of an error. The language of the description's text will be the same used by the operating system (if its language is supported; otherwise English is used).

C++

```
TPCANTPStatus result;
char strMsg[256];

// Gets the description text for PCANTP_ERROR_ALREADY_INITIALIZED using the Neutral
language.
result = CANTP_GetErrorText(PCANTP_ERROR_ALREADY_INITIALIZED, 0, strMsg);
if (result != PCANTP_ERROR_OK)
{
      // An error occurred.
      MessageBox(NULL, "An error occured", "Error", MB_OK);
}
else
      MessageBox(NULL, strMsg, "Success", MB_OK);
```

See also: Primary Language ID

3.7.6 CANTP_AddMapping

Adds a mapping between a CAN ID and an ISO-TP network addressing information within a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_AddMapping(
      TPCANTPHandle CanChannel,
      DWORD canID,
      DWORD canIDResponse,
      TPCANTPIdType canIdType,
      TPCANTPFormatType formatType,
      TPCANTPMessageType msgType,
      BYTE sourceAddr,
      BYTE targetAddr,
      TPCANTPAddressingType targetType,
      BYTE remoteAddr);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
canID	The CAN Identifier to be mapped.
canIdReponse	The CAN Identifier that is used to respond to a request with the CAN Id 'canId'.
canIdType	The type of CAN identifier
formatType	The format type of the ISO-TP network addressing information
msgType	The ISO-TP message type
sourceAddr	The source address

Parameters	Description
targetAddr	The target address
targetType	The type of the target
remoteAddr	The remote address

Returns

The return value is aTPCANTPStatus 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.
PCANTP_ERROR_ALREADY_INITIALIZED	A mapping with the same CAN ID already exists.
PCANTP_ERROR_WRONG_PARAM	Mapping is not valid in regards to ISO-TP standard.
PCANTP_ERROR_NO_MEMORY	Failed to allocate memory to define mapping.

Remarks

The following table summarizes requirements to get a valid mapping based on the addressing format type.

FormatType parameter	Valid canIdType parameter	Valid msgType parameter	Valid targetType parameter	Valid remoteAddr parameter
PCANTP_FORMAT_NORMAL	Any	PCANTP_MESSAGE_DIAGNOSTIC	Any values	0x00 (value is ignored)
PCANTP_FORMAT_EXTENDED	Any	Any	Any	Any
PCANTP_FORMAT_MIXED	PCANTP_ID_CAN_11BIT	PCANTP_MESSAGE_REMOTE_DIAGNOSTIC	Any	Any

When target type is functional addressing there is no need to define a CAN ID response, since responses from functional addressing will be physically addressed. The definition value CAN_ID_NO_MAPPING can be used to fill in the canIdResponse parameter in those cases.

Note: The formats PCANTP_FORMAT_FIXED_NORMAL and PCANTP_FORMAT_ENHANCED require a 29-bit CAN ID and do not need mappings to be defined, see ISO-TP Network Addressing for more information.

Example

The following example defines two CAN ID mappings in order to receive and transmit ISO-TP messages using 11-bit CAN Identifiers with “MIXED” format addressing.

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

C++

```
TPCANTPHandle CanChannel = PCANTP_USBBUS1;
TPCANTPStatus result;
BYTE canId = 0xD1;
BYTE canIdResponse = 0xD2;
BYTE N_SA = 0xF1;
BYTE N_TA = 0x13;
BYTE N_RA = 0x52;

// Defines a first mapping to allow communication from Source 0xF1 to Destination
0x13.
result = CANTP_AddMapping(CanChannel, canId, canIdResponse, PCANTP_ID_CAN_11BIT,
PCANTP_FORMAT_MIXED, PCANTP_MESSAGE_REMOTE_DIAGNOSTIC, N_SA, N_TA,
PCANTP_ADDRESSING_PHYSICAL, N_RA);
```

```
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Failed to add first mapping.", "Error", MB_OK);
// Defines a second mapping to allow communication from Destination 0x13 to Source
0xF1.
result = CANTP_AddMapping(CanChannel, canIdResponse, canId, PCANTP_ID_CAN_11BIT,
PCANTP_FORMAT_MIXED,       PCANTP_MESSAGE_REMOTE_DIAGNOSTIC,       N_TA,       N_SA,
PCANTP_ADDRESSING_PHYSICAL, N_RA);
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Failed to add second mapping.", "Error", MB_OK);
```

See also: CANTP_Remove Mapping below.

