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

Documentation
Contents

1 PCAN-ISO-TP API 5

2 Introduction 6
  2.1 Understanding PCAN-ISO-TP 6
  2.2 Using PCAN-ISO-TP 6
  2.3 License Regulations 7
  2.4 Features 7
  2.5 System Requirements 7
  2.6 Scope of Supply 7

3 DLL API Reference 8
  3.1 Namespaces 8
    3.1.1 Peak.Can.IsoTp 8
  3.2 Units 9
    3.2.1 PCANTP Unit 9
  3.3 Classes 10
    3.3.1 CanTpApi 10
    3.3.2 TCanTpApi 11
  3.4 Structures 12
    3.4.1 TPCANTPMsg 12
    3.4.2 TPCANTPTimestamp 14
  3.5 Types 16
    3.5.1 TPCANTPHandle 16
    3.5.2 TPCANTPStatus 17
    3.5.3 TPCANTPBaudrate 19
    3.5.4 TPCANTPBitrateFD 21
    3.5.5 TPCANTPHWType 22
    3.5.6 TPCANTPMessageType 24
    3.5.7 TPCANTPIdType 25
    3.5.8 TPCANTPFormatType 28
    3.5.9 TPCANTPAddressingType 30
    3.5.10 TPCANTPConfirmation 31
    3.5.11 TPCANTPParameter 33
  3.6 Methods 39
    3.6.1 Initialize 40
    3.6.2 Initialize (TPCANTPHandle, TPCANTPBaudrate) 40
    3.6.3 Initialize (TPCANTPHandle, TPCANTPBaudrate, TPCANTPHWType, UInt32, UInt16) 43
    3.6.4 InitializeFD 45
    3.6.5 Uninitialize 48
    3.6.6 SetValue 50
    3.6.7 SetValue (TPCANTPHandle, TPCANTPParameter, UInt32, UInt32) 50
    3.6.8 SetValue (TPCANTPHandle, TPCANTPParameter, StringBuffer, UInt32) 53
    3.6.9 SetValue (TPCANTPHandle, TPCANTPParameter, byte(), UInt32) 54
    3.6.10 GetErrorText(TPCANStatus, UInt16, StringBuffer) 57
    3.6.11 AddMapping 60
    3.6.12 RemoveMapping 64
    3.6.13 GetValue 67
3.6.14 GetValue (TPCANTPHandle, TPCANTPParameter, StringBuilder, Uint32) 67
3.6.15 GetValue (TPCANTPHandle, TPCANTPParameter, Uint32, Uint32) 70
3.6.16 GetValue (TPCANTPHandle, TPCANTPParameter, byte(), UInt32) 72
3.6.17 GetStatus 75
3.6.18 Read 77
3.6.19 Read(TPCANTPHandle, TPCANPMsg) 78
3.6.20 Read(TPCANTPHandle, TPCANPMsg, TPCANPTimestamp) 81
3.6.21 Write 84
3.6.22 Reset 88

3.7 Functions
3.7.1 CANTP_Initialize 90
3.7.2 CANTP_InitializeFD 92
3.7.3 CANTP_Uninitialize 93
3.7.4 CANTP_SetValue 94
3.7.5 CANTP_GetErrorText 95
3.7.6 CANTP_AddMapping 96
3.7.7 CANTP_RemoveMapping 98
3.7.8 CANTP_GetValue 99
3.7.9 CANTP_GetStatus 100
3.7.10 CANTP_Read 101
3.7.11 CANTP_Write 103
3.7.12 CANTP_Reset 104

3.8 Definitions
3.8.1 PCAN-ISO-TP Handle Definitions 106
3.8.2 Parameter Value Definitions 107
3.8.3 FD Bit Rate Parameter Definitions 108

4 Additional Information
4.1 PCAN Fundamentals 111
4.2 PCAN-Basic 112
4.3 PCAN-API 114
4.4 ISO-TP Network Addressing Format 116
4.5 Using Events 117
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.

- Introduction on page 6
- DLL API Reference on page 8
- Additional Information on page 111
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak</td>
<td>Contains all namespaces that are part of the managed programming environment from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can</td>
<td>Contains types and classes for using the PCAN API from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.Light</td>
<td>Contains types and classes for using the PCAN-Light API from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.Basic</td>
<td>Contains types and classes for using the PCAN-Basic API from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.Ccp</td>
<td>Contains types and classes for using the CCP API implementation from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.Xcp</td>
<td>Contains types and classes for using the XCP API implementation from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.Iso.Tp</td>
<td>Contains types and classes for using the PCAN-ISO-TP API implementation from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.Uds</td>
<td>Contains types and classes for using the PCAN-UDS API implementation from PEAK-System.</td>
</tr>
<tr>
<td>Peak.Can.ObdII</td>
<td>Contains types and classes for using the PCAN-OBD-2 API implementation from PEAK-System.</td>
</tr>
<tr>
<td>Peak.lin</td>
<td>Contains types and classes used to handle with LIN devices from PEAK-System.</td>
</tr>
<tr>
<td>Peak.RP1210A</td>
<td>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).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPHandle</td>
<td>Represents a PCAN-ISO-TP channel handle.</td>
</tr>
</tbody>
</table>

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanTpApi</td>
<td>Defines a class which represents the PCAN-ISO-TP API.</td>
</tr>
</tbody>
</table>
Structures

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPMsg</td>
<td>Defines a CAN ISO-TP message. The members of this structure are sequentially byte aligned.</td>
</tr>
<tr>
<td>TPCANPTimestamp</td>
<td>Defines a time-stamp of a CAN ISO-TP message.</td>
</tr>
</tbody>
</table>

Enumerations

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

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

<table>
<thead>
<tr>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>PCANTP Unit Delphi unit for using the PCAN-ISO-TP API from PEAK-System.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TPCANTPHandle Represents a PCAN-ISO-TP channel handle.</td>
</tr>
</tbody>
</table>

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TCanTpApi Defines a class which represents the PCAN-ISO-TP API.</td>
</tr>
</tbody>
</table>
Structures

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPMsg</td>
<td>Defines a CAN ISO-TP message. The members of this structure are sequentially byte aligned.</td>
</tr>
<tr>
<td>TPCANTPTimestamp</td>
<td>Defines a time-stamp of a CAN ISO-TP message.</td>
</tr>
</tbody>
</table>

Enumerations

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

3.3 Classes

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

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanTpApi</td>
<td>Defines a class to use the PCAN-ISO-TP API within the Microsoft's .NET Framework programming environment.</td>
</tr>
<tr>
<td>TCanTpApi</td>
<td>Defines a class to use the PCAN-ISO-TP API within the Delphi programming environment.</td>
</tr>
</tbody>
</table>

3.3.1 CanTpApi

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

Syntax

C#

```csharp
public static class CanTpApi
```

C++ / CLR

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

Visual Basic

```vbnet
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.:

```csharp
TPCANTPStatus res;
// Static use without any instance.
//
res = CanTpApi.Initialize(CanTpApi.PCANTP_USBBUS1, TPCANTP.Baudrate.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**

```pascal
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPMsg</td>
<td>Defines a CAN ISO-TP message.</td>
</tr>
<tr>
<td>TPCANTPTimestamp</td>
<td>Defines a time-stamp of a CAN ISO-TP message.</td>
</tr>
</tbody>
</table>

#### 3.4.1 TPCANTPMsg

Defines a CAN ISO-TP message.

**Syntax**

**C++**

