ISO-TP 3.x API

API Implementation of the ISO-TP 2016 Standard (ISO 15765-2)

Documentation

Document version 1.8.0 (2021-10-20)

PEAK-System Technik GmbH
Otto-Roehm-Strasse 69
64293 Darmstadt
Germany

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

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

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

Document version 1.8.0 (2021-10-20)

Contents

1 PCAN-ISO-TP API

Welcome to the documentation of ISO-TP 3.x API, a PEAK CAN API that implements ISO 15765-2:2016, or ISO-TP 2016, an international standard for sending data packets over a CAN bus.

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

2 Introduction

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

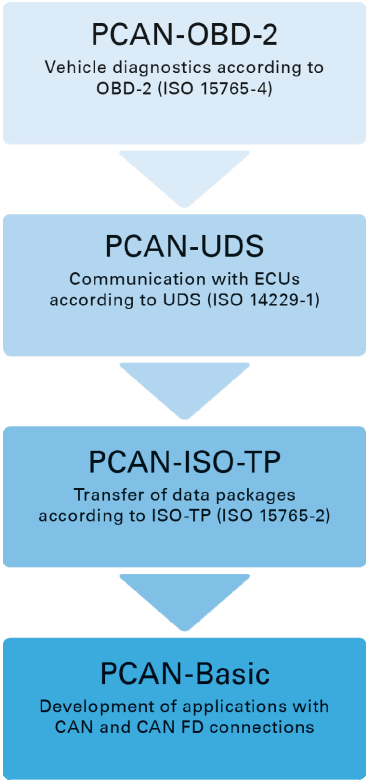
2.1 Understanding PCAN-ISO-TP

ISO-TP is an international standard for sending data packets over a CAN bus. The protocol defines data segmentation that allows to transmit messages which cannot be transmitted with a single CAN frame, the maximum data length that can be transmitted in a single data block is 4 Gigabytes.

The exchange of data between nodes (e.g. from Electronic Control Units (ECU) to ECU, or between an 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 15765-2:2016 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.

ISO-TP 3.x API is an implementation of the ISO-TP 2016 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_2016 (**class method version:** Initialize_2016). 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_2016 (**class method version:** AddMapping_2016).

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_2016 and CANTP_Write_2016 (**class method versions:** Read_2016 and Write_2016) can be then used to read and write CAN/CAN-FD frames and CAN/CAN-FD ISO-TP messages. 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_2016 (**class method version:** Uninitialize_2016) 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 Backward Compatibility Notes

Until version 2.x, the PCAN-ISO-TP API implemented the protocol ISO-TP as described in the ISO norm 15765-2, revision 2004. Support for the ISO norm 15765-2, revision 2016, was introduced with version 3.0 of the API PCAN-ISO-TP. New features of the norm 15765-2 caused changes in the API, that are to be considered when porting projects written with older versions of the PCAN-ISO API.

2.3.1 Binary Compatibility

The new `PCAN-ISO-TP.dll`, version 3.x, is full compatible with applications written using earlier versions of the API. Existing applications don't need to be rebuilt when updating the `PCAN-ISO-TP.dll`.

2.3.2 Code Compatibility

Both revisions of the ISO norm 15765-2 mentioned before are visually split into different header files for the supported programming languages, `PCAN-ISO-TP_2004.*` and `PCAN-ISO-TP_2016.*` (for Delphi `PCANTP_2004.pas`, and `PCANTP_2016.pas`). Additionally, for the C++ language, a header file called the same as in the previous API version, PCAN-ISO-TP.h, is now used for backward compatibility, making easier the update of existing C++ projects.

All these files are included in the PCAN-ISO-TP package under the folder "Include" (header files with suffix _2016) and its subfolder "Backward Compatibility" (other header files).

Depending on the needs of a developer and his project, he can:

1. **Start a new ISO-TP:2016 project (recommended)**

   The header file called `PCAN-ISO-TP_2016.*` (for Delphi, `PCANTP_2016.pas`) is to be used. No other ISO-TP header file is needed.

   Note on .NET and Delphi: These ISO-TP headers need information defined in the PCAN-Basic API, which is not part of the PCAN-ISO-TP Package. The PCAN-Basic package must be downloaded separately and the corresponding PCANBasic.* header file must be copied and added to the project. Otherwise the project will not compile.

2. **Start a new ISO-TP:2004 project**

   The header file called PCAN-ISO-TP_2004.* (for Delphi `PCANTP_2004.pas`) is to be used. No other ISO-TP header file is needed. The content of this file represents the content of the file `PCAN-ISO-TP.*` (for Delphi, `PCANTP.pas`)" from the version 2.x of the PCAN-ISO-TP API.

   Note on C++: This header doesn't include the required dependency "Window.h". It may be needed to include this reference manually, depending on the configuration of the project.

3. **Updating an existing project based on version 2.x of the PCAN-ISO-TP API**

   <u>C++ projects:</u>

1. Copy all 3 header files, `PCAN-ISO-TP.h`, `PCAN-ISO-TP_2004.h`, and `PCAN-ISO-TP_2016.h` into the project folder.

The header file called `PCAN-ISO-TP.h` will be replaced by the new backward-compatible header file of the same name. This header ensures that the old interface, i.e. all function prototypes from the PCAN-ISO-TP 2.x API, and also the new interface introduced with version 3.0, are available for the project.

2. Open and re-compile the project / solution.

<u>.NET and Delphi projects (Support for the revision 2004 only):</u>

1. Copy the header file `PCAN-ISO-TP_2004.*` (for Delphi, `PCANTP_2004.pas`) into the project folder.
2. Load the project / solution.
3. Exclude the file `PCAN-ISO-TP.*` (for Delphi, `PCANTP.pas`) from the project
4. Include the file `PCAN-ISO-TP_2004.*` (for Delphi, `PCANTP_2004.pas`) to the project.
5. Save and compile the project / solution.

<u>.NET and Delphi projects (Support for both revisions, 2004 and 2016):</u>

1. Copy the header files `PCAN-ISO-TP_2004.*`, and `PCAN-ISO-TP_2016.*` (for Delphi, `PCANTP_2004.pas` and `PCANTP_2016.pas`) into the project folder.
2. Download the PCAN-Basic package.
3. Locate the header file `PCANBasic.*` that corresponds to the project being updated, and copy it into the project folder.
4. Load the project / solution.
5. Exclude the file `PCAN-ISO-TP.*` (for Delphi, `PCANTP.pas`) from the project
6. Include the files `PCAN-ISO-TP_2004.*`, `PCAN-ISO-TP_2016.*` (for Delphi, `PCANTP_2004.pas` and `PCANTP_2016.pas`), and `PCANBasic.*` to the project.
7. Edit the file `PCAN-ISO-TP_2016.*` (for Delphi, `PCANTP_2016.pas`), and enable the commented define directive for "PCANTP_API_COMPATIBILITY_ISO_2004", within the description of the header file.
8. Save and compile the project / solution.

2.4 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.

2.5 Features

- Implementation of the ISO-TP protocol (ISO 15765-2:2016) for the transfer of data packages up to 4 Gigabytes 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

2.6 System Requirements

- Windows 10, 8.1 (32/64-bit)
  Note: Since Windows 7 is deprecated, this version is not supported anymore.
- 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

2.7 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 ISO-TP 3.x API.

3.1 Namespaces

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

Namespaces

	Name	Description
{}	Peak	Contains all namespaces that are part of the managed programming environment from PEAK-System.
{}	Peak.Can	Contains types and classes for using the PCAN API from PEAK-System.
{}	Peak.Can.Light	Contains types and classes for using the PCAN-Light API from PEAK-System.
{}	Peak.Can.Basic	Contains types and classes for using the PCAN-Basic API from PEAK-System.
{}	Peak.Can.Ccp	Contains types and classes for using the CCP API implementation from PEAK-System.
{}	Peak.Can.Xcp	Contains types and classes for using the XCP API implementation from PEAK-System.
{}	Peak.Can.IsoTp	Contains types and classes for using the PCAN-ISO-TP API implementation from PEAK-System.
{}	Peak.Can.Uds	Contains types and classes for using the PCAN-UDS API implementation from PEAK-System.
{}	Peak.Can.ObdII	Contains types and classes for using the PCAN-OBD-2 API implementation from PEAK-System.
{}	Peak.Lin	Contains types and classes used to handle with LIN devices from PEAK-System.
{}	Peak.RP1210A	Contains types and classes used to handle with CAN devices from PEAK-System through the TMC Recommended Practices 1210, version A, as known as RP1210(A).

3.1.1 Peak.Can.IsoTp

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

Remarks

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

Aliases

	Alias	Description
	cantp_pcanstatus	Represents the PCAN error and status codes (used when cantp_status encapsulates a PCAN-Basic error).
	cantp_bitrate	Represents a PCAN-FD bit rate string.
	cantp_timestamp	Defines a timestamp of a CANTP message.

Classes

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

Structures

	Class	Description
	cantp_msgoption	Represents message's options to override.
	cantp_msgoption_list	Represents a list of message's options to override.
	cantp_can_info	Represents common CAN information.
	cantp_netaddrinfo	Represents the network address information of an ISO-TP message.
	cantp_mapping	Represents a mapping between an ISO-TP network address information and a CAN ID.
	cantp_msgdata	Represents the content of a generic message.
	cantp_msgdata_can	Represents the content of a standard CAN frame.
	cantp_msgdata_canfd	Represents the content of a CAN FD frame.
	cantp_msgdata_isotp	Represents the content of an ISO-TP message.
	cantp_msg	Defines a CANTP message. A CANTP message encapsulates the content of a CAN frame, a CAN-FD frame, or an ISO-TP message. The members of this structure are sequentially byte aligned.
	cantp_msgprogress	Holds information on the communication progress of a message.

Enumerations

	Name	Description
	cantp_handle	Represents a PCAN-ISO-TP channel handle.
	cantp_baudrate	Represents the baudrate register value for the PCANTP channel.
	cantp_hwtype	Type of PCAN (non plug-n-play) hardware.
	cantp_device	Represents a PCAN device.
	cantp_statustype	Represents each group of errors a status can hold.
	cantp_netstatus	Represents the network result of the communication of an ISO-TP message (used in cantp_status).
	cantp_busstatus	Represents the status of a CAN bus (used in cantp_status).
	cantp_errstatus	Represents a general error (used in cantp_status).
	cantp_infostatus	Represents additional status information (used in cantp_status).
	cantp_status	Represent the PCANTP error and status codes.
	cantp_parameter	List of parameters handled by PCAN-ISO-TP (rev. 2016). PCAN-Basic parameters (PCAN_PARAM_xxx) are compatible via casting.
	cantp_msgtype	Represents the type of a CANTP message.
	cantp_msgflag	Represents the flags common to all types of cantp_msg.
	cantp_can_msgtype	Represents the flags of a CAN or CAN FD frame.
	cantp_isotp_msgtype	Represents the type of an ISO-TP message.
	cantp_isotp_format	Represents the addressing format of an ISO-TP message.
	cantp_isotp_addressing	Represents the type of target of an ISO-TP message.
	cantp_option	Represents the options of a message.
	cantp_msgprogress_state	Represents the status for a message whose transmission is in progress.
	cantp_msgdirection	Represents the direction of a message's communication.

3.2 Units

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

Namespaces

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

3.2.1 PCANTP Unit

The PCANTP Unit contains types and classes to use the PCAN-ISO-TP 2016 API within Delphi's programming environment and handle PCAN devices from PEAK-System.

Remarks

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

Aliases

	Alias	Description
	cantp_pcanstatus	Represents the PCAN error and status codes (used when cantp_status encapsulates a PCAN-Basic error).
	cantp_bitrate	Represents a PCAN-FD bit rate string.
	cantp_timestamp	Defines a timestamp of a CANTP message.

Classes

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

Structures

	Name	Description
	cantp_msgoption	Represents message's options to override.
	cantp_msgoption_list	Represents a list of message's options to override.
	cantp_can_info	Represents common CAN information.
	cantp_netaddrinfo	Represents the network address information of an ISO-TP message.
	cantp_mapping	Represents a mapping between an ISO-TP network address information and a CAN ID.
	cantp_msgdata	Represents the content of a generic message.
	cantp_msgdata_can	Represents the content of a standard CAN frame.
	cantp_msgdata_canfd	Represents the content of a CAN FD frame.
	cantp_msgdata_isotp	Represents the content of an ISO-TP message.
	cantp_msg	Defines a CANTP message. A CANTP message encapsulates the content of a CAN frame, a CAN-FD frame or an ISO-TP message. The members of this structure are sequentially byte aligned.
	cantp_msgprogress	Holds information on the communication progress of a message.

Enumerations

	Name	Description
	cantp_handle	Represents a PCAN-ISO-TP channel handle.
	cantp_baudrate	Represents the baudrate register value for the PCANTP channel.
	cantp_hwtype	Type of PCAN (non plug-n-play) hardware.
	cantp_device	Represents a PCAN device.
	cantp_statustype	Represents each group of errors a status can hold.