Class method version: RemoveMapping on page 64.

3.7.7 CANTP_Remove Mapping

Removes a previously defined CAN ID mapping.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_RemoveMapping(
      TPCANTPHandle CanChannel,
      DWORD canID);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
canID	The CAN Identifier that identifies the mapping to remove.

Returns

The return value is a TPCANTPStatus 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 or the mapping was not found.

Example

The following example shows the definition and removal of a CAN ID mapping used for functional addressing with NORMAL addressing.

C++

```
TPCANTPHandle CanChannel = PCANTP_USBBUS1;
TPCANTPStatus result;
BYTE canId = 0xD1;
BYTE N_SA = 0xF1;
BYTE N_TA = 0x30;
BYTE N_RA = 0x00;

// Adds a mapping to transmit functionally addressed messages.
result = CANTP_AddMapping(CanChannel,
      canId, CAN_ID_NO_MAPPING, PCANTP_ID_CAN_11BIT,
      PCANTP_FORMAT_NORMAL,
      PCANTP_MESSAGE_DIAGNOSTIC,
      N_SA, N_TA, PCANTP_ADDRESSING_FUNCTIONAL, N_RA);
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Failed to add mapping.", "Error", MB_OK);
```

```
// ...

// Removes the mapping.
result = CANTP_RemoveMapping(CanChannel, canId);
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Failed to add second mapping.", "Error", MB_OK);
```

See also: Primary Language ID **See also**: Primary Language ID

CANTP_AddMapping on page 96.

Class method version: RemoveMapping on page 64.

3.7.8 CANTP_GetValue

Retrieves information from a PCAN channel in numeric form.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_GetValue(
        TPCANTPHandle CanChannel,
        TPCANTPParameter Parameter,
        void* Buffer,
        DWORD BufferLength);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to retrieve (see TPCANTPParameter on page 33)
Buffer	The buffer to return the required numeric value.
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:

PCANTP_ERROR_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_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 function CANTP_GetValue on the channel PCANTP_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 = PCANTP_USBBUS1;
TPCANTPStatus result;
unsigned int iBuffer = 0;
char strMsg[256];
```

```
// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result   =   CANTP_GetValue(CanChannel,   PCANTP_PARAM_SEPERATION_TIME,   &iBuffer,
sizeof(unsigned int));
if (result != PCANTP_ERROR_OK)
      MessageBox(NULL, "Failed to get value", "Error", MB_OK);
else
{
      sprintf(strMsg, "%d", iBuffer);
      MessageBox(NULL, strMsg, "Success", MB_OK);
}
```

See also: CANTP_SetValue on page 94, TPCANTPParameter on page 33, Parameter Value Defintions on page 107.

Class method version: GetValue on page 67.

3.7.9 CANTP_GetStatus

Gets the current BUS status of a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_GetStatus(
        TPCANTPHandle CanChannel);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)

Returns

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

PCANTP_ERROR_OK	Indicates that the status of the given PCANTP channel is OK.
PCANTP_ERROR_BUSLIGHT	Indicates a bus error within the given PCANTP channel. The hardware is in bus-light status.
PCANTP_ERROR_BUSHEAVY	Indicates a bus error within the given PCANTP channel. The hardware is in bus-heavy status.
PCANTP_ERROR_BUSOFF	Indicates a bus error within the given PCANTP channel. The hardware is in bus-off status.
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

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 function CANTP_GetStatus on the channel PCANTP_PCIBUS1. 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;

// Checks the status of the PCI channel.
result = CANTP_GetStatus(PCANTP_PCIBUS1);
switch (result)
{
case PCANTP_ERROR_BUSLIGHT:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...",
"Success", MB_OK);
      break;
case PCANTP_ERROR_BUSHEAVY:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...",
"Success", MB_OK);
      break;
case PCANTP_ERROR_BUSOFF:
      MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-OFF status...", "Success",
MB_OK);
      break;
case PCANTP_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:

TPCANTPParameter on page 33, class method version: GetStatus on page 75.

Parameter Value Defintions on page 107.

3.7.10 CANTP_Read

Reads a CAN ISO-TP message from the receive queue of a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_Read(
      TPCANTPHandle CanChannel,
      TPCANTPMsg* MessageBuffer,
      TPCANTPTimestamp* TimestampBuffer _DEF_ARG);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Message Buffer	A TPCANTPMsg buffer to store the CAN ISO-TP message.
TimestampBuffer	A TPCANTimestamp buffer to get the reception time of the message.

Returns

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

PCANTP_ERROR_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
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

The message type (see TPCANTPMessageType) of a CAN ISO-TP message indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic), 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.

Specifying the value of "NULL" for the parameter TimestampBuffer causes reading a message without timestamp, when the reception time is not desired.

Example

The following example shows the use of the function CANTP_Read on the channel PCANTP_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;
TPCANTPTimestamp ts;
bool bStop = false;

do
{
	// Reads the first message in the queue.
	result = CANTP_Read(PCANTP_USBBUS1, &msg);
	if (result == PCANTP_ERROR_OK)
	{
		// Processes 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: CANTP_Write on page 103, class method version: Read on page 77.

3.7.11 CANTP_Write

Transmits a CAN ISO-TP message.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_Write(
        TPCANTPHandle CanChannel,
        TPCANTPMsg* MessageBuffer);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
MessageBuffer	A TPCANTPMsg buffer containing the CAN ISO-TP message to be sent.

Returns

The return value is a TPCANTPStatus 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 was not found in the list of initialized channels of the calling application or that a required CAN ID mapping was not found.
PCANTP_ERROR_WRONG_PARAM	The network addressing information of the message is not valid.
PCANTP_ERROR_NO_MEMORY	Failed to allocate memory and copy message in the transmission queue.

Remarks

The Write function do not actually send the ISO-TP 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 TPCANTPMsg.

Example

The following example shows the use of the function CANTP_Write on the channel PCANTP_USBBUS1. It 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 was configured.

C++