```cpp
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**

```pascal
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#**

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


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

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

### Fields

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

**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++**

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

**Pascal OO**

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

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

C++/CLR

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

Visual Basic

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

Remarks

Calculation of total of microseconds: \( \text{micros} + 1000 \times \text{millis} + 0x100000000 \times 1000 \times \text{millis}\_\text{overflow} \)

Fields

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>millis</td>
<td>Base-value: milliseconds: 0.. 2^32-1</td>
</tr>
<tr>
<td>millis_overflow</td>
<td>Roll-arounds of millis</td>
</tr>
<tr>
<td>micros</td>
<td>Microseconds: 0.. 999</td>
</tr>
</tbody>
</table>

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:

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

### 3.5.1 TPCANTPHandle

Represents a PCAN-ISO-TP channel handle.

**Syntax**

**C++**

```cpp
#define TPCANTPHandle WORD
```

**C++ / CLR**

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

**C#**

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

**Visual Basic**

```vbnet
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#

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

Visual Basic

```vbnet
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++

```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

```c
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#

```csharp
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

```pascal
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

```vb
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_OK</td>
<td>0x00000 (00000)</td>
<td>No error. Success</td>
</tr>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>0x00001 (00001)</td>
<td>Not initialized (ex. a non-initialized CANTP channel or CAN ID mapping).</td>
</tr>
<tr>
<td>PCANTP_ERROR_ALREADY_INITIALIZED</td>
<td>0x00002 (00002)</td>
<td>Already initialized (used by CANTP channel or CAN ID mapping).</td>
</tr>
<tr>
<td>PCANTP_ERROR_NO_MEMORY</td>
<td>0x00003 (00003)</td>
<td>Failed to allocate memory.</td>
</tr>
<tr>
<td>PCANTP_ERROR_OVERFLOW</td>
<td>0x00004 (00004)</td>
<td>Buffer overflow occurred (too many channels initialized or too many messages in queue).</td>
</tr>
<tr>
<td>PCANTP_ERROR_TIMEOUT</td>
<td>0x00006 (00006)</td>
<td>Timeout while trying to access the PCAN-ISO-TP API.</td>
</tr>
<tr>
<td>PCANTP_ERROR_NO_MESSAGE</td>
<td>0x00007 (00007)</td>
<td>No message available.</td>
</tr>
<tr>
<td>PCANTP_ERROR_WRONG_PARAM</td>
<td>0x00008 (00008)</td>
<td>Invalid parameter</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSLIGHT</td>
<td>0x00009 (00009)</td>
<td>Bus error: an error counter reached the 'light' limit.</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSHEAVY</td>
<td>0x0000A (000010)</td>
<td>Bus error: an error counter reached the 'heavy' limit.</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSOFF</td>
<td>0x0000B (000011)</td>
<td>Bus error: the CAN controller is in bus-off state.</td>
</tr>
<tr>
<td>PCANTP_ERROR_CAN_ERROR</td>
<td>0x80000000 (2147483648)</td>
<td>PCAN-Basic error flag (remove the flag to get a TPCANStatus error code)</td>
</tr>
</tbody>
</table>

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++

```cpp
#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
```
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#

```csharp
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

```cpp
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,
}```
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_BAUD_1M</td>
<td>20</td>
<td>1 MBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_800K</td>
<td>22</td>
<td>800 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_500K</td>
<td>28</td>
<td>500 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_250K</td>
<td>284</td>
<td>250 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_125K</td>
<td>796</td>
<td>125 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_100K</td>
<td>17199</td>
<td>100 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_95K</td>
<td>49998</td>
<td>95,238 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_83K</td>
<td>34091</td>
<td>83,333 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_50K</td>
<td>18223</td>
<td>50 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_47K</td>
<td>5140</td>
<td>47,619 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_33K</td>
<td>35631</td>
<td>33,333 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_20K</td>
<td>21296</td>
<td>20 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_10K</td>
<td>26415</td>
<td>10 kBit/s</td>
</tr>
<tr>
<td>PCANTP_BAUD_5K</td>
<td>32639</td>
<td>5 kBit/s</td>
</tr>
</tbody>
</table>

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
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#

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

Visual Basic

```vbnet
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++

```cpp
#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


ten_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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_TYPE_Isa</td>
<td>1</td>
<td>PCAN-ISA 82C200</td>
</tr>
<tr>
<td>PCANTP_TYPE_Isa_Sja</td>
<td>9</td>
<td>PCAN-ISA SJA1000</td>
</tr>
<tr>
<td>PCANTP_TYPE_Isa_Phytec</td>
<td>4</td>
<td>PHYTEC ISA</td>
</tr>
<tr>
<td>PCANTP_TYPE_Dng</td>
<td>2</td>
<td>PCAN-Dongle 82C200</td>
</tr>
<tr>
<td>PCANTP_TYPE_Dng_Epp</td>
<td>3</td>
<td>PCAN-Dongle EPP 82C200</td>
</tr>
<tr>
<td>PCANTP_TYPE_Dng_Sja</td>
<td>5</td>
<td>PCAN-Dongle SJA1000</td>
</tr>
<tr>
<td>PCANTP_TYPE_Dng_Sja_Epp</td>
<td>6</td>
<td>PCAN-Dongle EPP SJA1000</td>
</tr>
</tbody>
</table>

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++

```cpp
#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

```pascal
{$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#

```csharp
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

```csharp
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

```vbnet
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**

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

**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++**

```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**

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

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

C++ / CLR  

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

Visual Basic  

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

**Values**  

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ID_CAN_11BIT</td>
<td>1</td>
<td>Standard CAN ID (11-bit identifier)</td>
</tr>
<tr>
<td>PCANTP_ID_CAN_29BIT</td>
<td>2</td>
<td>Extended CAN ID (29-bit identifier)</td>
</tr>
</tbody>
</table>

**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:

- **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 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 29-bit CAN ID.
- `PCANTP_ID_CAN_HAS_PRIORITY(id_type: Byte)`: states if the IDTYPE value corresponds to a 29-bit CAN ID with a defined priority.
**PCAN-ISO-TP API – Documentation**

**Syntax**

**C++**

```cpp
#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_PRIORITY(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**

```pascal
class function PCANTP_ID_CAN_GET_29B(j1939_priority: Byte): TPCANTPIdType;
class function PCANTP_ID_CAN_GET_PRIORITY(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#**

```csharp
public static class TPCANTPIdTypePriority
{
    public static TPCANTPIdType PCANTP_ID_CAN_GET_29B(byte j1939_priority);
    public static byte PCANTP_ID_CAN_GET_PRIORITY(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**

```csharp
public ref class TPCANTPIdTypePriority
{
    public:
        static TPCANTPIdType PCANTP_ID_CAN_GET_29B(Byte j1939_priority);
        static Byte PCANTP_ID_CAN_GET_PRIORITY(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**

```vbnet
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_PRIORITY(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:

```c
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++

```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

```pascal
{$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#

```csharp
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

```cpp
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

```vb
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

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

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++**