	Name	Description
	cantp_netstatus	Represents the network result of the communication of an ISO-TP message (used in cantp_status).
	cantp_busstatus	Represents the status of a CAN bus (used in cantp_status).
	cantp_errstatus	Represents a general error (used in cantp_status).
	cantp_infostatus	Represents additional status information (used in cantp_status).
	cantp_status	Represent the PCANTP error and status codes.
	cantp_parameter	List of parameters handled by PCAN-ISO-TP (rev. 2016). PCAN-Basic parameters (PCAN_PARAM_xxx) are compatible via casting.
	cantp_msgtype	Represents the type of a CANTP message.
	cantp_msgflag	Represents the flags common to all types of cantp_msg.
	cantp_can_msgtype	Represents the flags of a CAN or CAN FD frame.
	cantp_isotp_msgtype	Represents the type of an ISO-TP message.
	cantp_isotp_format	Represents the addressing format of an ISO-TP message.
	cantp_isotp_addressing	Represents the type of target of an ISO-TP message.
	cantp_option	Represents the options of a message.
	cantp_msgprogress_state	Represents the status for a message whose transmission is in progress.
	cantp_msgdirection	Represents the direction of a message's communication.

3.3 Classes

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

Classes

	Class	Description
	CanTpApi	Defines a class to use the PCAN-ISO-TP 2016 API within the Microsoft's .NET Framework programming environment.
	TCanTpApi	Defines a class to use the PCAN-ISO-TP 2016 API within the Delphi programming environment.

3.3.1 CanTpApi

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

Syntax

C#

```
public static class CanTpApi
```

C++ / CLR

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

Visual Basic

```
Public NotInheritable Class CanTpApi
```

Remarks

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

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

```
cantp_status res;
// Static use without any instance.
//
res = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
        cantp_baudrate.PCANTP_BAUDRATE_500K);
```

Note: This class under Delphi is called TCanTpApi.

See also: Methods on page 98, types on page 33.

3.3.2 TCanTpApi

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

Syntax

Pascal OO

```
TCanTpApi = class
```

Remarks

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

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

See also: Methods on page 98, Types on page 33

3.4 Structures

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

	Name	Description
	cantp_msgoption	Represents message's options to override.
	cantp_msgoption_list	Represents a list of message's options to override.
	cantp_can_info	Represents common CAN information.
	cantp_netaddrinfo	Represents the network addressing information of an ISO-TP message.
	cantp_mapping	Represents a mapping between an ISO-TP network addressing information and a CAN ID.
	cantp_msgdata	Represents the content of a generic message.
	cantp_msgdata_can	Represents the content of a standard CAN frame.
	cantp_msgdata_canfd	Represents the content of a CAN FD frame.
	cantp_msgdata_isotp	Represents the content of an ISO-TP message.
	cantp_msg	Defines a CANTP message. A CANTP message encapsulates the content of a CAN frame, a CAN-FD frame or an ISO-TP message. The members of this structure are sequentially byte aligned.
	cantp_msgprogress	Holds information on the communication progress of a message.

3.4.1 cantp_msgoption

Represents message's options to override.

Syntax

C++

```
typedef struct _cantp_msgoption {
	cantp_option name;
	uint32_t value;
} cantp_msgoption;
```

Pascal OO

```
cantp_msgoption = record
    name: cantp_option;
    value: UInt32;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgoption
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_option name;
    public UInt32 value;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_msgoption
{
	[MarshalAs(UnmanagedType::U4)]
	cantp_option name;
	UInt32 value;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msgoption
    <MarshalAs(UnmanagedType.U4)>
    Public name As cantp_option
    Public value As UInt32
End Structure
```

Fields

Name	Description
name	Name of the option (see cantp_msgoption on page 16).
value	Value of the option.

See also: cantp_msgoption_list below, cantp_option on page 93, cantp_msgprogress_state on page 95, CANTP_MsgDataInitOptions_2016 on page 254, **class method version:** MsgDataInitOptions_2016 on page 194, getOption_2016 on page 216, setOption_2016 on page 218

3.4.2 cantp_msgoption_list

Represents a list of message's options to override.

Syntax

C++

```
typedef struct _cantp_msgoptions {
       cantp_msgoption* buffer;
       uint32_t count;
} cantp_msgoption_list;
```

Pascal OO

```
  cantp_msgoption_list = record
    buffer: ^cantp_msgoption;
    count: UInt32;
  end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgoption_list
{
    public IntPtr buffer;
    public UInt32 count;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_msgoption_list
{
       cantp_msgoption *buffer;
       UInt32 count;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
```

```
Public Structure cantp_msgoption_list
    Public buffer As IntPtr
    Public count As UInt32
End Structure
```

Fields

Name	Description
buffer	Pointer to an array of cantp_msgoption. (see cantp_msgoption on page 16)
count	Number of options in the array.

See also: cantp_option on page 93, cantp_msgprogress_state on page 95, CANTP_MsgDataInitOptions_2016 on page 254, **class method version:** MsgDataInitOptions_2016 on page 194, getOption_2016 on page 216, setOption_2016 on page 218

3.4.3 cantp_can_info

Represents common CAN information.

Syntax

C++

```
typedef struct _cantp_can_info {
       uint32_t can_id;
       cantp_can_msgtype can_msgtype;
       uint8_t dlc;
} cantp_can_info;
```

Pascal OO

```
cantp_can_info = record
    can_id: UInt32;
    can_msgtype: cantp_can_msgtype;
    dlc: Byte;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_can_info
{
    public UInt32 can_id;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_can_msgtype can_msgtype;
    public byte dlc;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_can_info
{
       UInt32 can_id;
       [MarshalAs(UnmanagedType::U4)]
       cantp_can_msgtype can_msgtype;
       Byte dlc;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_can_info
    Public can_id As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public can_msgtype As cantp_can_msgtype
    Public dlc As Byte
End Structure
```

Fields

Name	Description
can_id	CAN identifier.
can_msgtype	Types and flags of the CAN/CAN-FD frame (see cantp_can_msgtype on page 88).
dlc	Data Length Code of the frame.

Remarks

Specifying a non-zero dlc value when writing an ISO-TP message will override the value “can_tx_dlc” of its corresponding mapping (if it exists) and the value of parameter PCANTP_PARAMETER_CAN_TX_DL.

See also: cantp_msg on page 28

3.4.4 cantp_netaddrinfo

Represents the network address information of an ISO-TP message.

Syntax

C++

```
typedef struct _cantp_netaddrinfo {
       cantp_isotp_msgtype msgtype;
       cantp_isotp_format format;
       cantp_isotp_addressing target_type;
       uint16_t source_addr;
       uint16_t target_addr;
       uint8_t extension_addr;
} cantp_netaddrinfo;
```

Pascal OO

```
  cantp_netaddrinfo = record
    msgtype: cantp_isotp_msgtype;
    format: cantp_isotp_format;
    target_type: cantp_isotp_addressing;
    source_addr: UInt16;
    target_addr: UInt16;
    extension_addr: Byte;
  end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_netaddrinfo
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_isotp_msgtype msgtype;
    [MarshalAs(UnmanagedType.U4)]
```

```
    public cantp_isotp_format format;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_isotp_addressing target_type;
    public UInt16 source_addr;
    public UInt16 target_addr;
    public byte extension_addr;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_netaddrinfo
{
        [MarshalAs(UnmanagedType::U4)]
        cantp_isotp_msgtype msgtype;
        [MarshalAs(UnmanagedType::U4)]
        cantp_isotp_format format;
        [MarshalAs(UnmanagedType::U4)]
        cantp_isotp_addressing target_type;
        UInt16 source_addr;
        UInt16 target_addr;
        Byte extension_addr;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_netaddrinfo
    <MarshalAs(UnmanagedType.U4)>
    Public msgtype As cantp_isotp_msgtype
    <MarshalAs(UnmanagedType.U4)>
    Public format As cantp_isotp_format
    <MarshalAs(UnmanagedType.U4)>
    Public target_type As cantp_isotp_addressing
    Public source_addr As UInt16
    Public target_addr As UInt16
    Public extension_addr As Byte
End Structure
```

Fields

Name	Description
msgtype	ISO-TP message type (see cantp_isotp_msgtype on page 90).
format	ISO-TP format addressing (see cantp_isotp_format on page 91).
target_type	ISO-TP addressing/target type (see cantp_isotp_addressing on page 92).
source_addr	Source address.
target_addr	Target address.
extension_addr	Extension address.

See also: cantp_msgdata on page 22 and cantp_mapping below

3.4.5 cantp_mapping

Represents a mapping between an ISO-TP network address information and a CAN ID.

Syntax

C++

```
typedef struct _cantp_mapping {
	uintptr_t uid;
	uint32_t can_id;
	uint32_t can_id_flow_ctrl;
	cantp_can_msgtype can_msgtype;
	uint8_t can_tx_dlc;
	cantp_netaddrinfo netaddrinfo;
} cantp_mapping;
```

Pascal OO

```
cantp_mapping = record
    uid: Pointer;
    can_id: UInt32;
    can_id_flow_ctrl: UInt32;
    can_msgtype: cantp_can_msgtype;
    can_tx_dlc: Byte;
    netaddrinfo: cantp_netaddrinfo;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_mapping
{
    public UIntPtr uid;
    public UInt32 can_id;
    public UInt32 can_id_flow_ctrl;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_can_msgtype can_msgtype;
    public byte can_tx_dlc;
    public cantp_netaddrinfo netaddrinfo;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_mapping
{
	UIntPtr uid;
	UInt32 can_id;
	UInt32 can_id_flow_ctrl;
	[MarshalAs(UnmanagedType::U4)]
	cantp_can_msgtype can_msgtype;
	Byte can_tx_dlc;
	cantp_netaddrinfo netaddrinfo;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_mapping
    Public uid As UIntPtr
    Public can_id As UInt32
    Public can_id_flow_ctrl As UInt32
    <MarshalAs(UnmanagedType.U4)>
    Public can_msgtype As cantp_can_msgtype
    Public can_tx_dlc As Byte
    Public netaddrinfo As cantp_netaddrinfo
End Structure
```

Fields

Name	Description
uid	Mapping's unique ID, read-only, set by CANTP_AddMapping_2016 function or AddMapping_2016 method.
can_id	CAN ID mapped to the Network Address Information.
can_id_flow_ctrl	CAN ID used for the flow control frame (frame, transmitted by the receiver of an ISO-TP message, that provides communication information).
can_msgtype	CAN frame msgtype. Only PCANTP_CAN_MSGTYPE_STANDARD or PCANTP_CAN_MSGTYPE_EXTENDED is mandatory (see also cantp_can_msgtype on page 88).
can_tx_dlc	Default maximum CAN DLC value to use with segmented messages. Value can be 8 or more if CAN FD communication is supported. If non-zero, this value will supersede the parameter PCANTP_PARAMETER_CAN_TX_DL for communications involving this mapping.
netaddrinfo	ISO-TP Network Address Information (see cantp_netaddrinfo on page 19).

Remarks

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

PCANTP ISO-TP format (see cantp_isotp_format on page 91)	Valid can identifier	Valid isotp message type parameter	Valid target addressing
PCANTP_ISOTP_FORMAT_NORMAL	11bits & 29bits	PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC	Any values
PCANTP_ISOTP_FORMAT_EXTENDED	11bits & 29bits	PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC	Any values
PCANTP_ISOTP_FORMAT_MIXED	11bits	PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC	Any values

When target type is “functional addressing” there is no need to define can_id_flow_ctrl, since responses from functional addressing will be physically addressed. The definition value PCANTP_MAPPING_FLOW_CTRL_NONE can be used to fill in the can identifier flow control fields in those cases.

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

See also: CANTP_AddMapping_2016 on page 229, **class method:** AddMapping_2016 on page 116.

3.4.6 cantp_msgdata

Represents the content of a generic message.

Syntax

C++

```
typedef struct _cantp_msgdata {
       cantp_msgflag flags;
       uint32_t length;
       uint8_t* data;
       cantp_netstatus netstatus;
       cantp_msgoption_list* options;
} cantp_msgdata;
```

Pascal OO

```
cantp_msgdata = record
    flags: cantp_msgflag;
    length: UInt32;
    data: ^Byte;
    netstatus: cantp_netstatus;
```

```
    options: ^cantp_msgoption_list;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgdata
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_msgflag flags;
    public UInt32 length;
    public IntPtr data;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_netstatus netstatus;
    public IntPtr options;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_msgdata
{
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgflag flags;
       UInt32 length;
       Byte *data;
       [MarshalAs(UnmanagedType::U4)]
       cantp_netstatus netstatus;
       cantp_msgoption_list *options;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msgdata
    <MarshalAs(UnmanagedType.U4)>
    Public flags As cantp_msgflag
    Public length As UInt32
    Public data As IntPtr
    <MarshalAs(UnmanagedType.U4)>
    Public netstatus As cantp_netstatus
    Public options As IntPtr
End Structure
```

Fields

Name	Description
flags	Structure specific flags. (see cantp_msgflag on page 87)
length	Length of the message.
data	Data of the message.
netstatus	Network status. (see cantp_netstatus on page 53)
options	Defines specific options to override global message configuration. (see cantp_msgoption_list on page 17)

See also: cantp_msg on page 28, setData_2016 on page 208, getData_2016 on page 212

3.4.7 cantp_msgdata_can

Represents the content of a standard CAN message.