```
TPCANTPStatus result;
// Prepare an 11-bit CAN ID, physically addressed using extended format and
extended addressing ISO-TP message containing 4095 bytes of data.
TPCANTPMsg request;
char strMsg[256];

request.LEN = 4095;
request.MSGTYPE = PCANTP_MESSAGE_DIAGNOSTIC;
request.IDTYPE = PCANTP_ID_CAN_11BIT;
request.FORMAT = PCANTP_FORMAT_EXTENDED;

request.SA = 0xf1;
request.TA = 0x7e;
request.TA_TYPE = PCANTP_ADDRESSING_PHYSICAL;
request.RA = 0x00;

// The message is sent using the PCAN-USB.
result = CANTP_Write(PCANTP_USBBUS1, &request);
```

```
if (result == PCANTP_ERROR_OK)
{
      // Loop until the transmission confirmation is received.
      do
      {
            result = CANTP_Read(PCANTP_USBBUS1, &request);
            sprintf(strMsg,    "Read    =    %d,    type=%d,    result=%d",    result,
request.MSGTYPE, request.RESULT);
            MessageBox(NULL, strMsg, "Error", MB_OK);
      } while (result == PCANTP_ERROR_NO_MESSAGE);
}
else
{
      // An error occurred.
      sprintf(strMsg, "Error occured: %d", result);
      MessageBox(NULL, strMsg, "Error", MB_OK);
}
```

See also: CANTP_Read on page 101, class method version: Read on page 77.

3.7.12 CANTP_Reset

Resets the receive and transmit queues of a PCANTP channel.

Syntax

C++