```cpp
#define TPCANTPAddressingType BYTE
#define PCANTP_ADDRESSING_UNKNOWN 0x00
#define PCANTP_ADDRESSING_PHYSICAL 0x01
#define PCANTP_ADDRESSING_FUNCTIONAL 0x02
```

**Pascal OO**

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

**C#**

```csharp
public enum TPCANTPAddressingType : byte
{
```
PCANTP_ADDRESSING_UNKNOWN = 0x00,
PCANTP_ADDRESSING_PHYSICAL = 0x01,
PCANTP_ADDRESSING_FUNCTIONAL = 0x02,
}

C++ / CLR

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

Visual Basic

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

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ADDRESSING_UNKNOWN</td>
<td>0</td>
<td>Unknown addressing</td>
</tr>
<tr>
<td>PCANTP_ADDRESSING_PHYSICAL</td>
<td>1</td>
<td>Physical addressing (used for communication between 2 nodes)</td>
</tr>
<tr>
<td>PCANTP_ADDRESSING_FUNCTIONAL</td>
<td>2</td>
<td>Functional addressing (used to broadcast messages on the CAN bus)</td>
</tr>
</tbody>
</table>

See also: TPCANTPMs 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++

```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

```pascal
{$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#

```csharp
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

```cpp
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

```vb
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

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

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#

```csharp
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

```c++
public enum TPCANPPParameter : 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

```pascal
{$Z1}
TPCANPPParameter = (  
    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

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

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.
**PCAN-ISO-TP API – Documentation**

**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 Definitions 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 Definitions on page 107.

PCANTP_PARAM_SEPARATION_TIME_OPTIMIZATION

**Access:**

**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 Definitions 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>📈 S</td>
<td>Initialize</td>
</tr>
<tr>
<td>🌐</td>
<td>Initializes a PCANTP channel.</td>
</tr>
<tr>
<td>🌐 S</td>
<td>Uninitialize</td>
</tr>
<tr>
<td>🌐</td>
<td>Uninitializes a PCANTP channel.</td>
</tr>
</tbody>
</table>

#### Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>📈 S</td>
<td>SetValue</td>
</tr>
<tr>
<td>📈 S</td>
<td>Sets a configuration or information value within a PCANTP channel.</td>
</tr>
<tr>
<td>🌐 S</td>
<td>AddMapping</td>
</tr>
<tr>
<td>🌐 S</td>
<td>Configures the ISO-TP mapping between a CAN ID and an ISO-TP network</td>
</tr>
<tr>
<td>🌐</td>
<td>addressing information.</td>
</tr>
<tr>
<td>🌐 S</td>
<td>RemoveMapping</td>
</tr>
<tr>
<td>🌐</td>
<td>Removes a previously configured ISO-TP mapping.</td>
</tr>
</tbody>
</table>
### Information

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetValue</td>
<td>Retrieves information from a PCANTP channel.</td>
</tr>
<tr>
<td>GetStatus</td>
<td>Retrieves the current bus status of a PCANTP channel.</td>
</tr>
<tr>
<td>GetErrorText</td>
<td>Gets a descriptive text for an error code.</td>
</tr>
</tbody>
</table>

### Communication

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Reads a CAN message from the receive queue of a PCANTP channel.</td>
</tr>
<tr>
<td>Write</td>
<td>Transmits a CAN message using a connected PCANTP channel.</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets the receive and transmit queues of a PCANTP channel.</td>
</tr>
</tbody>
</table>

#### 3.6.1 Initialize

Initializes a PCANTP channel.

### Overloads

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize(TPCANTPHandle, TPCANTPBaudrate)</td>
<td>Initializes a Plug and Play PCANTP channel.</td>
</tr>
<tr>
<td>Initialize(TPCANTPHandle, TPCANTPBaudrate, TPCANTPHWType, UInt32, UInt16)</td>
<td>Initializes a Non-Plug-and Play PCANTP channel.</td>
</tr>
</tbody>
</table>

#### 3.6.2 Initialize(TPCANTPHandle, TPCANTPBaudrate)

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

### Syntax

**Pascal OO**

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

**C#**

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

**C++ / CLR**

```cpp
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Baudrate</td>
<td>The speed for the communication (see TPCANTPBaudrate on page 19)</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_ALREADY_INITIALIZED</td>
<td>Indicates that the desired PCANTP channel is already in use.</td>
</tr>
<tr>
<td>PCANTP_ERROR_CAN_ERROR</td>
<td>This error flag states that the error is composed of a more precise PCAN-Basic error.</td>
</tr>
</tbody>
</table>

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#

```csharp
TPCANTPStatus result;

// The Plug and Play channel (PCAN-FCI) is initialized.
result = CanTpApi.Initialize(CanTpApi.PCANTP_PCIEBUS2,
    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 = CCanApi.Initialize(CCanApi.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.
CCanApi.Uninitialize(CCanApi.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

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

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 uninitialized processes for a Non-Plug and Play channel (channel 1 of the PCAN-DNG).

C#:

```csharp
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle)</td>
</tr>
<tr>
<td>Bitrate</td>
<td>The speed for the communication (see TPCANTPBitrateFD)</td>
</tr>
</tbody>
</table>

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.

```csharp
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++
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);
```

```visualbasic
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( _
  ByVal CanChannel As TPCANTPHandle) As TPCANTPStatus
End Function
Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
</tbody>
</table>

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 uninitialised.

Example

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

**C#**

```csharp
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**

```csharp
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**

```vbnet
Dim result As TPCANTPStatus

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

---

49
result = CanTpApi.Initialize(CanTpApi.PCANTP_PCIBUS2, TPCANTP_Baudrate.PCANTP_BAUD_500K)
If result <> TPCANTP_Status.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: TPCANTP_Status;
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetValue(TPCANTPHandle, TPCANTPParameter, UInt32, UInt32);</td>
<td>Sets a configuration or information numeric value within a PCANTP channel.</td>
</tr>
<tr>
<td>SetValue(TPCANTPHandle, TPCANTPParameter, String, UInt32);</td>
<td>Sets a configuration or information string value within a PCANTP channel.</td>
</tr>
<tr>
<td>SetValue(TPCANTPHandle, TPCANTPParameter, Byte[], UInt32);</td>
<td>Sets a configuration or information with an array of bytes within a PCANTP channel.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to be set (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>NumericBuffer</td>
<td>The buffer containing the numeric value to be set.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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 Definitions. 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#

```csharp
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

```csharp
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

```vbnet
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, Delegate(UInt32, Convert.(iBuffer)))
If result <> TPCANTPStatus.PCANTP_ERROR_OK Then
```
```pascal
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**

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

**C#**

```csharp
[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**

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

```cpp
```
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANT channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to be set (see TPCANTPParameter on page 33).</td>
</tr>
<tr>
<td>NumericBuffer</td>
<td>The buffer containing the string value to be set.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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

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 Definitions 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

Imports System

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

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16).</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to be set (see TPCANTPParameter on page 33).</td>
</tr>
<tr>
<td>NumericBuffer</td>
<td>The buffer containing the array value to be set.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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 Definitions.