Syntax

C++

```
typedef struct _cantp_msgdata_can {
       cantp_msgflag flags;
       uint32_t length;
       uint8_t* data;
       cantp_netstatus netstatus;
       cantp_msgoption_list* options;
       uint8_t data_max[PCANTP_MAX_LENGTH_CAN_STANDARD];
} cantp_msgdata_can;
```

Pascal OO

```
cantp_msgdata_can = record
    flags: cantp_msgflag;
    length: UInt32;
    data: ^Byte;
    netstatus: cantp_netstatus;
    options: ^cantp_msgoption_list;
    data_max: array [0 .. PCANTP_MAX_LENGTH_CAN_STANDARD - 1] of Byte;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgdata_can
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_msgflag flags;
    public UInt32 length;
    public IntPtr data;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_netstatus netstatus;
    public IntPtr options;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = CanTpApi.PCANTP_MAX_LENGTH_CAN_STANDARD)]
    public byte[] data_max;
}
```

C++/CLR

```
public struct cantp_msgdata_can
{
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgflag flags;
       UInt32 length;
       Byte *data;
       [MarshalAs(UnmanagedType::U4)]
       cantp_netstatus netstatus;
       cantp_msgoption_list *options;
       [MarshalAs(UnmanagedType::ByValArray, SizeConst = 8)]
       Byte data_max[];
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msgdata_can
    <MarshalAs(UnmanagedType.U4)>
    Public flags As cantp_msgflag
    Public length As UInt32
```

```
    Public data As IntPtr
    <MarshalAs(UnmanagedType.U4)>
    Public netstatus As cantp_netstatus
    Public options As IntPtr
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=CanTpApi.PCANTP_MAX_LENGTH_CAN_STANDARD)>
    Public data_max As Byte()
End Structure
```

Fields

Name	Description
flags	Structure specific flags (see cantp_msgflag on page 87).
length	Length of the message (0..8).
data	Data of the message (when initialized, pointer points to data_max field).
netstatus	Network status. (see cantp_netstatus on page 53).
options	Defines specific options to override global CAN configuration (not used yet, see cantp_msgoption_list on page 17).
data_max	Data of the message (data[0]..data[7]).

See also: cantp_msg on page 28 cantp_msgdata on page 22, setData_2016 on page 208, getData_2016 on page 212

3.4.8 cantp_msgdata_canfd

Represents the content of a CAN FD message.

Syntax

C++

```
typedef struct _cantp_msgdata_canfd {
       cantp_msgflag flags;
       uint32_t length;
       uint8_t* data;
       cantp_netstatus netstatus;
       cantp_msgoption_list* options;
       uint8_t data_max[PCANTP_MAX_LENGTH_CAN_FD];
} cantp_msgdata_canfd;
```

Pascal OO

```
cantp_msgdata_canfd = record
    flags: cantp_msgflag;
    length: UInt32;
    data: ^Byte;
    netstatus: cantp_netstatus;
    options: ^cantp_msgoption_list;
    data_max: array [0 .. PCANTP_MAX_LENGTH_CAN_FD - 1] of Byte;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgdata_canfd
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_msgflag flags;
    public UInt32 length;
    public IntPtr data;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_netstatus netstatus;
    public IntPtr options;
```

```
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = CanTpApi.PCANTP_MAX_LENGTH_CAN_FD)]
    public byte[] data_max;
}
```

C++/CLR

```
public struct cantp_msgdata_canfd
{
        [MarshalAs(UnmanagedType::U4)]
        cantp_msgflag flags;
        UInt32 length;
        Byte *data;
        [MarshalAs(UnmanagedType::U4)]
        cantp_netstatus netstatus;
        cantp_msgoption_list *options;
        [MarshalAs(UnmanagedType::ByValArray, SizeConst = 64)]
        Byte data_max[];
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msgdata_canfd
    <MarshalAs(UnmanagedType.U4)>
    Public flags As cantp_msgflag
    Public length As UInt32
    Public data As IntPtr
    <MarshalAs(UnmanagedType.U4)>
    Public netstatus As cantp_netstatus
    Public options As IntPtr
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=CanTpApi.PCANTP_MAX_LENGTH_CAN_FD)>
    Public data_max As Byte()
End Structure
```

Fields

Name	Description
flags	Structure specific flags (see cantp_msgflag on page 87).
length	Length of the message (0..64).
data	Data of the message (when initialized, pointer points to data_max field).
netstatus	Network status (see cantp_netstatus on page 53)
options	Defines specific options to override global CAN configuration (not used yet, see cantp_msgoption_list on page 17).
data_max	Data of the message (data[0]..data[63]).

See also: cantp_msg on page 28 cantp_msgdata on page 22, setData_2016 on page 208, getData_2016 on page 212

3.4.9 cantp_msgdata_isotp

Represents the content of an ISO-TP message.

Syntax

C++

```
typedef struct _cantp_msgdata_isotp {
        cantp_msgflag flags;
        uint32_t length;
        uint8_t* data;
        cantp_netstatus netstatus;
```

```
        cantp_msgoption_list* options;
        cantp_netaddrinfo netaddrinfo;
        cantp_isotp_info reserved;
} cantp_msgdata_isotp;
```

Pascal OO

```
cantp_msgdata_isotp = record
    flags: cantp_msgflag;
    length: UInt32;
    data: ^Byte;
    netstatus: cantp_netstatus;
    options: ^cantp_msgoption_list;
    netaddrinfo: cantp_netaddrinfo;
    reserved: cantp_isotp_info;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgdata_isotp
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_msgflag flags;
    public UInt32 length;
    public IntPtr data;
    [MarshalAs(UnmanagedType.U4)]
    public cantp_netstatus netstatus;
    public IntPtr options;
    public cantp_netaddrinfo netaddrinfo;
    public cantp_isotp_info reserved;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_msgdata_isotp
{
        [MarshalAs(UnmanagedType::U4)]
        cantp_msgflag flags;
        UInt32 length;
        Byte *data;
        [MarshalAs(UnmanagedType::U4)]
        cantp_netstatus netstatus;
        cantp_msgoption_list *options;
        cantp_netaddrinfo netaddrinfo;
        cantp_isotp_info reserved;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msgdata_isotp
    <MarshalAs(UnmanagedType.U4)>
    Public flags As cantp_msgflag
    Public length As UInt32
    Public data As IntPtr
    <MarshalAs(UnmanagedType.U4)>
    Public netstatus As cantp_netstatus
    Public options As IntPtr
    Public netaddrinfo As cantp_netaddrinfo
    Public reserved As cantp_isotp_info
End Structure
```

Fields

Name	Description
flags	Structure specific flags (see cantp_msgflag on page 87).
length	Length of the data.
data	Data of the message.
netstatus	Network status (see cantp_netstatus on page 53).
options	Defines specific options to override global CAN configuration(see cantp_msgoption_list on page 17).
netaddrinfo	ISO-TP network address information (see cantp_netaddrinfo on page 19).
reserved	Reserved ISO-TP information.

See also: cantp_msg below cantp_msgdata on page 22, setData_2016 on page 208, getData_2016 on page 212.

3.4.10 cantp_msg

A cantp_msg message is a generic CAN related message than can be either a standard CAN frame, a CAN FD frame, an ISO-TP message.

Syntax

C++

```
typedef struct _cantp_msg {
	cantp_msgtype type;
	cantp_msginfo reserved;
	cantp_can_info can_info;
	union {
		cantp_msgdata* any;
		cantp_msgdata_can* can;
		cantp_msgdata_canfd* canfd;
		cantp_msgdata_isotp* isotp;
	} msgdata;
} cantp_msg;
```

Pascal OO

```
cantp_msg = record
    typem: cantp_msgtype;
    reserved: cantp_msginfo;
    can_info: cantp_can_info;
    case Integer of
      0:
        (msgdata_any: ^cantp_msgdata;);
      1:
        (msgdata_can: ^cantp_msgdata_can;);
      2:
        (msgdata_canfd: ^cantp_msgdata_canfd;);
      3:
        (msgdata_isotp: ^cantp_msgdata_isotp;);
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msg
{
  [MarshalAs(UnmanagedType.U4)]
  public cantp_msgtype type;
  public cantp_msginfo reserved;
  public cantp_can_info can_info;
```

```
  private IntPtr msgdata;
  public IntPtr Msgdata
  {
    get { return msgdata; }
  }
  public cantp_msgdata Msgdata_any_Copy
  {
    get
    {
      return (cantp_msgdata)Marshal.PtrToStructure(msgdata, typeof(cantp_msgdata));
    }
  }
  public cantp_msgdata_can Msgdata_can_Copy
  {
    get
    {
      return (cantp_msgdata_can)Marshal.PtrToStructure(msgdata, typeof(cantp_msgdata_can));
    }
  }
  public cantp_msgdata_canfd Msgdata_canfd_Copy
  {
    get
    {
      return (cantp_msgdata_canfd)Marshal.PtrToStructure(msgdata, typeof(cantp_msgdata_canfd));
    }
  }
  public cantp_msgdata_isotp Msgdata_isotp_Copy
  {
    get
    {
      return (cantp_msgdata_isotp)Marshal.PtrToStructure(msgdata, typeof(cantp_msgdata_isotp));
    }
  }
}
```

C++/CLR

```
[StructLayout(LayoutKind::Explicit)]
public value struct cantp_msg_union_msgdata
{
       [FieldOffset(0)]
       cantp_msgdata *any;
       [FieldOffset(0)]
       cantp_msgdata_can *can;
       [FieldOffset(0)]
       cantp_msgdata_canfd *canfd;
       [FieldOffset(0)]
       cantp_msgdata_isotp *isotp;
};

[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_msg
{
public:
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgtype type;
       cantp_msginfo reserved;
       cantp_can_info can_info;
       cantp_msg_union_msgdata msgdata;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msg

  <MarshalAs(UnmanagedType.U4)>
  Public type As cantp_msgtype
  Public reserved As cantp_msginfo
  Public can_info As cantp_can_info
  Private _msgdata As IntPtr
  Public ReadOnly Property Msgdata() As IntPtr
    Get
      Return _msgdata
    End Get
  End Property

  Public ReadOnly Property Msgdata_any_Copy() As cantp_msgdata
    Get
      Return CType(Marshal.PtrToStructure(_msgdata, GetType(cantp_msgdata)), cantp_msgdata)
    End Get
  End Property

  Public ReadOnly Property Msgdata_can_Copy() As cantp_msgdata_can
    Get
      Return CType(Marshal.PtrToStructure(_msgdata, GetType(cantp_msgdata_can)),
          cantp_msgdata_can)
    End Get
  End Property

  Public ReadOnly Property Msgdata_canfd_Copy() As cantp_msgdata_canfd
    Get
      Return CType(Marshal.PtrToStructure(_msgdata, GetType(cantp_msgdata_canfd)),
          cantp_msgdata_canfd)
    End Get
  End Property

  Public ReadOnly Property Msgdata_isotp_Copy() As cantp_msgdata_isotp
    Get
      Return CType(Marshal.PtrToStructure(_msgdata, GetType(cantp_msgdata_isotp)),
          cantp_msgdata_isotp)
    End Get
  End Property
End Structure
```

Fields

Name	Description
type	Type of the message (see cantp_msgtype on page 85).
reserved	Reserved miscellaneous read-only information.
can_info	Common CAN information (see cantp_can_info on page 18).
msgdata.any	Shortcut to access msgdata as Generic content (see cantp_msgdata on page 22).
msgdata.can	Shortcut to access msgdata as CAN content (see cantp_msgdata_can on page 23).
msgdata.canfd	Shortcut to access msgdata as CAN-FD content (see cantp_msgdata_canfd on page 25).
msgdata.isotp	Shortcut to access msgdata as ISO-TP content (see cantp_msgdata_isotp on page 26).

Remarks

The cantp_msg structure is initialized and allocated by the PCAN-ISOTP API using CANTP_MsgDataAlloc_2016 / CANTP_MsgDataInit_2016 or CANTP_Read_2016 functions or class method equivalent MsgDataAlloc_2016 / MsgDataInit_2016 or Read_2016.

Once processed, the cantp_msg structure should be free using CANTP_MsgDataFree_2016 function or MsgDataFree_2016 method.

See also

Associated function:

CANTP_MsgDataAlloc_2016 on page 247, CANTP_MsgDataInit_2016 on page 248, CANTP_MsgDataFree_2016 on page 249, CANTP_MsgEqual_2016 on page 250, CANTP_MsgCopy_2016 on page 251, CANTP_MsgDlcToLength_2016 on page 252, CANTP_MsgLengthToDlc_2016 on page 253, CANTP_MsgDataInitOptions_2016 on page 254, CANTP_Read_2016 on page 243, CANTP_Write_2016 on page 245

Associated class method:

MsgDataAlloc_2016 on page 186, MsgDataInit_2016 on page 188, MsgDataFree_2016 on page 196, MsgEqual_2016 on page 177, MsgCopy_2016 on page 180, MsgDlcToLength_2016 on page 183, MsgLengthToDlc_2016 on page 184, MsgDataInitOptions_2016 on page 194, Read_2016 on page 160, Write_2016 on page 170

3.4.11 cantp_msgprogress

Holds information on the communication progress of a message.