```
TPCANTPStatus __stdcall CANTP_Reset(
        TPCANTPHandle CanChannel);
```

Parameters

Parameters	Description
CanChannel	The handle of a PCANTP channel (see TPCANTPHandle on page 16)
Parameter	The code of the value to be set (see TPCANTPParameter on page 33)
NumericBuffer	The buffer containing the string value to be set.
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:

PCANTP_ERROR_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.

Remarks

This method 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 function CANTP_Reset on the channel PCANTP_PCIBUS1. 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;

// The PCI channel is reset.
result = CANTP_Reset(PCANTP_PCIBUS1);
if (result != PCANTP_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: CANTP_Uninitialize on page 93, class-method: Reset on page 88.

3.8 Definitions

The PCAN-Basic API defines the following values:

Name	Description
PCAN-ISO-TP Handle Definitions	Defines the handles for the different PCAN channels.
Parameter Value Defintions	Defines the possible values for setting and getting PCAN's environment information with the functions CANTP_SetValue and CANTP_GetValue.

3.8.1 PCAN-ISO-TP Handle Definitions

Defines the handles for the different PCAN busses (channels) within a class. The values are used as parameters where a TPCANTPHandle is needed.

Default/Undefined handle:

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_NONEBUS	0x0	Undefined/default value for a PCAN-ISO-TP channel

Handles for the ISA bus (Non-Plug and Play):

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_ISABUS1	0x21	PCAN-ISA interface, channel 1
	TPCANTPHandle	PCANTP_ISABUS2	0x22	PCAN-ISA interface, channel 2
	TPCANTPHandle	PCANTP_ISABUS3	0x23	PCAN-ISA interface, channel 3
	TPCANTPHandle	PCANTP_ISABUS4	0x24	PCAN-ISA interface, channel 4
	TPCANTPHandle	PCANTP_ISABUS5	0x25	PCAN-ISA interface, channel 5
	TPCANTPHandle	PCANTP_ISABUS6	0x26	PCAN-ISA interface, channel 6
	TPCANTPHandle	PCANTP_ISABUS7	0x27	PCAN-ISA interface, channel 7
	TPCANTPHandle	PCANTP_ISABUS8	0x28	PCAN-ISA interface, channel 8

Handles for the Dongle Bus (Non-Plug and Play):

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_DNGBUS1	0x31	PCAN-Dongle/LPT interface, channel 1

Handles for the PCI bus:

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_PCIBUS1	0x41	PCAN-PCI interface, channel 1
	TPCANTPHandle	PCANTP_PCIBUS2	0x42	PCAN-PCI interface, channel 2
	TPCANTPHandle	PCANTP_PCIBUS3	0x43	PCAN-PCI interface, channel 3
	TPCANTPHandle	PCANTP_PCIBUS4	0x44	PCAN-PCI interface, channel 4
	TPCANTPHandle	PCANTP_PCIBUS5	0x45	PCAN-PCI interface, channel 5
	TPCANTPHandle	PCANTP_PCIBUS6	0x46	PCAN-PCI interface, channel 6
	TPCANTPHandle	PCANTP_PCIBUS7	0x47	PCAN-PCI interface, channel 7
	TPCANTPHandle	PCANTP_PCIBUS8	0x48	PCAN-PCI interface, channel 8

Handles for the USB Bus:

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_USBBUS1	0x51	PCAN-USB interface, channel 1
	TPCANTPHandle	PCANTP_USBBUS2	0x52	PCAN-USB interface, channel 2
	TPCANTPHandle	PCANTP_USBBUS3	0x53	PCAN-USB interface, channel 3
	TPCANTPHandle	PCANTP_USBBUS4	0x54	PCAN-USB interface, channel 4

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_USBBUS5	0x55	PCAN-USB interface, channel 5
	TPCANTPHandle	PCANTP_USBBUS6	0x56	PCAN-USB interface, channel 6
	TPCANTPHandle	PCANTP_USBBUS7	0x57	PCAN-USB interface, channel 7
	TPCANTPHandle	PCANTP_USBBUS8	0x58	PCAN-USB interface, channel 8

Handles for the PC Card Bus:

	Type	Constant	Value	Description
	TPCANTPHandle	PCANTP_PCCBUS1	0x61	PCAN-PC Card interface, channel 1
	TPCANTPHandle	PCANTP_PCCBUS2	0x62	PCAN-PC Card interface, channel 2

Remarks

Note: These definitions are constant values in an object oriented environment (Delphi, .NET Framework) and declared as defines in C++ and Pascal (plain API).

Hardware Type and Channels

Non-Plug and Play: The hardware channels of this kind are used as registered. This mean, for example, it is allowed to register the PCANTP_ISABUS3 without having registered PCANTP_ISA1 and PCANTP_ISA2. It is a decision of each user, how to associate a PCAN-channel (logical part) and a port/interrupt pair (physical part).

Plug and 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 PCANTP_USBBUS1 to PCANTP_USBBUS4. The association of each channel with hardware is managed internally using the handle of hardware.

See also: Parameter Value Defintions below.

3.8.2 Parameter Value Defintions

Define the possible values for setting and getting PCAN-ISO-TP environment information with the functions CANTP_SetValue and CANTP_GetValue.

Debug configuration values:

	Type	Constant	Value	Description
	Int32	PCANTP_DEBUG_NONE	0	No CAN debug messages are being generated.