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#

```csharp
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

```cpp
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

```vbnet
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

```pascal
var
    result: TPCANTPStatus;
    bufferArray: array[0..2] of Byte;
```
begin
  // Defines unlimited blocksize.
  bufferArray[0] := 0;
  result := TCanTpApi.SetValue(TCanTpApi.PCANTP_USBUS1, PCANTP_PARAM_BLOCK_SIZE,
                              Pointer(@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 Definitions 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( _
  anError As TPCANTPStatus, _
  Language As UInt16, _
  StringBuffer As StringBuilder, _
): TPCANTPStatus;

begin
  // Defines unlimited blocksize.
  bufferArray[0] := 0;
  result := TCanTpApi.SetValue(TCanTpApi.PCANTP_USBUS1, PCANTP_PARAM_BLOCK_SIZE,
                              Pointer(@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 Definitions 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( _
  anError As TPCANTPStatus, _
  Language As UInt16, _
  StringBuffer As StringBuilder, _
): TPCANTPStatus;

begin
  // Defines unlimited blocksize.
  bufferArray[0] := 0;
  result := TCanTpApi.SetValue(TCanTpApi.PCANTP_USBUS1, PCANTP_PARAM_BLOCK_SIZE,
                              Pointer(@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 Definitions 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( _
  anError As TPCANTPStatus, _
  Language As UInt16, _
  StringBuffer As StringBuilder, _
): TPCANTPStatus;
ByVal StringBuffer As StringBuilder) As TPCANTPStatus

End Function

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>A TPCANTPStatus error code</td>
</tr>
<tr>
<td>Language</td>
<td>Indicates a &quot;Primary Language ID&quot;.</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>A buffer for a null-terminated char array.</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_WRONG_PARAM</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Primary Language ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral (System dependant)</td>
<td>00h (0)</td>
</tr>
<tr>
<td>English</td>
<td>09h (9)</td>
</tr>
<tr>
<td>German</td>
<td>07h (7)</td>
</tr>
<tr>
<td>French</td>
<td>0Ch (12)</td>
</tr>
<tr>
<td>Italian</td>
<td>10h (16)</td>
</tr>
<tr>
<td>Spanish</td>
<td>0Ah (10)</td>
</tr>
</tbody>
</table>

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 GetLastErrorText 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#:

```csharp
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 occurred");
}
else
```
MessageBox.Show(strMsg.ToString());

**C++ / CLR**

```cpp
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 occurred");
}
else
    MessageBox::Show(strMsg->ToString());
```

**Visual Basic**

```vbnet
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 occurred")
Else
    MessageBox.Show(strMsg.ToString())
End If
```

**Pascal OO**

```pascal
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( _
        TPCANTPStatus.PCANTP_ERROR_ALREADY_INITIALIZED, 0, strMsg);
    if (result <> TPCANTP_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 PCAN-TP channel.

**Syntax**

**Pascal OO**

```pascal
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#**

```csharp
[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**

```csharp
[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,
  [MarshalAs(UnmanagedType::U1)] TPCANTPAddressingType targetType,
  [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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCAN-TP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>canID</td>
<td>The CAN Identifier to be mapped.</td>
</tr>
<tr>
<td>canIDResponse</td>
<td>The CAN Identifier that is used to respond to a request with the CAN Id 'canId'.</td>
</tr>
<tr>
<td>canIDType</td>
<td>Type of CAN identifier</td>
</tr>
<tr>
<td>formatType</td>
<td>Format type of the ISO-TP network addressing information.</td>
</tr>
<tr>
<td>msgType</td>
<td>ISO-TP message type</td>
</tr>
<tr>
<td>sourceAddr</td>
<td>Source address</td>
</tr>
<tr>
<td>targetAddr</td>
<td>Target address</td>
</tr>
<tr>
<td>targetType</td>
<td>Type of the target</td>
</tr>
<tr>
<td>remoteAddr</td>
<td>Remote address</td>
</tr>
</tbody>
</table>

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 PCAN-TP 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:

<table>
<thead>
<tr>
<th>FormatType parameter</th>
<th>Valid canIdType parameter</th>
<th>Valid msgType parameter</th>
<th>Valid targetType parameter</th>
<th>Valid remoteAddr parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_FORMAT_NORMAL</td>
<td>Any</td>
<td>PCANTP_MESSAGE_DIAGNOSTIC</td>
<td>Any values</td>
<td>0x00 (value is ignored)</td>
</tr>
<tr>
<td>PCANTP_FORMAT_EXTENDED</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>PCANTP_FORMAT_MIXED</td>
<td>PCANTP_ID_CAN_11BIT</td>
<td>PCANTP_MESSAGE_REMOTE_DIAGNOSTIC</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>

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 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#

```csharp
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

```cpp
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.");
```
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_USBUSB1;
  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**

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

**C#**

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

**C++ / CLR**

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

```cpp
...
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>canID</td>
<td>The CAN Identifier that identifies the mapping to remove.</td>
</tr>
</tbody>
</table>

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#**

```csharp
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**

```cpp
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 = &H1
Dim N_SA As Byte = &H1
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, PCANTP_ID_CAN_11BIT, 
                          PCANTP_FORMAT_NORMAL, 
                          PCANTP_MESSAGE_DIAGNOSTIC, 
                          N_SA, N_TA, 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetValue(TPCANTPHandle, TPCANTPParameter, UInt32, UInt32);</td>
<td>Retrieves information from a PCANTP channel in numeric form.</td>
</tr>
<tr>
<td>GetValue(TPCANTPHandle, TPCANTPParameter, String, UInt32);</td>
<td>Retrieves information from a PCANTP channel in text form.</td>
</tr>
<tr>
<td>GetValue(TPCANTPHandle, TPCANTPParameter, Byte[], UInt32)</td>
<td>Retrieves information from a PCANTP channel in byte array form.</td>
</tr>
</tbody>
</table>

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#

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

C++ / CLR

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

Visual Basic

```vbnet
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTPGetValue")> _
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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to retrieve (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>The buffer to return the required string value.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

**Returns**

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.</td>
</tr>
<tr>
<td>PCANTP_ERROR_WRONG_PARAM</td>
<td>Indicates that the parameters passed to the method are invalid. Check the value of 'Parameter' and assert it is compatible with a string buffer.</td>
</tr>
</tbody>
</table>

**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#

```csharp
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

```csharp
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

```vbnet
Dim result As TPCANTPStatus
Dim BufferString As StringBuilder

' Get API version.
BufferString = New StringBuilder(255)
result = CanTpApi.GetValue(CanTpApi.PCANTP_NONEBUS,
    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

```pascal
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);
```
See also: SetValue on page 50, TPCANTPPParameter on page 33, Parameter Value Definitions on page 107.

Plain function version: CANTP_GetValue on page 99.

3.6.15 GetValue (TPCANTPHandle, TPCANTPPParameter, UInt32, UInt32)

Retrieves information from a PCAN channel in numeric form.

Syntax

Pascal OO

    class function GetValue( 
        CanChannel: TPCANTPHandle; 
        Parameter: TPCANTPPParameter; 
        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)] 
        TPCANTPPParameter 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)] 
        TPCANTPPParameter Parameter, 
        UInt32% NumericBuffer, 
        UInt32 BufferLength);

Visual Basic

    <DllImport("PCAN-ISO-TP.dll", EntryPoint:"CANTP_GetValue")> _
    Public Shared Function GetValue( _
        ByVal CanChannel As TPCANTPHandle, _
        ByVal Parameter As TPCANTPPParameter, _
        ByRef NumericBuffer As UInt32, _
        ByVal BufferLength As UInt32) As TPCANTPStatus _
    End Function
Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to retrieve (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>NumericBuffer</td>
<td>The buffer to return the required numeric value.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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#**