Syntax

C++

```
typedef struct _cantp_msgprogress {
       cantp_msgprogress_state state;
       uint8_t percentage;
       cantp_msg* buffer;
} cantp_msgprogress;
```

Pascal OO

```
cantp_msgprogress = record
    state: cantp_msgprogress_state;
    percentage: Byte;
    buffer: ^cantp_msg;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct cantp_msgprogress
{
    [MarshalAs(UnmanagedType.U4)]
    public cantp_msgprogress_state state;
    public byte percentage;
    public IntPtr buffer;
}
```

C++/CLR

```
[StructLayout(LayoutKind::Sequential, Pack = 8)]
public value struct cantp_msgprogress
{
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgprogress_state state;
       Byte percentage;
       cantp_msg *buffer;
};
```

Visual Basic

```
<StructLayout(LayoutKind.Sequential, Pack:=8)>
Public Structure cantp_msgprogress
    <MarshalAs(UnmanagedType.U4)>
    Public state As cantp_msgprogress_state
    Public percentage As Byte
    Public buffer As IntPtr
End Structure
```

Fields

Name	Description
state	State of the message (see cantp_msgprogress_state on page 95).
percentage	Progress of the transmission/reception in percent.
buffer	Buffer to get a copy of the pending message (see cantp_msg on page 28). Use NULL/nullptr to disable the feature: copying the pending message can be time consuming and will delay the communication of the message. Depending on the configuration of ISO-TP timeouts, ISO-TP communication could be disrupted.

See also: CANTP_GetMsgProgress_2016 on page 238, **class method version:** GetMsgProgress_2016 on page 143, allocProgressBuffer_2016 on page 198, freeProgressBuffer_2016 on page 200, getProgressBuffer_2016 on page 202

3.5 Types

The PCAN-ISO-TP 2016 API defines the following types:

Name	Description
cantp_pcanstatus	Represents the PCAN error and status codes (used when cantp_status encapsulates a PCAN-Basic error).
cantp_bitrate	Represents a PCAN-FD bit rate string.
cantp_timestamp	Defines a timestamp of a CANTP message.
cantp_handle	Represents a PCAN-ISO-TP channel handle.
cantp_baudrate	Represents the baudrate register value for the PCANTP channel.
cantp_hwtype	Type of PCAN (non plug-n-play) hardware.
cantp_device	Represents a PCAN device.
cantp_statustype	Represents each group of errors a status can hold.
cantp_netstatus	Represents the network result of the communication of an ISO-TP message (used in cantp_status).
cantp_busstatus	Represents the status of a CAN bus (used in cantp_status).
cantp_errstatus	Represents a general error (used in cantp_status).
cantp_infostatus	Represents additional status information (used in cantp_status).
cantp_status	Represent the PCANTP error and status codes.
cantp_parameter	List of parameters handled by PCAN-ISO-TP (rev. 2016). PCAN-Basic parameters (PCAN_PARAM_xxx) are compatible via casting.
cantp_msgtype	Represents the type of a CANTP message.
cantp_msgflag	Represents the flags common to all types of cantp_msg.
cantp_can_msgtype	Represents the flags of a CAN or CAN FD frame.
cantp_isotp_msgtype	Represents the type of an ISO-TP message.
cantp_isotp_format	Represents the addressing format of an ISO-TP message.
cantp_isotp_addressing	Represents the type of target of an ISO-TP message.
cantp_option	Represents the options of a message.
cantp_msgprogress_state	Represents the status for a message whose transmission is in progress.
cantp_msgdirection	Represents the direction of a message's communication.

3.5.1 cantp_pcanstatus

Represents the PCAN error and status codes. It is only used when a cantp_status encapsulates a PCAN-Basic error.

Syntax

C++

```
typedef uint32_t cantp_pcanstatus;
```

Pascal OO

```
cantp_pcanstatus = UInt32;
```

C#

```
using cantp_pcanstatus = UInt32;
```

C++ / CLR

```
using cantp_pcanstatus = UInt32;
```

Visual Basic

```
Imports cantp_pcanstatus = System.UInt32
```

.NET Framework programming languages:

An alias is used to represent a PCAN-Basic error 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.

C#

```
using System;
using Peak.Can.IsoTp;
using cantp_pcanstatus = System.UInt32;
```

Visual Basic

```
Imports System
Imports Peak.Can.IsoTp
Imports cantp_pcanstatus = System.UInt32
```

3.5.2 cantp_bitrate

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

Syntax

C++

```
#define cantp_bitrate char*
```

Pascal OO

```
cantp_bitrate = PAnsiChar;
```

C#

```
using cantp_bitrate = String;
```

C++ / CLR

```
using cantp_bitrate = String^;
```

Visual Basic

```
Imports cantp_bitrate = System.String
```

Remarks

.NET Framework programming languages:

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

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

If it is wished to work with aliases, those must be copied to the working file, right after the inclusion of the Peak.Can.IsoTp Namespace. Otherwise, just use the native type, which in this case is a string.

C#

```
using System;
using Peak.Can.IsoTp;
using cantp_bitrate = System.String; // Alias' declaration for System.String
```

Visual Basic

```
Imports System
Imports Peak.Can.IsoTp
Imports cantp_bitrate = System.String  ' Alias declaration for System.String
```

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_2016 (class method version: InitializeFD_2016).

Clock frequency parameters:

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

Nominal bit rate parameters:

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

Data bit rate parameters:

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

Remarks

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

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

- The string must contain only one of the two possible clock frequency parameters, depending on the unit used (Hz, or MHz).
- The frequency to use must be one of the 6 listed within the clock frequency parameters.

- The value for each parameter must be separated with a '='. **Example**: "data_brp=1"
- Each pair of parameter/value must be separated with a ','. Blank spaces are allowed but are not necessary. Example: "f_clock_mhz=24, nom_brp=1,"
- Both bit rates, or only the nominal one, must be defined within the string (PCANTP_BR_DATA_* and PCANTP_BR_NOM_*, or only PCANTP_BR_NOM_*).

Example with nominal bit rate only:

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

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

Example with nominal and data bit rate:

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

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

Parameter value ranges:

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

See Also: CANTP_InitializeFD_2016 on page 226, **class method version:** InitializeFD_2016 on page 105

3.5.3 cantp_timestamp

Represents a Timestamp.

Syntax

C++

```
#define cantp_timestamp uint64_t
```

Pascal OO

```
cantp_timestamp = uint64;
```

C#

```
using cantp_timestamp = UInt64;
```

C++ / CLR

```
using cantp_timestamp = UInt64;
```

Visual Basic

```
Imports cantp_timestamp = System.UInt64
```

Remarks

.NET Framework programming languages:

An alias is used to represent a CAN timestamp 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.

C#

```
using System;
using Peak.Can.IsoTp;
using cantp_timestamp = System.UInt64;
```

Visual Basic

```
Imports System
Imports Peak.Can.IsoTp
Imports cantp_timestamp = System.UInt64
```

3.5.4 cantp_handle

Represents currently defined and supported PCANTP handle (a.k.a. channels).

Syntax

C++

```
#define PCAN_NONEBUS                0x00U
#define PCAN_ISABUS1                0x21U
#define PCAN_ISABUS2                0x22U
#define PCAN_ISABUS3                0x23U
#define PCAN_ISABUS4                0x24U
#define PCAN_ISABUS5                0x25U
#define PCAN_ISABUS6                0x26U
#define PCAN_ISABUS7                0x27U
#define PCAN_ISABUS8                0x28U
#define PCAN_DNGBUS1                0x31U
#define PCAN_PCIBUS1                0x41U
#define PCAN_PCIBUS2                0x42U
#define PCAN_PCIBUS3                0x43U
#define PCAN_PCIBUS4                0x44U
#define PCAN_PCIBUS5                0x45U
#define PCAN_PCIBUS6                0x46U
#define PCAN_PCIBUS7                0x47U
#define PCAN_PCIBUS8                0x48U
#define PCAN_PCIBUS9                0x409U
#define PCAN_PCIBUS10               0x40AU
```

```
#define PCAN_PCIBUS11                 0x40BU
#define PCAN_PCIBUS12                 0x40CU
#define PCAN_PCIBUS13                 0x40DU
#define PCAN_PCIBUS14                 0x40EU
#define PCAN_PCIBUS15                 0x40FU
#define PCAN_PCIBUS16                 0x410U
#define PCAN_USBBUS1                  0x51U
#define PCAN_USBBUS2                  0x52U
#define PCAN_USBBUS3                  0x53U
#define PCAN_USBBUS4                  0x54U
#define PCAN_USBBUS5                  0x55U
#define PCAN_USBBUS6                  0x56U
#define PCAN_USBBUS7                  0x57U
#define PCAN_USBBUS8                  0x58U
#define PCAN_USBBUS9                  0x509U
#define PCAN_USBBUS10                 0x50AU
#define PCAN_USBBUS11                 0x50BU
#define PCAN_USBBUS12                 0x50CU
#define PCAN_USBBUS13                 0x50DU
#define PCAN_USBBUS14                 0x50EU
#define PCAN_USBBUS15                 0x50FU
#define PCAN_USBBUS16                 0x510U
#define PCAN_PCCBUS1                  0x61U
#define PCAN_PCCBUS2                  0x62U
#define PCAN_LANBUS1                  0x801U
#define PCAN_LANBUS2                  0x802U
#define PCAN_LANBUS3                  0x803U
#define PCAN_LANBUS4                  0x804U
#define PCAN_LANBUS5                  0x805U
#define PCAN_LANBUS6                  0x806U
#define PCAN_LANBUS7                  0x807U
#define PCAN_LANBUS8                  0x808U
#define PCAN_LANBUS9                  0x809U
#define PCAN_LANBUS10                 0x80AU
#define PCAN_LANBUS11                 0x80BU
#define PCAN_LANBUS12                 0x80CU
#define PCAN_LANBUS13                 0x80DU
#define PCAN_LANBUS14                 0x80EU
#define PCAN_LANBUS15                 0x80FU
#define PCAN_LANBUS16                 0x810U


typedef enum _cantp_handle {
       PCANTP_HANDLE_NONEBUS = PCAN_NONEBUS,

       PCANTP_HANDLE_ISABUS1 = PCAN_ISABUS1,
       PCANTP_HANDLE_ISABUS2 = PCAN_ISABUS2,
       PCANTP_HANDLE_ISABUS3 = PCAN_ISABUS3,
       PCANTP_HANDLE_ISABUS4 = PCAN_ISABUS4,
       PCANTP_HANDLE_ISABUS5 = PCAN_ISABUS5,
       PCANTP_HANDLE_ISABUS6 = PCAN_ISABUS6,
       PCANTP_HANDLE_ISABUS7 = PCAN_ISABUS7,
       PCANTP_HANDLE_ISABUS8 = PCAN_ISABUS8,

       PCANTP_HANDLE_DNGBUS1 = PCAN_DNGBUS1,

       PCANTP_HANDLE_PCIBUS1 = PCAN_PCIBUS1,
       PCANTP_HANDLE_PCIBUS2 = PCAN_PCIBUS2,
       PCANTP_HANDLE_PCIBUS3 = PCAN_PCIBUS3,
       PCANTP_HANDLE_PCIBUS4 = PCAN_PCIBUS4,
       PCANTP_HANDLE_PCIBUS5 = PCAN_PCIBUS5,
```

```
        PCANTP_HANDLE_PCIBUS6 = PCAN_PCIBUS6,
        PCANTP_HANDLE_PCIBUS7 = PCAN_PCIBUS7,
        PCANTP_HANDLE_PCIBUS8 = PCAN_PCIBUS8,
        PCANTP_HANDLE_PCIBUS9 = PCAN_PCIBUS9,
        PCANTP_HANDLE_PCIBUS10 = PCAN_PCIBUS10,
        PCANTP_HANDLE_PCIBUS11 = PCAN_PCIBUS11,
        PCANTP_HANDLE_PCIBUS12 = PCAN_PCIBUS12,
        PCANTP_HANDLE_PCIBUS13 = PCAN_PCIBUS13,
        PCANTP_HANDLE_PCIBUS14 = PCAN_PCIBUS14,
        PCANTP_HANDLE_PCIBUS15 = PCAN_PCIBUS15,
        PCANTP_HANDLE_PCIBUS16 = PCAN_PCIBUS16,

        PCANTP_HANDLE_USBBUS1 = PCAN_USBBUS1,
        PCANTP_HANDLE_USBBUS2 = PCAN_USBBUS2,
        PCANTP_HANDLE_USBBUS3 = PCAN_USBBUS3,
        PCANTP_HANDLE_USBBUS4 = PCAN_USBBUS4,
        PCANTP_HANDLE_USBBUS5 = PCAN_USBBUS5,
        PCANTP_HANDLE_USBBUS6 = PCAN_USBBUS6,
        PCANTP_HANDLE_USBBUS7 = PCAN_USBBUS7,
        PCANTP_HANDLE_USBBUS8 = PCAN_USBBUS8,
        PCANTP_HANDLE_USBBUS9 = PCAN_USBBUS9,
        PCANTP_HANDLE_USBBUS10 = PCAN_USBBUS10,
        PCANTP_HANDLE_USBBUS11 = PCAN_USBBUS11,
        PCANTP_HANDLE_USBBUS12 = PCAN_USBBUS12,
        PCANTP_HANDLE_USBBUS13 = PCAN_USBBUS13,
        PCANTP_HANDLE_USBBUS14 = PCAN_USBBUS14,
        PCANTP_HANDLE_USBBUS15 = PCAN_USBBUS15,
        PCANTP_HANDLE_USBBUS16 = PCAN_USBBUS16,

        PCANTP_HANDLE_PCCBUS1 = PCAN_PCCBUS1,
        PCANTP_HANDLE_PCCBUS2 = PCAN_PCCBUS2,

        PCANTP_HANDLE_LANBUS1 = PCAN_LANBUS1,
        PCANTP_HANDLE_LANBUS2 = PCAN_LANBUS2,
        PCANTP_HANDLE_LANBUS3 = PCAN_LANBUS3,
        PCANTP_HANDLE_LANBUS4 = PCAN_LANBUS4,
        PCANTP_HANDLE_LANBUS5 = PCAN_LANBUS5,
        PCANTP_HANDLE_LANBUS6 = PCAN_LANBUS6,
        PCANTP_HANDLE_LANBUS7 = PCAN_LANBUS7,
        PCANTP_HANDLE_LANBUS8 = PCAN_LANBUS8,
        PCANTP_HANDLE_LANBUS9 = PCAN_LANBUS9,
        PCANTP_HANDLE_LANBUS10 = PCAN_LANBUS10,
        PCANTP_HANDLE_LANBUS11 = PCAN_LANBUS11,
        PCANTP_HANDLE_LANBUS12 = PCAN_LANBUS12,
        PCANTP_HANDLE_LANBUS13 = PCAN_LANBUS13,
        PCANTP_HANDLE_LANBUS14 = PCAN_LANBUS14,
        PCANTP_HANDLE_LANBUS15 = PCAN_LANBUS15,
        PCANTP_HANDLE_LANBUS16 = PCAN_LANBUS16,
} cantp_handle;
```