	Int32	PCANTP_DEBUG_CAN	1	CAN debug messages are written to the stdout output.

Channel values:

	Type	Constant	Value	Description
	Int32	PCANTP_CHANNEL_UNAVAILABLE	0	The ISO-TP PCAN channel handle is illegal, or its associated hardware is not available.
	Int32	PCANTP_CHANNEL_AVAILABLE	1	The ISO-TP PCAN channel handle is valid to connect/initialize. Furthermore, for Plug and Play hardware, this means that the hardware is plugged-in.
	Int32	PCANTP_CHANNEL_OCCUPIED	2	The ISO-TP PCAN channel handle is valid and is currently being used.

ISO-TP WaitForFrame parameter values:

	Type	Constant	Value	Description
	Int32	PCANTP_WFT_MAX_UNLIMITED	0x00	Disables checks for ISO-TP WaitForFrames overrun when receiving a FlowControl frames (PCANTP_N_WFT_OVRN error will never occur).
	Int32	PCANTP_WFT_MAX_DEFAULT	0x10	The default value used by the API: if the number of consecutive FlowControl frame with the wait status exceeds this value, a PCANTP_N_WFT_OVRN error will occur.

ISO-TP message pending indication values:

	Type	Constant	Value	Description
	Int32	PCANTP_MSG_PENDING_HIDE	0x00	Messages with the type PCANTP_MESSAGE_INDICATION will be automatically removed from the result of the CANTP_Read function.
	Int32	PCANTP_MSG_PENDING_SHOW	0x01	Messages with the type PCANTP_MESSAGE_INDICATION can be retrieved from the CANTP_Read function.

ISO-TP data padding values:

	Type	Constant	Value	Description
	Int32	PCANTP_CAN_DATA_PADDING_NONE	0x00	CAN frame data optimization is enabled.
	Int32	PCANTP_CAN_DATA_PADDING_ON	0x01	CAN frame data optimization is disabled: CAN data length is always 8 and data is padded with zeros.

ISO-TP CAN segmented values:

	Type	Constant	Value	Description
	Int32	PCANTP_CAN_UNSEGMENTED_OFF	0x00	Reception of unformatted (NON-ISO-TP) CAN frames is disabled.
	Int32	PCANTP_CAN_UNSEGMENTED_ON	0x01	Reception of unformatted (NON-ISO-TP) CAN frames is enabled.
	Int32	PCANTP_CAN_UNSEGMENTED_ALL_FRAMES	0x02	Reception of unformatted (NON-ISO-TP) CAN frames is enabled and frames composing segmented ISO-TP messages will be available in the reception queue too.

Remarks

These definitions are constant values in an object oriented environment (Delphi, .NET Framework) and declared as defines in C++ (plain API).

See also: TPCANTPParameter on page 33, PCAN-ISO-TP Handle Definitions on page 106.

3.8.3 FD Bit Rate Parameter Definitions

Defines the different configuration parameters used to create a flexible data rate string for FD capable PCAN channel initialization. These values are used as parameter with CANTP_InitializeFD (Class method version: InitializeFD).

Clock frequency parameters:

Type	Constant	Value	Description
String	PCANTP_BR_CLOCK	"f_clock"	Clock frequency in Hertz (80000000, 60000000, 40000000, 30000000, 24000000, 20000000)
String	PCANTP_BR_CLOCK_MHZ	"f_clock_mhz"	Clock frequency in Megahertz (80, 60, 40, 30, 24, 20)

Nominal bit rate parameters:

Type	Constant	Value	Description
String	PCANTP_BR_NOM_BRP	"nom_brp"	Clock prescaler for nominal time quantum (1..1024).
String	PCANTP_BR_NOM_TSEG1	"nom_tseg1"	TSEG1 segment for nominal bit rate in time quanta (1..256).
String	PCANTP_BR_NOM_TSEG2	"nom_tseg2"	TSEG2 segment for nominal bit rate in time quanta (1..128).
String	PCANTP_BR_NOM_SJW	"nom_sjw"	Synchronization Jump Width for nominal bit rate in time quanta (1..128)

Data bit rate parameters:

Type	Constant	Value	Description
String	PCANTP_BR_DATA_BRP	"data_brp"	Clock prescaler for fast data time quantum (1..1024).
String	PCANTP_BR_DATA_TSEG1	"data_tseg1"	TSEG1 segment for fast data bit rate in time quanta (1..32).
String	PCANTP_BR_DATA_TSEG2	"data_tseg2"	TSEG2 segment for fast data bit rate in time quanta (1..16).
String	PCANTP_BR_DATA_SJW	"data_sjw"	Synchronization Jump Width for fast data bit rate in time quanta (1..16)

Remarks

These definitions are constant values in an object oriented environment (Delphi, .NET Framework) and declared as defines in C++ (plain API).

Following points are to be respected in order to construct a valid FD bit rate string:

- The string must contain only one of the two possible clock frequency parameters, depending on the unit used (Hz, or MHz).
- The frequency to use must be one of the 6 listed within the clock frequency parameters.
- The value for each parameter must be separated with a '='. **Example**: "data_brp=1"
- Each pair of parameter/value must be separated with a ','. Blank spaces are allowed but are not necessary. Example: "f_clock_mhz=24, nom_brp=1,"
- Both bit rates, or only the nominal one, must be defined within the string (PCANTP_BR_DATA_* and PCANTP_BR_NOM_*, or only PCANTP_BR_NOM_*).

Example with nominal bit rate only:

A valid string representing 1 Mbit/sec for both, nominal and data bit rates:

```
"f_clock_mhz=20, nom_brp=5, nom_tseg1=2, nom_tseg2=1, nom_sjw=1"
```

Example with nominal and data bit rate:

A valid string representing 1 Mbit/sec for nominal bit rate, and 2 Mbit/sec for data bit rate:

```
"f_clock_mhz=20, nom_brp=5, nom_tseg1=2, nom_tseg2=1, nom_sjw=1, data_brp=2, data_tseg1=3, data_tseg2=1, data_sjw=1"
```

Parameter value ranges:

Parameter	Value Range
f_clock	[80000000, 60000000, 40000000, 30000000, 24000000, 20000000]
f_clock_mhz	[80, 60, 40, 30, 24, 20]
nom_brp	1 .. 1024
nom_tseg1	1 .. 256
nom_tseg2	1 .. 128
nom_sjw	1 .. 128
data_brp	1 .. 1024
data_tseg1	1 .. 32
data_tseg2	1 .. 16
data_sjw	1 .. 16

See Also: CANTP_InitializeFD on page 92, class method version: InitializeFD on page 45.

4 Additional Information

PCAN is the platform for PCAN-OBD-2, 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).

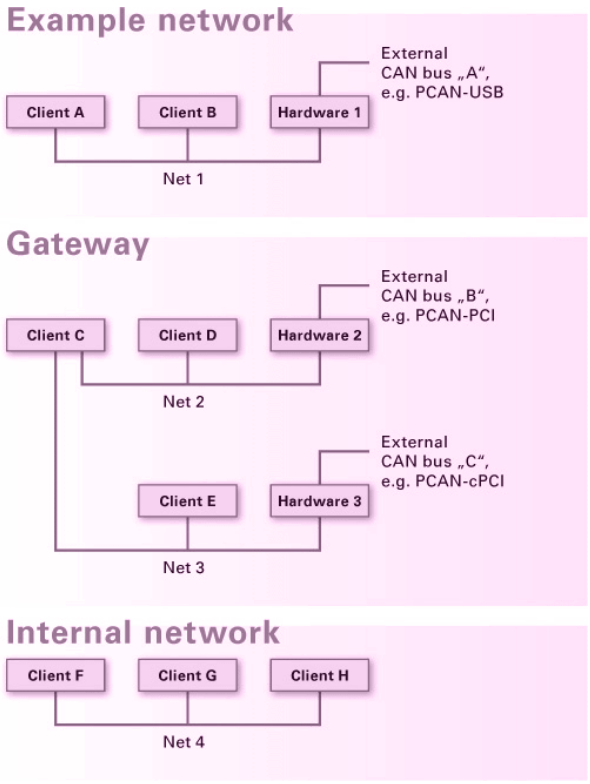
Topics	Description
PCAN Fundamentals	This section contains an introduction to PCAN.
PCAN-Basic	This section contains general information about the PCAN-Basic API.
PCAN-API	This section contains general information about the PCAN-API.
ISO-TP Network Addressing	This section contains general information 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`, and `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 116.

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:

- Retrieves 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, e.g. "Listen-Only" mode, automatic reset of the CAN controller, etc.
- The use of event notifications for faster processing of incoming 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 two 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 three 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 version**: Initialize) or CAN_InitializeFD (**Class method version**: 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 method versions**: 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 method versions**: 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 version**: 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-PC Card	Yes	Pcan_pcc.sys

See also: PCAN Fundamentals on page 111, PCAN-API on page 114, ISO-TP Network Addressing Format on page 116.

4.3 PCAN-API

Also called CanApi2 interface, is a synonym for CAN Application Programming Interface (version 2) and is a comprehensively programming interface to the PCAN system of the company PEAK-System Technik GmbH. This interface is more comprehensive than PCAN-Basic.

Important difference to PCAN-Basic:

- Transmit a CAN message at a fixed point of time.
- Several application programs could be connected to a single PCAN-PC hardware.
- Detailed information to PCAN-PC hardware and the PCAN system (PCAN net and PCAN client)
- The PCAN client is connected via the net to the PCAN-PC hardware.

The following text is a short overview to the CanApi2 functions. The functions itself can be categorized as follows: fields control, register, and remove functions for nets and hardware.

Function	Description
CloseAll	Disconnects all hardware, nets, and clients.
RegisterHardware	Registers a non-Plug and Play CAN hardware.
RegisterHardwarePCI	Registers a PCI CAN hardware.
RegisterNet	Registers a PCAN net.
RemoveHardware	Removes and deactivates CAN hardware.
RemoveNet	Removes a PCAN net.

Fields configuaration, configuration functions for nets and hardware:

Function	Description
SetDeviceName	Sets the PCAN device to be used for subsequent CanApi2 function calls.
SetDriverParam	Configures a driver parameter, e.g. the size of the receive or transmit buffer.
SetHwParam	Configures a hardware parameter, e.g. the PEAK serial number and additional parameters for the PCAN-USB hardware.
SetNetParam	Configures net parameter

Fields client, functions for the management of the clients:

Function	Description
ConnectToNet	Connects a client to a PCAN net.
DisconnectFromNet	Disconnects a client from a PCAN net.
RegisterClient	Registers an application as PCAN client.
RegisterMsg	Expands the reception filter of a client.
RemoveAllMsgs	Resets the filter of a client for a connected net.
RemoveClient	Removes a client from the driver.
ResetClient	Resets the receive and transmit queue of a client.
ResetHardware	Resets a CAN hardware.
SetClientFilter	Configures the reception filter of a client.
SetClientFilterEx	Configures the reception filter of a client.
SetClientParam	Configures a client parameter, e.g. - self-receive mode of transmitted messages - improves the accuracy of the reception filter.

Fields communication, functions for the data interchange over the CAN bus:

Function	Description
Read	Reads a received CAN message, including the reception time stamp.
Read-Multi	Reads multiple received CAN messages.
Write	Transmits a CAN message at a specified time.

Fields information, functions for the information about clients, nets, drivers, and hardware:

Function	Description
GetClientParam	Retrieves client parameter, e.g. - total number of transmitted or received CAN messages - the PCAN driver name, PCAN net, or PCAN client name - the number of received bits
GetDeviceName	Retrieves the currently used PCAN device.
GetDiagnostic	Reads the diagnostic text buffer.
GetDriverName	Retrieves the name of a PCAN device type.
GetDriverParam	Retrieves a driver parameter.
GetErrText	Translates an error code into a text.
GetHwParam	Retrieves a hardware parameter.
GetNetParam	Retrieves a net parameter.
GetSystemTime	Gets the system time.
Msg2Text	Creates a text form of a CAN message.
Status	Detects the current status of a CAN hardware.
VersionInfo	Reads version and copyright information from the driver.

See also: PCAN Fundamentals on page 111, PCAN-Basic on page 112, ISO-TP Network Addressing FormatISO-TP Network Addressing on page 116.

4.4 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	Defines mappings with CANTP_AddMapping.
	29 bits	Defines 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	Defines mappings with CANTP_AddMapping.
	29 bits	Defines mappings with CANTP_AddMapping.
Mixed addressing PCANTP_FORMAT_MIXED	11 bits	Defines 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).

4.5 Using Events

Event objects can be used to automatically notify a client on reception of an ISO-TP 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 CANTP_SetValue function (class method version: SetValue) to set the parameter PCANTP_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 CANTP_Read function (class method version: Read), and the ISO-TP messages can be processed.

Remarks

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:

CANTP_SetValue on page 94, class method version: SetValue on page 50.

CANTP_Read on page 101, class method version: Read on page 77.