```csharp
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**

```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, 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**

```vbnet
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.ToInt32(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 Definitions 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#**

```csharp
[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**

```csharp
[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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to be set (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>Buffer</td>
<td>The buffer containing the array value to retrieve.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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#

```csharp
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");
```
```cpp
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>PCANTP_ERROR_OK</th>
<th>Indicates that the status of the given PCANTP channel is OK.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_BUSLIGHT</td>
<td>Indicates a bus error within the given PCANTP channel. The hardware is in bus-light status.</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSHEAVY:</td>
<td>Indicates a bus error within the given PCANTP channel. The hardware is in bus-heavy status.</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSOFF:</td>
<td>Indicates a bus error within the given PCANTP channel. The hardware is in bus-off status.</td>
</tr>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED:</td>
<td>Indicates that the given PCANTP channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.</td>
</tr>
</tbody>
</table>

Remarks

When the hardware status is bus-off, an application cannot communicate anymore. Consider using the PCAN-Basic property PCAN_BUSOFF_AUTORESET which instructs the API to automatically reset the CAN controller when a bus-off state is detected.
Another way to reset errors like bus-off, bus-heavy, and bus-light, is to uninitialized and initialize again the channel used. This causes a hardware reset.

Example

The following example shows the use of the method “GetStatus” on the channel PCANTP_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```csharp
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

```cpp
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

```vbnet
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

```pascal
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 Definitions 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.

<table>
<thead>
<tr>
<th>Overloads</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Read icon]</td>
<td>Read(TPCANTPHandle, TPCANTPMsg)</td>
<td>Reads a CAN ISO-TP message from the receive queue of a PCANTP channel.</td>
</tr>
</tbody>
</table>
3.6.19 **Read(TPCANTPHandle, TPCANTPMsg)**

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

**Syntax**

**Pascal OO**

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

**C#**

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

**C++ / CLR**

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

**Visual Basic**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Message Buffer</td>
<td>A TPCANTPMsg buffer to store the CAN ISO-TP message.</td>
</tr>
</tbody>
</table>

**Returns**

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

<table>
<thead>
<tr>
<th>PCANTP_ERROR_NO_MESSAGE</th>
<th>Indicates that the receive queue of the channel is empty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>Indicates that the given PCANTP channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.</td>
</tr>
</tbody>
</table>
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#  

```csharp
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 occurred");
        // Here can be decided if the loop has to be terminated.
        bStop = HandleReadError(result);
    }
} while (!bStop);
```

C++ / CLR  

```cpp
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**

```vbnet
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**

```pascal
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

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

C#

```csharp
[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

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

Visual Basic

```vb
<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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Message Buffer</td>
<td>A TPCANTPMsg buffer to store the CAN ISO-TP message.</td>
</tr>
<tr>
<td>TimestampBuffer</td>
<td>A TPCANTPTimestamp buffer to get the reception time of the message.</td>
</tr>
</tbody>
</table>
Returns
The return value is a TPCANTPStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_NO_MESSAGE</td>
<td>Indicates that the receive queue of the channel is empty.</td>
</tr>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>Indicates that the given PCANTP channel cannot be used because it was not found in the list of reserved channels of the calling application.</td>
</tr>
</tbody>
</table>

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_USB BUS1. Depending on the result, a message will be shown to the user.

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

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

do
{
    // Reads the first message in the queue.
    result = CanTpApi.Read(CanTpApi.PCANTP_USB BUS1, 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 occurred");
        // Here can be decided if the loop has to be terminated
        bStop = HandleReadError(result);
    }
} while (!bStop);
```
C++/CLR

```c++
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 == TPCANTPStatus::PCANTP_ERROR_OK)
    {
        // Processes the received message.
        MessageBox::Show(“A message was received”);
        //ProcessMessage(msg);
    }
    else
    {
        // An error occurred
        MessageBox::Show(“An error occurred”);
        // Here can be decided if the loop has to be terminated.
        //bStop = HandleReadError(result);
    }
} while (!bStop);
```

Visual Basic

```vb
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 occurred”)
        ' Here can be decided if the loop has to be terminated.
        bStop = HandleReadError(result)
    End If
Loop While bStop = False
```

Pascal OO

```pascal
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 occurred', '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);
Visual Basic

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

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Message Buffer</td>
<td>A TPCANTPMsg buffer containing the CAN ISO-TP message to be sent.</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>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.</td>
</tr>
<tr>
<td>PCANTP_ERROR_WRONG_PARAM</td>
<td>The network addressing information of the message is not valid.</td>
</tr>
<tr>
<td>PCANTP_ERROR_NO_MEMORY</td>
<td>Failed to allocate memory and copy message in the transmission queue.</td>
</tr>
</tbody>
</table>

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#
```csharp
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 = TPCANTP_MESSAGE_DIAGNOSTIC;
request->IDTYPE = TPCANTP_ID_CAN_11BIT;
request->FORMAT = TPCANTP_FORMAT_EXTENDED;
request->SA = 0xf1;
request->TA = 0x7e;
request->TA_TYPE = TPCANTP_ADDRESSING_PHYSICAL;
request->RA = 0x00;

// The message is sent using the PCAN-USB.
result = CanTpApi::Write(CanTpApi::PCANTP_USBBUS1, *request);
if (result == TPCANTP_STATUS_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}\", result,
            request->MSGTYPE).ToString(), ((int)request->RESULT).ToString());
    } while (result == TPCANTP_STATUS_ERROR_NO_MESSAGE);
}
else
{
    // An error occurred.
    MessageBox::Show("Error occurred: " + result.ToString());
}
```

**Visual Basic**

```vbnet
' 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
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 := TPCANTP_MESSAGE_DIAGNOSTIC;
  request.IDTYPE := TPCANTP_ID_CAN_11BIT;
  request.FORMAT := TPCANTP_FORMAT_EXTENDED;
  request.SA := $f1;
  request.TA := $7e;
  request.TA_TYPE := TPCANTP_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

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

C#

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

C++ / CLR

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

Visual Basic

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

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
</tbody>
</table>

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#

```csharp
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 occurred");
}
else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset");
```

C++ / CLR

```cpp
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 occurred");
}
else
    MessageBox::Show("PCAN-PCI (Ch-1) was reset");
```

Visual Basic

```vbnet
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 occurred")
Else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset")
End If
```