C++ / CLR

```
public enum cantp_handle : UInt32
{
        PCANTP_HANDLE_NONEBUS = PCANBasic::PCAN_NONEBUS,
        PCANTP_HANDLE_ISABUS1 = PCANBasic::PCAN_ISABUS1,
        PCANTP_HANDLE_ISABUS2 = PCANBasic::PCAN_ISABUS2,
        PCANTP_HANDLE_ISABUS3 = PCANBasic::PCAN_ISABUS3,
        PCANTP_HANDLE_ISABUS4 = PCANBasic::PCAN_ISABUS4,
        PCANTP_HANDLE_ISABUS5 = PCANBasic::PCAN_ISABUS5,
        PCANTP_HANDLE_ISABUS6 = PCANBasic::PCAN_ISABUS6,
        PCANTP_HANDLE_ISABUS7 = PCANBasic::PCAN_ISABUS7,
```

```
        PCANTP_HANDLE_ISABUS8 = PCANBasic::PCAN_ISABUS8,
        PCANTP_HANDLE_DNGBUS1 = PCANBasic::PCAN_DNGBUS1,
        PCANTP_HANDLE_PCIBUS1 = PCANBasic::PCAN_PCIBUS1,
        PCANTP_HANDLE_PCIBUS2 = PCANBasic::PCAN_PCIBUS2,
        PCANTP_HANDLE_PCIBUS3 = PCANBasic::PCAN_PCIBUS3,
        PCANTP_HANDLE_PCIBUS4 = PCANBasic::PCAN_PCIBUS4,
        PCANTP_HANDLE_PCIBUS5 = PCANBasic::PCAN_PCIBUS5,
        PCANTP_HANDLE_PCIBUS6 = PCANBasic::PCAN_PCIBUS6,
        PCANTP_HANDLE_PCIBUS7 = PCANBasic::PCAN_PCIBUS7,
        PCANTP_HANDLE_PCIBUS8 = PCANBasic::PCAN_PCIBUS8,
        PCANTP_HANDLE_PCIBUS9 = PCANBasic::PCAN_PCIBUS9,
        PCANTP_HANDLE_PCIBUS10 = PCANBasic::PCAN_PCIBUS10,
        PCANTP_HANDLE_PCIBUS11 = PCANBasic::PCAN_PCIBUS11,
        PCANTP_HANDLE_PCIBUS12 = PCANBasic::PCAN_PCIBUS12,
        PCANTP_HANDLE_PCIBUS13 = PCANBasic::PCAN_PCIBUS13,
        PCANTP_HANDLE_PCIBUS14 = PCANBasic::PCAN_PCIBUS14,
        PCANTP_HANDLE_PCIBUS15 = PCANBasic::PCAN_PCIBUS15,
        PCANTP_HANDLE_PCIBUS16 = PCANBasic::PCAN_PCIBUS16,
        PCANTP_HANDLE_USBBUS1 = PCANBasic::PCAN_USBBUS1,
        PCANTP_HANDLE_USBBUS2 = PCANBasic::PCAN_USBBUS2,
        PCANTP_HANDLE_USBBUS3 = PCANBasic::PCAN_USBBUS3,
        PCANTP_HANDLE_USBBUS4 = PCANBasic::PCAN_USBBUS4,
        PCANTP_HANDLE_USBBUS5 = PCANBasic::PCAN_USBBUS5,
        PCANTP_HANDLE_USBBUS6 = PCANBasic::PCAN_USBBUS6,
        PCANTP_HANDLE_USBBUS7 = PCANBasic::PCAN_USBBUS7,
        PCANTP_HANDLE_USBBUS8 = PCANBasic::PCAN_USBBUS8,
        PCANTP_HANDLE_USBBUS9 = PCANBasic::PCAN_USBBUS9,
        PCANTP_HANDLE_USBBUS10 = PCANBasic::PCAN_USBBUS10,
        PCANTP_HANDLE_USBBUS11 = PCANBasic::PCAN_USBBUS11,
        PCANTP_HANDLE_USBBUS12 = PCANBasic::PCAN_USBBUS12,
        PCANTP_HANDLE_USBBUS13 = PCANBasic::PCAN_USBBUS13,
        PCANTP_HANDLE_USBBUS14 = PCANBasic::PCAN_USBBUS14,
        PCANTP_HANDLE_USBBUS15 = PCANBasic::PCAN_USBBUS15,
        PCANTP_HANDLE_USBBUS16 = PCANBasic::PCAN_USBBUS16,
        PCANTP_HANDLE_PCCBUS1 = PCANBasic::PCAN_PCCBUS1,
        PCANTP_HANDLE_PCCBUS2 = PCANBasic::PCAN_PCCBUS2,
        PCANTP_HANDLE_LANBUS1 = PCANBasic::PCAN_LANBUS1,
        PCANTP_HANDLE_LANBUS2 = PCANBasic::PCAN_LANBUS2,
        PCANTP_HANDLE_LANBUS3 = PCANBasic::PCAN_LANBUS3,
        PCANTP_HANDLE_LANBUS4 = PCANBasic::PCAN_LANBUS4,
        PCANTP_HANDLE_LANBUS5 = PCANBasic::PCAN_LANBUS5,
        PCANTP_HANDLE_LANBUS6 = PCANBasic::PCAN_LANBUS6,
        PCANTP_HANDLE_LANBUS7 = PCANBasic::PCAN_LANBUS7,
        PCANTP_HANDLE_LANBUS8 = PCANBasic::PCAN_LANBUS8,
        PCANTP_HANDLE_LANBUS9 = PCANBasic::PCAN_LANBUS9,
        PCANTP_HANDLE_LANBUS10 = PCANBasic::PCAN_LANBUS10,
        PCANTP_HANDLE_LANBUS11 = PCANBasic::PCAN_LANBUS11,
        PCANTP_HANDLE_LANBUS12 = PCANBasic::PCAN_LANBUS12,
        PCANTP_HANDLE_LANBUS13 = PCANBasic::PCAN_LANBUS13,
        PCANTP_HANDLE_LANBUS14 = PCANBasic::PCAN_LANBUS14,
        PCANTP_HANDLE_LANBUS15 = PCANBasic::PCAN_LANBUS15,
        PCANTP_HANDLE_LANBUS16 = PCANBasic::PCAN_LANBUS16,
};
```

C#

```
public enum cantp_handle : UInt32
{
    PCANTP_HANDLE_NONEBUS = PCANBasic.PCAN_NONEBUS,
    PCANTP_HANDLE_ISABUS1 = PCANBasic.PCAN_ISABUS1,
    PCANTP_HANDLE_ISABUS2 = PCANBasic.PCAN_ISABUS2,
    PCANTP_HANDLE_ISABUS3 = PCANBasic.PCAN_ISABUS3,
```

```
    PCANTP_HANDLE_ISABUS4 = PCANBasic.PCAN_ISABUS4,
    PCANTP_HANDLE_ISABUS5 = PCANBasic.PCAN_ISABUS5,
    PCANTP_HANDLE_ISABUS6 = PCANBasic.PCAN_ISABUS6,
    PCANTP_HANDLE_ISABUS7 = PCANBasic.PCAN_ISABUS7,
    PCANTP_HANDLE_ISABUS8 = PCANBasic.PCAN_ISABUS8,
    PCANTP_HANDLE_DNGBUS1 = PCANBasic.PCAN_DNGBUS1,
    PCANTP_HANDLE_PCIBUS1 = PCANBasic.PCAN_PCIBUS1,
    PCANTP_HANDLE_PCIBUS2 = PCANBasic.PCAN_PCIBUS2,
    PCANTP_HANDLE_PCIBUS3 = PCANBasic.PCAN_PCIBUS3,
    PCANTP_HANDLE_PCIBUS4 = PCANBasic.PCAN_PCIBUS4,
    PCANTP_HANDLE_PCIBUS5 = PCANBasic.PCAN_PCIBUS5,
    PCANTP_HANDLE_PCIBUS6 = PCANBasic.PCAN_PCIBUS6,
    PCANTP_HANDLE_PCIBUS7 = PCANBasic.PCAN_PCIBUS7,
    PCANTP_HANDLE_PCIBUS8 = PCANBasic.PCAN_PCIBUS8,
    PCANTP_HANDLE_PCIBUS9 = PCANBasic.PCAN_PCIBUS9,
    PCANTP_HANDLE_PCIBUS10 = PCANBasic.PCAN_PCIBUS10,
    PCANTP_HANDLE_PCIBUS11 = PCANBasic.PCAN_PCIBUS11,
    PCANTP_HANDLE_PCIBUS12 = PCANBasic.PCAN_PCIBUS12,
    PCANTP_HANDLE_PCIBUS13 = PCANBasic.PCAN_PCIBUS13,
    PCANTP_HANDLE_PCIBUS14 = PCANBasic.PCAN_PCIBUS14,
    PCANTP_HANDLE_PCIBUS15 = PCANBasic.PCAN_PCIBUS15,
    PCANTP_HANDLE_PCIBUS16 = PCANBasic.PCAN_PCIBUS16,
    PCANTP_HANDLE_USBBUS1 = PCANBasic.PCAN_USBBUS1,
    PCANTP_HANDLE_USBBUS2 = PCANBasic.PCAN_USBBUS2,
    PCANTP_HANDLE_USBBUS3 = PCANBasic.PCAN_USBBUS3,
    PCANTP_HANDLE_USBBUS4 = PCANBasic.PCAN_USBBUS4,
    PCANTP_HANDLE_USBBUS5 = PCANBasic.PCAN_USBBUS5,
    PCANTP_HANDLE_USBBUS6 = PCANBasic.PCAN_USBBUS6,
    PCANTP_HANDLE_USBBUS7 = PCANBasic.PCAN_USBBUS7,
    PCANTP_HANDLE_USBBUS8 = PCANBasic.PCAN_USBBUS8,
    PCANTP_HANDLE_USBBUS9 = PCANBasic.PCAN_USBBUS9,
    PCANTP_HANDLE_USBBUS10 = PCANBasic.PCAN_USBBUS10,
    PCANTP_HANDLE_USBBUS11 = PCANBasic.PCAN_USBBUS11,
    PCANTP_HANDLE_USBBUS12 = PCANBasic.PCAN_USBBUS12,
    PCANTP_HANDLE_USBBUS13 = PCANBasic.PCAN_USBBUS13,
    PCANTP_HANDLE_USBBUS14 = PCANBasic.PCAN_USBBUS14,
    PCANTP_HANDLE_USBBUS15 = PCANBasic.PCAN_USBBUS15,
    PCANTP_HANDLE_USBBUS16 = PCANBasic.PCAN_USBBUS16,
    PCANTP_HANDLE_PCCBUS1 = PCANBasic.PCAN_PCCBUS1,
    PCANTP_HANDLE_PCCBUS2 = PCANBasic.PCAN_PCCBUS2,
    PCANTP_HANDLE_LANBUS1 = PCANBasic.PCAN_LANBUS1,
    PCANTP_HANDLE_LANBUS2 = PCANBasic.PCAN_LANBUS2,
    PCANTP_HANDLE_LANBUS3 = PCANBasic.PCAN_LANBUS3,
    PCANTP_HANDLE_LANBUS4 = PCANBasic.PCAN_LANBUS4,
    PCANTP_HANDLE_LANBUS5 = PCANBasic.PCAN_LANBUS5,
    PCANTP_HANDLE_LANBUS6 = PCANBasic.PCAN_LANBUS6,
    PCANTP_HANDLE_LANBUS7 = PCANBasic.PCAN_LANBUS7,
    PCANTP_HANDLE_LANBUS8 = PCANBasic.PCAN_LANBUS8,
    PCANTP_HANDLE_LANBUS9 = PCANBasic.PCAN_LANBUS9,
    PCANTP_HANDLE_LANBUS10 = PCANBasic.PCAN_LANBUS10,
    PCANTP_HANDLE_LANBUS11 = PCANBasic.PCAN_LANBUS11,
    PCANTP_HANDLE_LANBUS12 = PCANBasic.PCAN_LANBUS12,
    PCANTP_HANDLE_LANBUS13 = PCANBasic.PCAN_LANBUS13,
    PCANTP_HANDLE_LANBUS14 = PCANBasic.PCAN_LANBUS14,
    PCANTP_HANDLE_LANBUS15 = PCANBasic.PCAN_LANBUS15,
    PCANTP_HANDLE_LANBUS16 = PCANBasic.PCAN_LANBUS16,
}
```