Pascal OO

```pascal
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 occurred', '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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTP_Initialize</td>
<td>Initializes a PCANTP channel.</td>
</tr>
<tr>
<td>CANTP_Uninitialize</td>
<td>Uninitializes a PCANTP channel.</td>
</tr>
</tbody>
</table>

### Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTP_SetValue</td>
<td>Sets a configuration or information value within a PCANTP channel.</td>
</tr>
<tr>
<td>CANTP_AddMapping</td>
<td>Configures the ISO-TP mapping between a CAN ID and an ISO-TP network addressing information.</td>
</tr>
<tr>
<td>CANTP_RemoveMapping</td>
<td>Removes a previously configured ISO-TP mapping.</td>
</tr>
</tbody>
</table>

### Information

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTP_GetValue</td>
<td>Retrieves information from a PCANTP channel.</td>
</tr>
<tr>
<td>CANTP_GetStatus</td>
<td>Retrieves the current BUS status of a PCANTP channel.</td>
</tr>
<tr>
<td>CANTP_GetErrorText</td>
<td>Gets a descriptive text for an error code.</td>
</tr>
</tbody>
</table>

### Communication

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTP_Read</td>
<td>Reads a CAN message from the receive queue of a PCANTP channel.</td>
</tr>
<tr>
<td>CANTP_Write</td>
<td>Transmits a CAN message using a connected PCANTP channel.</td>
</tr>
<tr>
<td>CANTP_Reset</td>
<td>Resets the receive and transmit queues of a PCANTP channel.</td>
</tr>
</tbody>
</table>

#### 3.7.1 CANTP_Initialize

Initializes a PCANTP channel.

**Syntax**

C++

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Baudrate</td>
<td>The speed for the communication (see TPCANTPBaudrate on page 19)</td>
</tr>
<tr>
<td>HwType</td>
<td>The type of hardware (TPCANTPHWType)</td>
</tr>
<tr>
<td>IOPort</td>
<td>The I/O address for the parallel port.</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Interrupt number of the parallel port.</td>
</tr>
</tbody>
</table>
Returns
The return value is a TPCANTPStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_ALREADY_INITIALIZED</td>
<td>Indicates that the desired PCANTP channel is already in use.</td>
</tr>
<tr>
<td>PCANTP_ERROR_CAN_ERROR</td>
<td>This error flag states that the error is composed of a more precise PCAN-Basic error.</td>
</tr>
</tbody>
</table>

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).

```cpp
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++

```c
TPCANTPStatus __stdcall CANTP_InitializeFD(
    TPCANTPHandle channel,
    TPCANPBitrageFD BitrateFD
);
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>BitrateFD</td>
<td>The speed for the communication (see TPCANPBitrageFD on page 21)</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_ALREADY_INITIALIZED</td>
<td>Indicates that the desired PCANTP channel is already in use.</td>
</tr>
<tr>
<td>PCANTP_ERROR_CAN_ERROR</td>
<td>This error flag states that the error is composed of a more precise PCAN-Basic error.</td>
</tr>
</tbody>
</table>

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 uninitialized processes for a Plug and Play channel (channel 2 of a PCAN-PCI hardware).

C++

```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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
</tbody>
</table>

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++

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

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to be set (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>Buffer</td>
<td>The buffer containing the numeric value to be set.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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 Definitions. 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++

```cpp
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**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>A TPCANTPStatus error code</td>
</tr>
<tr>
<td>Language</td>
<td>Indicates a &quot;Primary Language ID&quot;</td>
</tr>
<tr>
<td>StringBuffer</td>
<td>A buffer for a null-terminated char array</td>
</tr>
</tbody>
</table>

**Returns**

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

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_WRONG_PARAM</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Primary Language ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral (System dependant)</td>
<td>00h (0)</td>
</tr>
<tr>
<td>English</td>
<td>09h (9)</td>
</tr>
<tr>
<td>German</td>
<td>07h (7)</td>
</tr>
<tr>
<td>French</td>
<td>0Ch (12)</td>
</tr>
<tr>
<td>Italian</td>
<td>10h (16)</td>
</tr>
<tr>
<td>Spanish</td>
<td>0Ah (10)</td>
</tr>
</tbody>
</table>
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++

```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 occurred", "Error", MB_OK);
}
else
    MessageBox(NULL, strMsg, "Success", MB_OK);
```

See also: Primary Language ID

### 3.7.6 CANTP_AddMapping

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

**Syntax**

C++

```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**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>canID</td>
<td>The CAN Identifier to be mapped.</td>
</tr>
<tr>
<td>canIDResponse</td>
<td>The CAN Identifier that is used to respond to a request with the CAN Id 'canId'.</td>
</tr>
<tr>
<td>canIdType</td>
<td>The type of CAN identifier</td>
</tr>
<tr>
<td>formatType</td>
<td>The format type of the ISO-TP network addressing information</td>
</tr>
<tr>
<td>msgType</td>
<td>The ISO-TP message type</td>
</tr>
<tr>
<td>sourceAddr</td>
<td>The source address</td>
</tr>
<tr>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>targetAddr</td>
<td>The target address</td>
</tr>
<tr>
<td>targetType</td>
<td>The type of the target</td>
</tr>
<tr>
<td>remoteAddr</td>
<td>The remote address</td>
</tr>
</tbody>
</table>

**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_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.

<table>
<thead>
<tr>
<th>FormatType parameter</th>
<th>Valid canIdType parameter</th>
<th>Valid msgType parameter</th>
<th>Valid targetType parameter</th>
<th>Valid remoteAddr parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_FORMAT_NORMAL</td>
<td>Any</td>
<td>PCANTP_MESSAGE_DIAGNOSTIC</td>
<td>Any values</td>
<td>0x00 (value is ignored)</td>
</tr>
<tr>
<td>PCANTP_FORMAT_EXTENDED</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>PCANTP_FORMAT_MIXED</td>
<td>PCANTP_ID_CAN_11BIT</td>
<td>PCANTP_MESSAGE_REMOTE_DIAGNOSTIC</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>

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++**

```cpp
TPCANTPHandle CanChannel = PCANTP_USB0BUS1;
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++

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

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>canID</td>
<td>The CAN Identifier that identifies the mapping to remove.</td>
</tr>
</tbody>
</table>

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++

```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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to retrieve (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>Buffer</td>
<td>The buffer to return the required numeric value.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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 Definitions 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>PCANTP_ERROR_OK</th>
<th>Indicates that the status of the given PCANTP channel is OK.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_BUSLIGHT</td>
<td>Indicates a bus error within the given PCANTP channel. The hardware is in bus-light status.</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSHEAVY</td>
<td>Indicates a bus error within the given PCANTP channel. The hardware is in bus-heavy status.</td>
</tr>
<tr>
<td>PCANTP_ERROR_BUSOFF</td>
<td>Indicates a bus error within the given PCANTP channel. The hardware is in bus-off status.</td>
</tr>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>Indicates that the given PCANTP channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.</td>
</tr>
</tbody>
</table>

Remarks

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

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

Example

The following example shows the use of the function 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 ValueDefinitions 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Message Buffer</td>
<td>A TPCANTPMsg buffer to store the CAN ISO-TP message.</td>
</tr>
<tr>
<td>TimestampBuffer</td>
<td>A TPCANTTimestamp buffer to get the reception time of the message.</td>
</tr>
</tbody>
</table>
Returns
The return value is a TPCANTPStatus code. PCANTP_ERROR_OK is returned on success. The typical errors in case of failure are:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCANTP_ERROR_NO_MESSAGE</td>
<td>Indicates that the receive queue of the channel is empty.</td>
</tr>
<tr>
<td>PCANTP_ERROR_NOT_INITIALIZED</td>
<td>Indicates that the given PCANTP channel cannot be uninitialized because it was not found in the list of reserved channels of the calling application.</td>
</tr>
</tbody>
</table>

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++

```cpp
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 occurred", "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++

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

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>MessageBuffer</td>
<td>A TPCANTPMsg buffer containing the CAN ISO-TP message to be sent.</td>
</tr>
</tbody>
</table>

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++

```cpp
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 occurred: %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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanChannel</td>
<td>The handle of a PCANTP channel (see TPCANTPHandle on page 16)</td>
</tr>
<tr>
<td>Parameter</td>
<td>The code of the value to be set (see TPCANTPParameter on page 33)</td>
</tr>
<tr>
<td>NumericBuffer</td>
<td>The buffer containing the string value to be set.</td>
</tr>
<tr>
<td>BufferLength</td>
<td>The length in bytes of the given buffer.</td>
</tr>
</tbody>
</table>

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++

```cpp
TPCANTPStatus result;

// The PCI channel is reset.
result = CANTP_Reset(PCANTP_PCIEBUS1);
if (result != PCANTP_ERROR_OK)
{
   // An error occurred.
   MessageBox(NULL, "An error occurred", "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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCAN-ISO-TP Handle Definitions</td>
<td>Defines the handles for the different PCAN channels.</td>
</tr>
<tr>
<td>Parameter Value Definitions</td>
<td>Defines the possible values for setting and getting PCAN's environment information with the functions CANTP_SetValue and CANTP_GetValue.</td>
</tr>
</tbody>
</table>

### 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:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_NONEBUS</td>
<td>0x0</td>
<td>Undefined/default value for a PCAN-ISO-TP channel</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_DNGBUS1</td>
<td>0x31</td>
<td>PCAN-Dongle/LPT interface, channel 1</td>
</tr>
</tbody>
</table>

**Handles for the PCI bus:**

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

**Handles for the USB Bus:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_USBBUS1</td>
<td>0x51</td>
<td>PCAN-USB interface, channel 1</td>
</tr>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_USBBUS2</td>
<td>0x52</td>
<td>PCAN-USB interface, channel 2</td>
</tr>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_USBBUS3</td>
<td>0x53</td>
<td>PCAN-USB interface, channel 3</td>
</tr>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_USBBUS4</td>
<td>0x54</td>
<td>PCAN-USB interface, channel 4</td>
</tr>
</tbody>
</table>
### Handles for the PC Card Bus:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_PCCBUS1</td>
<td>0x61</td>
<td>PCAN-PC Card interface, channel 1</td>
</tr>
<tr>
<td>TPCANTPHandle</td>
<td>PCANTP_PCCBUS2</td>
<td>0x62</td>
<td>PCAN-PC Card interface, channel 2</td>
</tr>
</tbody>
</table>

### 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 means, 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 Definitions below.

### 3.8.2 Parameter Value Definitions

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

#### Debug configuration values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int32</td>
<td>PCANTP_DEBUG_NO</td>
<td>0</td>
<td>No CAN debug messages are being generated.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_DEBUG_CAN</td>
<td>1</td>
<td>CAN debug messages are written to the stdout output.</td>
</tr>
</tbody>
</table>

#### Channel values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int32</td>
<td>PCANTP_CHANNEL_UNAVAIL</td>
<td>0</td>
<td>The ISO-TP PCAN channel handle is illegal, or its associated hardware is not available.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_CHANNEL_AVAIL</td>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_CHANNEL_OCCUPIED</td>
<td>2</td>
<td>The ISO-TP PCAN channel handle is valid and is currently being used.</td>
</tr>
</tbody>
</table>
ISO-TP WaitForFrame parameter values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int32</td>
<td>PCANTP_WFT_MAX_UNLIMITED</td>
<td>0x00</td>
<td>Disables checks for ISO-TP WaitForFrames overrun when receiving a FlowControl frames (PCANTP_N_WFT_OVRN error will never occur).</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_WFT_MAX_DEFAULT</td>
<td>0x10</td>
<td>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.</td>
</tr>
</tbody>
</table>

ISO-TP message pending indication values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int32</td>
<td>PCANTP_MSG_PENDING_HIDE</td>
<td>0x00</td>
<td>Messages with the type PCANTP_MESSAGE_INDICATION will be automatically removed from the result of the CANTP_Read function.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_MSG_PENDING_SHOW</td>
<td>0x01</td>
<td>Messages with the type PCANTP_MESSAGE_INDICATION can be retrieved from the CANTP_Read function.</td>
</tr>
</tbody>
</table>

ISO-TP data padding values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int32</td>
<td>PCANTP_CAN_DATA_PADDING_NONE</td>
<td>0x00</td>
<td>CAN frame data optimization is enabled.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_CAN_DATA_PADDING_ON</td>
<td>0x01</td>
<td>CAN frame data optimization is disabled: CAN data length is always 8 and data is padded with zeros.</td>
</tr>
</tbody>
</table>

ISO-TP CAN segmented values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int32</td>
<td>PCANTP_CAN_UNSEGMENTED_OFF</td>
<td>0x00</td>
<td>Reception of unformatted (NON-ISO-TP) CAN frames is disabled.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_CAN_UNSEGMENTED_ON</td>
<td>0x01</td>
<td>Reception of unformatted (NON-ISO-TP) CAN frames is enabled.</td>
</tr>
<tr>
<td>Int32</td>
<td>PCANTP_CAN_UNSEGMENTED_ALL_FRAMES</td>
<td>0x02</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>PCANTP_BR_CLOCK</td>
<td>&quot;f_clock&quot;</td>
<td>Clock frequency in Hertz (800000000, 600000000, 400000000, 300000000, 240000000, 200000000)</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_CLOCK_MHZ</td>
<td>&quot;f_clock_mhz&quot;</td>
<td>Clock frequency in Megahertz (80, 60, 40, 30, 24, 20)</td>
</tr>
</tbody>
</table>
Nominal bit rate parameters:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>PCANTP_BR_NOM_BRP</td>
<td>&quot;nom_brp&quot;</td>
<td>Clock prescaler for nominal time quantum (1..1024).</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_NOM_TSEG1</td>
<td>&quot;nom_tseg1&quot;</td>
<td>TSEG1 segment for nominal bit rate in time quanta (1..256).</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_NOM_TSEG2</td>
<td>&quot;nom_tseg2&quot;</td>