Pascal OO

```
cantp_handle = (
    PCANTP_HANDLE_NONEBUS = PCAN_NONEBUS,
    PCANTP_HANDLE_ISABUS1 = PCAN_ISABUS1,
    PCANTP_HANDLE_ISABUS2 = PCAN_ISABUS2,
    PCANTP_HANDLE_ISABUS3 = PCAN_ISABUS3,
    PCANTP_HANDLE_ISABUS4 = PCAN_ISABUS4,
    PCANTP_HANDLE_ISABUS5 = PCAN_ISABUS5,
    PCANTP_HANDLE_ISABUS6 = PCAN_ISABUS6,
    PCANTP_HANDLE_ISABUS7 = PCAN_ISABUS7,
    PCANTP_HANDLE_ISABUS8 = PCAN_ISABUS8,
    PCANTP_HANDLE_DNGBUS1 = PCAN_DNGBUS1,
    PCANTP_HANDLE_PCIBUS1 = PCAN_PCIBUS1,
    PCANTP_HANDLE_PCIBUS2 = PCAN_PCIBUS2,
    PCANTP_HANDLE_PCIBUS3 = PCAN_PCIBUS3,
    PCANTP_HANDLE_PCIBUS4 = PCAN_PCIBUS4,
    PCANTP_HANDLE_PCIBUS5 = PCAN_PCIBUS5,
    PCANTP_HANDLE_PCIBUS6 = PCAN_PCIBUS6,
    PCANTP_HANDLE_PCIBUS7 = PCAN_PCIBUS7,
    PCANTP_HANDLE_PCIBUS8 = PCAN_PCIBUS8,
    PCANTP_HANDLE_PCIBUS9 = PCAN_PCIBUS9,
    PCANTP_HANDLE_PCIBUS10 = PCAN_PCIBUS10,
    PCANTP_HANDLE_PCIBUS11 = PCAN_PCIBUS11,
    PCANTP_HANDLE_PCIBUS12 = PCAN_PCIBUS12,
    PCANTP_HANDLE_PCIBUS13 = PCAN_PCIBUS13,
    PCANTP_HANDLE_PCIBUS14 = PCAN_PCIBUS14,
    PCANTP_HANDLE_PCIBUS15 = PCAN_PCIBUS15,
    PCANTP_HANDLE_PCIBUS16 = PCAN_PCIBUS16,
    PCANTP_HANDLE_USBBUS1 = PCAN_USBBUS1,
    PCANTP_HANDLE_USBBUS2 = PCAN_USBBUS2,
    PCANTP_HANDLE_USBBUS3 = PCAN_USBBUS3,
    PCANTP_HANDLE_USBBUS4 = PCAN_USBBUS4,
    PCANTP_HANDLE_USBBUS5 = PCAN_USBBUS5,
    PCANTP_HANDLE_USBBUS6 = PCAN_USBBUS6,
    PCANTP_HANDLE_USBBUS7 = PCAN_USBBUS7,
    PCANTP_HANDLE_USBBUS8 = PCAN_USBBUS8,
    PCANTP_HANDLE_USBBUS9 = PCAN_USBBUS9,
    PCANTP_HANDLE_USBBUS10 = PCAN_USBBUS10,
    PCANTP_HANDLE_USBBUS11 = PCAN_USBBUS11,
    PCANTP_HANDLE_USBBUS12 = PCAN_USBBUS12,
    PCANTP_HANDLE_USBBUS13 = PCAN_USBBUS13,
    PCANTP_HANDLE_USBBUS14 = PCAN_USBBUS14,
    PCANTP_HANDLE_USBBUS15 = PCAN_USBBUS15,
    PCANTP_HANDLE_USBBUS16 = PCAN_USBBUS16,
    PCANTP_HANDLE_PCCBUS1 = PCAN_PCCBUS1,
    PCANTP_HANDLE_PCCBUS2 = PCAN_PCCBUS2,
    PCANTP_HANDLE_LANBUS1 = PCAN_LANBUS1,
    PCANTP_HANDLE_LANBUS2 = PCAN_LANBUS2,
    PCANTP_HANDLE_LANBUS3 = PCAN_LANBUS3,
    PCANTP_HANDLE_LANBUS4 = PCAN_LANBUS4,
    PCANTP_HANDLE_LANBUS5 = PCAN_LANBUS5,
    PCANTP_HANDLE_LANBUS6 = PCAN_LANBUS6,
    PCANTP_HANDLE_LANBUS7 = PCAN_LANBUS7,
    PCANTP_HANDLE_LANBUS8 = PCAN_LANBUS8,
    PCANTP_HANDLE_LANBUS9 = PCAN_LANBUS9,
    PCANTP_HANDLE_LANBUS10 = PCAN_LANBUS10,
    PCANTP_HANDLE_LANBUS11 = PCAN_LANBUS11,
    PCANTP_HANDLE_LANBUS12 = PCAN_LANBUS12,
    PCANTP_HANDLE_LANBUS13 = PCAN_LANBUS13,
    PCANTP_HANDLE_LANBUS14 = PCAN_LANBUS14,
    PCANTP_HANDLE_LANBUS15 = PCAN_LANBUS15,
```

```
    PCANTP_HANDLE_LANBUS16 = PCAN_LANBUS16
);
```

Visual Basic

```
Public Enum cantp_handle As UInt32
        PCANTP_HANDLE_NONEBUS = PCANBasic.PCAN_NONEBUS
        PCANTP_HANDLE_ISABUS1 = PCANBasic.PCAN_ISABUS1
        PCANTP_HANDLE_ISABUS2 = PCANBasic.PCAN_ISABUS2
        PCANTP_HANDLE_ISABUS3 = PCANBasic.PCAN_ISABUS3
        PCANTP_HANDLE_ISABUS4 = PCANBasic.PCAN_ISABUS4
        PCANTP_HANDLE_ISABUS5 = PCANBasic.PCAN_ISABUS5
        PCANTP_HANDLE_ISABUS6 = PCANBasic.PCAN_ISABUS6
        PCANTP_HANDLE_ISABUS7 = PCANBasic.PCAN_ISABUS7
        PCANTP_HANDLE_ISABUS8 = PCANBasic.PCAN_ISABUS8
        PCANTP_HANDLE_DNGBUS1 = PCANBasic.PCAN_DNGBUS1
        PCANTP_HANDLE_PCIBUS1 = PCANBasic.PCAN_PCIBUS1
        PCANTP_HANDLE_PCIBUS2 = PCANBasic.PCAN_PCIBUS2
        PCANTP_HANDLE_PCIBUS3 = PCANBasic.PCAN_PCIBUS3
        PCANTP_HANDLE_PCIBUS4 = PCANBasic.PCAN_PCIBUS4
        PCANTP_HANDLE_PCIBUS5 = PCANBasic.PCAN_PCIBUS5
        PCANTP_HANDLE_PCIBUS6 = PCANBasic.PCAN_PCIBUS6
        PCANTP_HANDLE_PCIBUS7 = PCANBasic.PCAN_PCIBUS7
        PCANTP_HANDLE_PCIBUS8 = PCANBasic.PCAN_PCIBUS8
        PCANTP_HANDLE_PCIBUS9 = PCANBasic.PCAN_PCIBUS9
        PCANTP_HANDLE_PCIBUS10 = PCANBasic.PCAN_PCIBUS10
        PCANTP_HANDLE_PCIBUS11 = PCANBasic.PCAN_PCIBUS11
        PCANTP_HANDLE_PCIBUS12 = PCANBasic.PCAN_PCIBUS12
        PCANTP_HANDLE_PCIBUS13 = PCANBasic.PCAN_PCIBUS13
        PCANTP_HANDLE_PCIBUS14 = PCANBasic.PCAN_PCIBUS14
        PCANTP_HANDLE_PCIBUS15 = PCANBasic.PCAN_PCIBUS15
        PCANTP_HANDLE_PCIBUS16 = PCANBasic.PCAN_PCIBUS16
        PCANTP_HANDLE_USBBUS1 = PCANBasic.PCAN_USBBUS1
        PCANTP_HANDLE_USBBUS2 = PCANBasic.PCAN_USBBUS2
        PCANTP_HANDLE_USBBUS3 = PCANBasic.PCAN_USBBUS3
        PCANTP_HANDLE_USBBUS4 = PCANBasic.PCAN_USBBUS4
        PCANTP_HANDLE_USBBUS5 = PCANBasic.PCAN_USBBUS5
        PCANTP_HANDLE_USBBUS6 = PCANBasic.PCAN_USBBUS6
        PCANTP_HANDLE_USBBUS7 = PCANBasic.PCAN_USBBUS7
        PCANTP_HANDLE_USBBUS8 = PCANBasic.PCAN_USBBUS8
        PCANTP_HANDLE_USBBUS9 = PCANBasic.PCAN_USBBUS9
        PCANTP_HANDLE_USBBUS10 = PCANBasic.PCAN_USBBUS10
        PCANTP_HANDLE_USBBUS11 = PCANBasic.PCAN_USBBUS11
        PCANTP_HANDLE_USBBUS12 = PCANBasic.PCAN_USBBUS12
        PCANTP_HANDLE_USBBUS13 = PCANBasic.PCAN_USBBUS13
        PCANTP_HANDLE_USBBUS14 = PCANBasic.PCAN_USBBUS14
        PCANTP_HANDLE_USBBUS15 = PCANBasic.PCAN_USBBUS15
        PCANTP_HANDLE_USBBUS16 = PCANBasic.PCAN_USBBUS16
        PCANTP_HANDLE_PCCBUS1 = PCANBasic.PCAN_PCCBUS1
        PCANTP_HANDLE_PCCBUS2 = PCANBasic.PCAN_PCCBUS2
        PCANTP_HANDLE_LANBUS1 = PCANBasic.PCAN_LANBUS1
        PCANTP_HANDLE_LANBUS2 = PCANBasic.PCAN_LANBUS2
        PCANTP_HANDLE_LANBUS3 = PCANBasic.PCAN_LANBUS3
        PCANTP_HANDLE_LANBUS4 = PCANBasic.PCAN_LANBUS4
        PCANTP_HANDLE_LANBUS5 = PCANBasic.PCAN_LANBUS5
        PCANTP_HANDLE_LANBUS6 = PCANBasic.PCAN_LANBUS6
        PCANTP_HANDLE_LANBUS7 = PCANBasic.PCAN_LANBUS7
        PCANTP_HANDLE_LANBUS8 = PCANBasic.PCAN_LANBUS8
        PCANTP_HANDLE_LANBUS9 = PCANBasic.PCAN_LANBUS9
        PCANTP_HANDLE_LANBUS10 = PCANBasic.PCAN_LANBUS10
        PCANTP_HANDLE_LANBUS11 = PCANBasic.PCAN_LANBUS11
        PCANTP_HANDLE_LANBUS12 = PCANBasic.PCAN_LANBUS12
```

```
        PCANTP_HANDLE_LANBUS13 = PCANBasic.PCAN_LANBUS13
        PCANTP_HANDLE_LANBUS14 = PCANBasic.PCAN_LANBUS14
        PCANTP_HANDLE_LANBUS15 = PCANBasic.PCAN_LANBUS15
        PCANTP_HANDLE_LANBUS16 = PCANBasic.PCAN_LANBUS16
End Enum
```

Definitions

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

Default/Undefined handle:

	Name	Value	Description
	PCANTP_HANDLE_NONEBUS	0x0	Undefined/default value for a PCAN-ISO-TP channel

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

	Name	Value	Description
	PCANTP_HANDLE _ISABUS1	0x21	PCAN-ISA interface, channel 1
	PCANTP_HANDLE _ISABUS2	0x22	PCAN-ISA interface, channel 2
	PCANTP_HANDLE _ISABUS3	0x23	PCAN-ISA interface, channel 3
	PCANTP_HANDLE _ISABUS4	0x24	PCAN-ISA interface, channel 4
	PCANTP_HANDLE _ISABUS5	0x25	PCAN-ISA interface, channel 5
	PCANTP_HANDLE _ISABUS6	0x26	PCAN-ISA interface, channel 6
	PCANTP_HANDLE _ISABUS7	0x27	PCAN-ISA interface, channel 7
	PCANTP_HANDLE _ISABUS8	0x28	PCAN-ISA interface, channel 8