<td>TSEG2 segment for nominal bit rate in time quanta (1..128).</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_NOM_SJW</td>
<td>&quot;nom_sjw&quot;</td>
<td>Synchronization Jump Width for nominal bit rate in time quanta (1..128).</td>
</tr>
</tbody>
</table>

Data bit rate parameters:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>PCANTP_BR_DATA_BRP</td>
<td>&quot;data_brp&quot;</td>
<td>Clock prescaler for fast data time quantum (1..1024).</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_DATA_TSEG1</td>
<td>&quot;data_tseg1&quot;</td>
<td>TSEG1 segment for fast data bit rate in time quanta (1..32).</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_DATA_TSEG2</td>
<td>&quot;data_tseg2&quot;</td>
<td>TSEG2 segment for fast data bit rate in time quanta (1..16).</td>
</tr>
<tr>
<td>String</td>
<td>PCANTP_BR_DATA_SJW</td>
<td>&quot;data_sjw&quot;</td>
<td>Synchronization Jump Width for fast data bit rate in time quanta (1..16).</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_clock</td>
<td>[800000000, 600000000, 400000000, 300000000, 240000000, 200000000]</td>
</tr>
<tr>
<td>f_clock_mhz</td>
<td>[80, 60, 40, 30, 24, 20]</td>
</tr>
<tr>
<td>nom_brp</td>
<td>1 .. 1024</td>
</tr>
<tr>
<td>nom_tseg1</td>
<td>1 .. 256</td>
</tr>
<tr>
<td>nom_tseg2</td>
<td>1 .. 128</td>
</tr>
<tr>
<td>nom_sjw</td>
<td>1 .. 128</td>
</tr>
<tr>
<td>data_brp</td>
<td>1 .. 1024</td>
</tr>
<tr>
<td>data_tseg1</td>
<td>1 .. 32</td>
</tr>
<tr>
<td>data_tseg2</td>
<td>1 .. 16</td>
</tr>
<tr>
<td>data_sjw</td>
<td>1 .. 16</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>Topics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCAN Fundamentals</td>
<td>This section contains an introduction to PCAN.</td>
</tr>
<tr>
<td>PCAN-Basic</td>
<td>This section contains general information about the PCAN-Basic API.</td>
</tr>
<tr>
<td>PCAN-API</td>
<td>This section contains general information about the PCAN-API.</td>
</tr>
<tr>
<td>ISO-TP Network Addressing</td>
<td>This section contains general information about the ISO-TP network addressing format.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>PCAN-ISA</th>
<th>PCAN-Dongle</th>
<th>PCAN-PCI</th>
<th>PCAN-USB</th>
<th>PCAN-PC-Card</th>
<th>PCAN-LAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of channels</td>
<td>8</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Plug and Play hardware</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCAN-Dongle</td>
<td>No</td>
<td>Pcan_dng.sys</td>
</tr>
<tr>
<td>PCAN-ISA</td>
<td>No</td>
<td>Pcan_isa.sys</td>
</tr>
<tr>
<td>PCAN-PC/104</td>
<td>No</td>
<td>Pcan_isa.sys</td>
</tr>
<tr>
<td>PCAN-PCI</td>
<td>Yes</td>
<td>Pcan_pci.sys</td>
</tr>
<tr>
<td>PCAN-PCI Express</td>
<td>Yes</td>
<td>Pcan_pci.sys</td>
</tr>
<tr>
<td>PCAN-cPCI</td>
<td>Yes</td>
<td>Pcan_pci.sys</td>
</tr>
<tr>
<td>PCAN-miniPCI</td>
<td>Yes</td>
<td>Pcan_pci.sys</td>
</tr>
<tr>
<td>PCAN-PC/104-Plus</td>
<td>Yes</td>
<td>Pcan_pci.sys</td>
</tr>
<tr>
<td>PCAN-USB</td>
<td>Yes</td>
<td>Pcan_usb.sys</td>
</tr>
<tr>
<td>PCAN-USB Pro</td>
<td>Yes</td>
<td>Pcan_usb.sys</td>
</tr>
<tr>
<td>PCAN-PC Card</td>
<td>Yes</td>
<td>Pcan_pcc.sys</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseAll</td>
<td>Disconnects all hardware, nets, and clients.</td>
</tr>
<tr>
<td>RegisterHardware</td>
<td>Registers a non-Plug and Play CAN hardware.</td>
</tr>
<tr>
<td>RegisterHardwarePCI</td>
<td>Registers a PCI CAN hardware.</td>
</tr>
<tr>
<td>RegisterNet</td>
<td>Registers a PCAN net.</td>
</tr>
<tr>
<td>RemoveHardware</td>
<td>Removes and deactivates CAN hardware.</td>
</tr>
<tr>
<td>RemoveNet</td>
<td>Removes a PCAN net.</td>
</tr>
</tbody>
</table>

Fields configuration, configuration functions for nets and hardware:

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

Fields client, functions for the management of the clients:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectToNet</td>
<td>Connects a client to a PCAN net.</td>
</tr>
<tr>
<td>DisconnectFromNet</td>
<td>Disconnects a client from a PCAN net.</td>
</tr>
<tr>
<td>RegisterClient</td>
<td>Registers an application as PCAN client.</td>
</tr>
<tr>
<td>RegisterMsg</td>
<td>Expands the reception filter of a client.</td>
</tr>
<tr>
<td>RemoveAllMsgs</td>
<td>Resets the filter of a client for a connected net.</td>
</tr>
<tr>
<td>RemoveClient</td>
<td>Removes a client from the driver.</td>
</tr>
<tr>
<td>ResetClient</td>
<td>Resets the receive and transmit queue of a client.</td>
</tr>
<tr>
<td>ResetHardware</td>
<td>Resets a CAN hardware.</td>
</tr>
<tr>
<td>SetClientFilter</td>
<td>Configures the reception filter of a client.</td>
</tr>
<tr>
<td>SetClientFilterEx</td>
<td>Configures the reception filter of a client.</td>
</tr>
<tr>
<td>SetClientParam</td>
<td>Configures a client parameter, e.g.</td>
</tr>
<tr>
<td></td>
<td>- self-receive mode of transmitted messages</td>
</tr>
<tr>
<td></td>
<td>- improves the accuracy of the reception filter.</td>
</tr>
</tbody>
</table>
Fields communication, functions for the data interchange over the CAN bus:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Reads a received CAN message, including the reception time stamp.</td>
</tr>
<tr>
<td>Read-Multi</td>
<td>Reads multiple received CAN messages.</td>
</tr>
<tr>
<td>Write</td>
<td>Transmits a CAN message at a specified time.</td>
</tr>
</tbody>
</table>

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

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

### 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:

<table>
<thead>
<tr>
<th>Addressing format</th>
<th>CAN ID length</th>
<th>Mandatory configuration steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal addressing</td>
<td>11 bits</td>
<td>Defines mappings with CANTP_AddMapping.</td>
</tr>
<tr>
<td>PCANTP_FORMAT_NORMAL</td>
<td>29 bits</td>
<td>Defines mappings with CANTP_AddMapping.</td>
</tr>
<tr>
<td>Normal fixed addressing</td>
<td>11 bits</td>
<td>Addressing is invalid.</td>
</tr>
<tr>
<td>PCANTP_FORMAT_FIXED_NORMAL</td>
<td>29 bits</td>
<td>-</td>
</tr>
<tr>
<td>Extended addressing</td>
<td>11 bits</td>
<td>Defines mappings with CANTP_AddMapping.</td>
</tr>
<tr>
<td>PCANTP_FORMAT_EXTENDED</td>
<td>29 bits</td>
<td>Defines mappings with CANTP_AddMapping.</td>
</tr>
<tr>
<td>Mixed addressing</td>
<td>11 bits</td>
<td>Defines mappings with CANTP_AddMapping.</td>
</tr>
<tr>
<td>PCANTP_FORMAT_MIXED</td>
<td>29 bits</td>
<td>-</td>
</tr>
<tr>
<td>Enhanced addressing</td>
<td>11 bits</td>
<td>Addressing is invalid.</td>
</tr>
<tr>
<td>PCANTP_FORMAT_ENHANCED</td>
<td>29 bits</td>
<td>-</td>
</tr>
</tbody>
</table>

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.