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

	Name	Value	Description
	PCANTP_HANDLE _DNGBUS1	0x31	PCAN-Dongle/LPT interface, channel 1

Handles for the PCI bus:

	Name	Value	Description
	PCANTP_HANDLE _PCIBUS1	0x41	PCAN-PCI interface, channel 1
	PCANTP_HANDLE _PCIBUS2	0x42	PCAN-PCI interface, channel 2
	PCANTP_HANDLE _PCIBUS3	0x43	PCAN-PCI interface, channel 3
	PCANTP_HANDLE _PCIBUS4	0x44	PCAN-PCI interface, channel 4
	PCANTP_HANDLE _PCIBUS5	0x45	PCAN-PCI interface, channel 5
	PCANTP_HANDLE _PCIBUS6	0x46	PCAN-PCI interface, channel 6
	PCANTP_HANDLE _PCIBUS7	0x47	PCAN-PCI interface, channel 7
	PCANTP_HANDLE _PCIBUS8	0x48	PCAN-PCI interface, channel 8
	PCANTP_HANDLE _PCIBUS9	0x409	PCAN-PCI interface, channel 9
	PCANTP_HANDLE _PCIBUS10	0x40A	PCAN-PCI interface, channel 10
	PCANTP_HANDLE _PCIBUS11	0x40B	PCAN-PCI interface, channel 11
	PCANTP_HANDLE _PCIBUS12	0x40C	PCAN-PCI interface, channel 12
	PCANTP_HANDLE _PCIBUS13	0x40D	PCAN-PCI interface, channel 13
	PCANTP_HANDLE _PCIBUS14	0x40E	PCAN-PCI interface, channel 14
	PCANTP_HANDLE _PCIBUS15	0x40F	PCAN-PCI interface, channel 15
	PCANTP_HANDLE _PCIBUS16	0x410	PCAN-PCI interface, channel 16

Handles for the USB Bus:

	Name	Value	Description
	PCANTP_HANDLE _USBBUS1	0x51	PCAN-USB interface, channel 1
	PCANTP_HANDLE _USBBUS2	0x52	PCAN-USB interface, channel 2
	PCANTP_HANDLE _USBBUS3	0x53	PCAN-USB interface, channel 3
	PCANTP_HANDLE _USBBUS4	0x54	PCAN-USB interface, channel 4
	PCANTP_HANDLE _USBBUS5	0x55	PCAN-USB interface, channel 5
	PCANTP_HANDLE _USBBUS6	0x56	PCAN-USB interface, channel 6
	PCANTP_HANDLE _USBBUS7	0x57	PCAN-USB interface, channel 7
	PCANTP_HANDLE _USBBUS8	0x58	PCAN-USB interface, channel 8
	PCANTP_HANDLE _USBBUS9	0x509	PCAN-USB interface, channel 9
	PCANTP_HANDLE _USBBUS10	0x50A	PCAN-USB interface, channel 10
	PCANTP_HANDLE _USBBUS11	0x50B	PCAN-USB interface, channel 11
	PCANTP_HANDLE _USBBUS12	0x50C	PCAN-USB interface, channel 12
	PCANTP_HANDLE _USBBUS13	0x50D	PCAN-USB interface, channel 13
	PCANTP_HANDLE _USBBUS14	0x50E	PCAN-USB interface, channel 14
	PCANTP_HANDLE _USBBUS15	0x50F	PCAN-USB interface, channel 15
	PCANTP_HANDLE _USBBUS16	0x510	PCAN-USB interface, channel 16

Handles for the PC Card Bus:

	Name	Value	Description
	PCANTP_HANDLE _PCCBUS1	0x61	PCAN-PC Card interface, channel 1
	PCANTP_HANDLE _PCCBUS2	0x62	PCAN-PC Card interface, channel 2

Handles for the LAN interface:

	Name	Value	Description
	PCANTP_HANDLE _LANBUS1	0x801	PCAN-LAN interface, channel 1
	PCANTP_HANDLE _LANBUS2	0x802	PCAN-LAN interface, channel 2
	PCANTP_HANDLE _LANBUS3	0x803	PCAN-LAN interface, channel 3
	PCANTP_HANDLE _LANBUS4	0x804	PCAN-LAN interface, channel 4
	PCANTP_HANDLE _LANBUS5	0x805	PCAN-LAN interface, channel 5
	PCANTP_HANDLE _LANBUS6	0x806	PCAN-LAN interface, channel 6
	PCANTP_HANDLE _LANBUS7	0x807	PCAN-LAN interface, channel 7
	PCANTP_HANDLE _LANBUS8	0x808	PCAN-LAN interface, channel 8
	PCANTP_HANDLE _LANBUS9	0x809	PCAN-LAN interface, channel 9
	PCANTP_HANDLE _LANBUS10	0x80A	PCAN-LAN interface, channel 10
	PCANTP_HANDLE _LANBUS11	0x80B	PCAN-LAN interface, channel 11
	PCANTP_HANDLE _LANBUS12	0x80C	PCAN-LAN interface, channel 12
	PCANTP_HANDLE _LANBUS13	0x80D	PCAN-LAN interface, channel 13
	PCANTP_HANDLE _LANBUS14	0x80E	PCAN-LAN interface, channel 14
	PCANTP_HANDLE _LANBUS15	0x80F	PCAN-LAN interface, channel 15
	PCANTP_HANDLE _LANBUS16	0x810	PCAN-LAN interface, channel 16

Remarks

Hardware Type and Channels

Non-Plug and Play: The hardware channels of this kind are used as registered. This mean, for example, it can register the PCANTP_HANDLE_ISABUS3 without having registered PCANTP_HANDLE_ISA1 and PCANTP_HANDLE_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_HANDLE_USBBUS1 to PCANTP_HANDLE_USBBUS4. The association of each channel with hardware is managed internally using the handle of hardware.

See also: Detailed parameters values on page 75.

3.5.5 cantp_baudrate

Represents a PCAN Baud rate register value for the PCANTP channel.

Syntax

C

```c
#define PCAN_BAUD_1M                    0x0014U
#define PCAN_BAUD_800K                  0x0016U
#define PCAN_BAUD_500K                  0x001CU
#define PCAN_BAUD_250K                  0x011CU
#define PCAN_BAUD_125K                  0x031CU
#define PCAN_BAUD_100K                  0x432FU
#define PCAN_BAUD_95K                   0xC34EU
#define PCAN_BAUD_83K                   0x852BU
#define PCAN_BAUD_50K                   0x472FU
#define PCAN_BAUD_47K                   0x1414U
#define PCAN_BAUD_33K                   0x8B2FU
#define PCAN_BAUD_20K                   0x532FU
#define PCAN_BAUD_10K                   0x672FU
#define PCAN_BAUD_5K                    0x7F7FU

typedef enum _cantp_baudrate {
       PCANTP_BAUDRATE_1M = PCAN_BAUD_1M,
       PCANTP_BAUDRATE_800K = PCAN_BAUD_800K,
       PCANTP_BAUDRATE_500K = PCAN_BAUD_500K,
       PCANTP_BAUDRATE_250K = PCAN_BAUD_250K,
       PCANTP_BAUDRATE_125K = PCAN_BAUD_125K,
       PCANTP_BAUDRATE_100K = PCAN_BAUD_100K,
       PCANTP_BAUDRATE_95K = PCAN_BAUD_95K,
       PCANTP_BAUDRATE_83K = PCAN_BAUD_83K,
       PCANTP_BAUDRATE_50K = PCAN_BAUD_50K,
       PCANTP_BAUDRATE_47K = PCAN_BAUD_47K,
       PCANTP_BAUDRATE_33K = PCAN_BAUD_33K,
       PCANTP_BAUDRATE_20K = PCAN_BAUD_20K,
       PCANTP_BAUDRATE_10K = PCAN_BAUD_10K,
       PCANTP_BAUDRATE_5K = PCAN_BAUD_5K,
} cantp_baudrate;
```

Pascal OO

```pascal
cantp_baudrate = (
    PCANTP_BAUDRATE_1M = UInt32(PCAN_BAUD_1M),
    PCANTP_BAUDRATE_800K = UInt32(PCAN_BAUD_800K),
    PCANTP_BAUDRATE_500K = UInt32(PCAN_BAUD_500K),
    PCANTP_BAUDRATE_250K = UInt32(PCAN_BAUD_250K),
    PCANTP_BAUDRATE_125K = UInt32(PCAN_BAUD_125K),
    PCANTP_BAUDRATE_100K = UInt32(PCAN_BAUD_100K),
    PCANTP_BAUDRATE_95K = UInt32(PCAN_BAUD_95K),
    PCANTP_BAUDRATE_83K = UInt32(PCAN_BAUD_83K),
    PCANTP_BAUDRATE_50K = UInt32(PCAN_BAUD_50K),
```

```
    PCANTP_BAUDRATE_47K = UInt32(PCAN_BAUD_47K),
    PCANTP_BAUDRATE_33K = UInt32(PCAN_BAUD_33K),
    PCANTP_BAUDRATE_20K = UInt32(PCAN_BAUD_20K),
    PCANTP_BAUDRATE_10K = UInt32(PCAN_BAUD_10K),
    PCANTP_BAUDRATE_5K = UInt32(PCAN_BAUD_5K)
);
```

C#

```
public enum cantp_baudrate : UInt32
{
    PCANTP_BAUDRATE_1M = TPCANBaudrate.PCAN_BAUD_1M,
    PCANTP_BAUDRATE_800K = TPCANBaudrate.PCAN_BAUD_800K,
    PCANTP_BAUDRATE_500K = TPCANBaudrate.PCAN_BAUD_500K,
    PCANTP_BAUDRATE_250K = TPCANBaudrate.PCAN_BAUD_250K,
    PCANTP_BAUDRATE_125K = TPCANBaudrate.PCAN_BAUD_125K,
    PCANTP_BAUDRATE_100K = TPCANBaudrate.PCAN_BAUD_100K,
    PCANTP_BAUDRATE_95K = TPCANBaudrate.PCAN_BAUD_95K,
    PCANTP_BAUDRATE_83K = TPCANBaudrate.PCAN_BAUD_83K,
    PCANTP_BAUDRATE_50K = TPCANBaudrate.PCAN_BAUD_50K,
    PCANTP_BAUDRATE_47K = TPCANBaudrate.PCAN_BAUD_47K,
    PCANTP_BAUDRATE_33K = TPCANBaudrate.PCAN_BAUD_33K,
    PCANTP_BAUDRATE_20K = TPCANBaudrate.PCAN_BAUD_20K,
    PCANTP_BAUDRATE_10K = TPCANBaudrate.PCAN_BAUD_10K,
    PCANTP_BAUDRATE_5K = TPCANBaudrate.PCAN_BAUD_5K,
}
```

C++ / CLR

```
public enum cantp_baudrate : UInt32
{
        PCANTP_BAUDRATE_1M = (UInt32)TPCANBaudrate::PCAN_BAUD_1M,
        PCANTP_BAUDRATE_800K = (UInt32)TPCANBaudrate::PCAN_BAUD_800K,
        PCANTP_BAUDRATE_500K = (UInt32)TPCANBaudrate::PCAN_BAUD_500K,
        PCANTP_BAUDRATE_250K = (UInt32)TPCANBaudrate::PCAN_BAUD_250K,
        PCANTP_BAUDRATE_125K = (UInt32)TPCANBaudrate::PCAN_BAUD_125K,
        PCANTP_BAUDRATE_100K = (UInt32)TPCANBaudrate::PCAN_BAUD_100K,
        PCANTP_BAUDRATE_95K = (UInt32)TPCANBaudrate::PCAN_BAUD_95K,
        PCANTP_BAUDRATE_83K = (UInt32)TPCANBaudrate::PCAN_BAUD_83K,
        PCANTP_BAUDRATE_50K = (UInt32)TPCANBaudrate::PCAN_BAUD_50K,
        PCANTP_BAUDRATE_47K = (UInt32)TPCANBaudrate::PCAN_BAUD_47K,
        PCANTP_BAUDRATE_33K = (UInt32)TPCANBaudrate::PCAN_BAUD_33K,
        PCANTP_BAUDRATE_20K = (UInt32)TPCANBaudrate::PCAN_BAUD_20K,
        PCANTP_BAUDRATE_10K = (UInt32)TPCANBaudrate::PCAN_BAUD_10K,
        PCANTP_BAUDRATE_5K = (UInt32)TPCANBaudrate::PCAN_BAUD_5K,
};
```

Visual Basic

```
Public Enum cantp_baudrate : UInt32
    PCANTP_BAUDRATE_1M = TPCANBaudrate.PCAN_BAUD_1M
    PCANTP_BAUDRATE_800K = TPCANBaudrate.PCAN_BAUD_800K
    PCANTP_BAUDRATE_500K = TPCANBaudrate.PCAN_BAUD_500K
    PCANTP_BAUDRATE_250K = TPCANBaudrate.PCAN_BAUD_250K
    PCANTP_BAUDRATE_125K = TPCANBaudrate.PCAN_BAUD_125K
    PCANTP_BAUDRATE_100K = TPCANBaudrate.PCAN_BAUD_100K
    PCANTP_BAUDRATE_95K = TPCANBaudrate.PCAN_BAUD_95K
    PCANTP_BAUDRATE_83K = TPCANBaudrate.PCAN_BAUD_83K
    PCANTP_BAUDRATE_50K = TPCANBaudrate.PCAN_BAUD_50K
    PCANTP_BAUDRATE_47K = TPCANBaudrate.PCAN_BAUD_47K
    PCANTP_BAUDRATE_33K = TPCANBaudrate.PCAN_BAUD_33K
    PCANTP_BAUDRATE_20K = TPCANBaudrate.PCAN_BAUD_20K
```

```
    PCANTP_BAUDRATE_10K = TPCANBaudrate.PCAN_BAUD_10K
    PCANTP_BAUDRATE_5K = TPCANBaudrate.PCAN_BAUD_5K
End Enum
```

Values

Name	Value	Description
PCANTP_BAUDRATE_1M	20	1 MBit/s
PCANTP_BAUDRATE_800K	22	800 kBit/s
PCANTP_BAUDRATE_500K	28	500 kBit/s
PCANTP_BAUDRATE_250K	284	250 kBit/s
PCANTP_BAUDRATE_125K	796	125 kBit/s
PCANTP_BAUDRATE_100K	17199	100 kBit/s
PCANTP_BAUDRATE_95K	49998	95,238 kBit/s
PCANTP_BAUDRATE_83K	34091	83,333 kBit/s
PCANTP_BAUDRATE_50K	18223	50 kBit/s
PCANTP_BAUDRATE_47K	5140	47,619 kBit/s
PCANTP_BAUDRATE_33K	35631	33,333 kBit/s
PCANTP_BAUDRATE_20K	21295	20 kBit/s
PCANTP_BAUDRATE_10K	26415	10 kBit/s
PCANTP_BAUDRATE_5K	32639	5 kBit/s

See also: CANTP_Initialize_2016 function on page 225, Initialize_2016 method on page 99

3.5.6 cantp_hwtype

Represents the type of PCAN (non-Plug and Play) hardware to be initialized. According with the programming language, this type can be a group of defined values or an enumeration.

Syntax

C++

```
#define PCAN_TYPE_ISA                0x01U
#define PCAN_TYPE_ISA_SJA            0x09U
#define PCAN_TYPE_ISA_PHYTEC         0x04U
#define PCAN_TYPE_DNG                0x02U
#define PCAN_TYPE_DNG_EPP            0x03U
#define PCAN_TYPE_DNG_SJA            0x05U
#define PCAN_TYPE_DNG_SJA_EPP        0x06U

typedef enum _cantp_hwtype {
       PCANTP_HWTYPE_ISA = PCAN_TYPE_ISA,
       PCANTP_HWTYPE_ISA_SJA = PCAN_TYPE_ISA_SJA,
       PCANTP_HWTYPE_ISA_PHYTEC = PCAN_TYPE_ISA_PHYTEC,
       PCANTP_HWTYPE_DNG = PCAN_TYPE_DNG,
       PCANTP_HWTYPE_DNG_EPP = PCAN_TYPE_DNG_EPP,
       PCANTP_HWTYPE_DNG_SJA = PCAN_TYPE_DNG_SJA,
       PCANTP_HWTYPE_DNG_SJA_EPP = PCAN_TYPE_DNG_SJA_EPP,
} cantp_hwtype;
```

Pascal OO

```
cantp_hwtype = (
    PCANTP_HWTYPE_ISA = UInt32(PCAN_TYPE_ISA),
    PCANTP_HWTYPE_ISA_SJA = UInt32(PCAN_TYPE_ISA_SJA),
    PCANTP_HWTYPE_ISA_PHYTEC = UInt32(PCAN_TYPE_ISA_PHYTEC),
    PCANTP_HWTYPE_DNG = UInt32(PCAN_TYPE_DNG),
    PCANTP_HWTYPE_DNG_EPP = UInt32(PCAN_TYPE_DNG_EPP),
```

```
    PCANTP_HWTYPE_DNG_SJA = UInt32(PCAN_TYPE_DNG_SJA),
    PCANTP_HWTYPE_DNG_SJA_EPP = UInt32(PCAN_TYPE_DNG_SJA_EPP)
);
```

C#

```
public enum cantp_hwtype : UInt32
{
    PCANTP_HWTYPE_ISA = TPCANType.PCAN_TYPE_ISA,
    PCANTP_HWTYPE_ISA_SJA = TPCANType.PCAN_TYPE_ISA_SJA,
    PCANTP_HWTYPE_ISA_PHYTEC = TPCANType.PCAN_TYPE_ISA_PHYTEC,
    PCANTP_HWTYPE_DNG = TPCANType.PCAN_TYPE_DNG,
    PCANTP_HWTYPE_DNG_EPP = TPCANType.PCAN_TYPE_DNG_EPP,
    PCANTP_HWTYPE_DNG_SJA = TPCANType.PCAN_TYPE_DNG_SJA,
    PCANTP_HWTYPE_DNG_SJA_EPP = TPCANType.PCAN_TYPE_DNG_SJA_EPP,
}
```

C++ / CLR

```
public enum cantp_hwtype : UInt32
{
       PCANTP_HWTYPE_ISA = (UInt32)TPCANType::PCAN_TYPE_ISA,
       PCANTP_HWTYPE_ISA_SJA = (UInt32)TPCANType::PCAN_TYPE_ISA_SJA,
       PCANTP_HWTYPE_ISA_PHYTEC = (UInt32)TPCANType::PCAN_TYPE_ISA_PHYTEC,
       PCANTP_HWTYPE_DNG = (UInt32)TPCANType::PCAN_TYPE_DNG,
       PCANTP_HWTYPE_DNG_EPP = (UInt32)TPCANType::PCAN_TYPE_DNG_EPP,
       PCANTP_HWTYPE_DNG_SJA = (UInt32)TPCANType::PCAN_TYPE_DNG_SJA,
       PCANTP_HWTYPE_DNG_SJA_EPP = (UInt32)TPCANType::PCAN_TYPE_DNG_SJA_EPP,
};
```

Visual Basic

```
Public Enum cantp_hwtype : UInt32
    PCANTP_HWTYPE_ISA = TPCANType.PCAN_TYPE_ISA
    PCANTP_HWTYPE_ISA_SJA = TPCANType.PCAN_TYPE_ISA_SJA
    PCANTP_HWTYPE_ISA_PHYTEC = TPCANType.PCAN_TYPE_ISA_PHYTEC
    PCANTP_HWTYPE_DNG = TPCANType.PCAN_TYPE_DNG
    PCANTP_HWTYPE_DNG_EPP = TPCANType.PCAN_TYPE_DNG_EPP
    PCANTP_HWTYPE_DNG_SJA = TPCANType.PCAN_TYPE_DNG_SJA
    PCANTP_HWTYPE_DNG_SJA_EPP = TPCANType.PCAN_TYPE_DNG_SJA_EPP
End Enum
```

Values

Name	Value	Description
PCANTP_HWTYPE_ISA	1	PCAN-ISA 82C200
PCANTP_HWTYPE_ISA_SJA	9	PCAN-ISA SJA1000
PCANTP_HWTYPE_ISA_PHYTEC	4	PHYTEC ISA
PCANTP_HWTYPE_DNG	2	PCAN-Dongle 82C200
PCANTP_HWTYPE_DNG_EPP	3	PCAN-Dongle EPP 82C200
PCANTP_HWTYPE_DNG_SJA	5	PCAN-Dongle SJA1000
PCANTP_HWTYPE_DNG_SJA_EPP	6	PCAN-Dongle EPP SJA1000

See also: CANTP_Initialize_2016 function on page 225, Initialize_2016 method on page 99

3.5.7 cantp_device

Represents PCAN devices type.

Syntax

C++

```
#define PCAN_NONE                    0x00U
#define PCAN_PEAKCAN                 0x01U
#define PCAN_ISA                     0x02U
#define PCAN_DNG                     0x03U
#define PCAN_PCI                     0x04U
#define PCAN_USB                     0x05U
#define PCAN_PCC                     0x06U
#define PCAN_VIRTUAL                 0x07U
#define PCAN_LAN                     0x08U

typedef enum _cantp_device {
       PCANTP_DEVICE_NONE = PCAN_NONE,
       PCANTP_DEVICE_PEAKCAN = PCAN_PEAKCAN,
       PCANTP_DEVICE_ISA = PCAN_ISA,
       PCANTP_DEVICE_DNG = PCAN_DNG,
       PCANTP_DEVICE_PCI = PCAN_PCI,
       PCANTP_DEVICE_USB = PCAN_USB,
       PCANTP_DEVICE_PCC = PCAN_PCC,
       PCANTP_DEVICE_VIRTUAL = PCAN_VIRTUAL,
       PCANTP_DEVICE_LAN = PCAN_LAN,
} cantp_device;
```

Pascal OO

```
cantp_device = (
    PCANTP_DEVICE_NONE = UInt32(PCAN_NONE),
    PCANTP_DEVICE_PEAKCAN = UInt32(PCAN_PEAKCAN),
    PCANTP_DEVICE_ISA = UInt32(PCAN_ISA),
    PCANTP_DEVICE_DNG = UInt32(PCAN_DNG),
    PCANTP_DEVICE_PCI = UInt32(PCAN_PCI),
    PCANTP_DEVICE_USB = UInt32(PCAN_USB),
    PCANTP_DEVICE_PCC = UInt32(PCAN_PCC),
    PCANTP_DEVICE_VIRTUAL = UInt32(PCAN_VIRTUAL),
    PCANTP_DEVICE_LAN = UInt32(PCAN_LAN)
);
```

C#

```
public enum cantp_device : UInt32
{
    PCANTP_DEVICE_NONE = TPCANDevice.PCAN_NONE,
    PCANTP_DEVICE_PEAKCAN = TPCANDevice.PCAN_PEAKCAN,
    PCANTP_DEVICE_ISA = TPCANDevice.PCAN_ISA,
    PCANTP_DEVICE_DNG = TPCANDevice.PCAN_DNG,
    PCANTP_DEVICE_PCI = TPCANDevice.PCAN_PCI,
    PCANTP_DEVICE_USB = TPCANDevice.PCAN_USB,
    PCANTP_DEVICE_PCC = TPCANDevice.PCAN_PCC,
    PCANTP_DEVICE_VIRTUAL = TPCANDevice.PCAN_VIRTUAL,
    PCANTP_DEVICE_LAN = TPCANDevice.PCAN_LAN,
}
```

C++ / CLR

```
public enum cantp_device : UInt32
{
        PCANTP_DEVICE_NONE = (UInt32)TPCANDevice::PCAN_NONE,
        PCANTP_DEVICE_PEAKCAN = (UInt32)TPCANDevice::PCAN_PEAKCAN,
        PCANTP_DEVICE_ISA = (UInt32)TPCANDevice::PCAN_ISA,
        PCANTP_DEVICE_DNG = (UInt32)TPCANDevice::PCAN_DNG,
        PCANTP_DEVICE_PCI = (UInt32)TPCANDevice::PCAN_PCI,
        PCANTP_DEVICE_USB = (UInt32)TPCANDevice::PCAN_USB,
        PCANTP_DEVICE_PCC = (UInt32)TPCANDevice::PCAN_PCC,
        PCANTP_DEVICE_VIRTUAL = (UInt32)TPCANDevice::PCAN_VIRTUAL,
        PCANTP_DEVICE_LAN = (UInt32)TPCANDevice::PCAN_LAN,
};
```

Visual Basic

```
Public Enum cantp_device As UInt32
    PCANTP_DEVICE_NONE = TPCANDevice.PCAN_NONE
    PCANTP_DEVICE_PEAKCAN = TPCANDevice.PCAN_PEAKCAN
    PCANTP_DEVICE_ISA = TPCANDevice.PCAN_ISA
    PCANTP_DEVICE_DNG = TPCANDevice.PCAN_DNG
    PCANTP_DEVICE_PCI = TPCANDevice.PCAN_PCI
    PCANTP_DEVICE_USB = TPCANDevice.PCAN_USB
    PCANTP_DEVICE_PCC = TPCANDevice.PCAN_PCC
    PCANTP_DEVICE_VIRTUAL = TPCANDevice.PCAN_VIRTUAL
    PCANTP_DEVICE_LAN = TPCANDevice.PCAN_LAN
End Enum
```

Values

Name	Value	Description
PCANTP_DEVICE_NONE	0x00	Undefined, unknown or not selected PCAN device value.
PCANTP_DEVICE_PEAKCAN	0x01	PCAN Non-Plug&Play devices. NOT USED WITHIN PCAN-Basic API.
PCANTP_DEVICE_ISA	0x02	PCAN-ISA, PCAN-PC/104, and PCAN-PC/104-Plus.
PCANTP_DEVICE_DNG	0x03	PCAN-Dongle.
PCANTP_DEVICE_PCI	0x04	PCAN-PCI, PCAN-cPCI, PCAN-miniPCI, and PCAN-PCI Express.
PCANTP_DEVICE_USB	0x05	PCAN-USB and PCAN-USB Pro.
PCANTP_DEVICE_PCC	0x06	PCAN-PC Card.
PCANTP_DEVICE_VIRTUAL	0x07	PCAN Virtual hardware. NOT USED WITHIN PCAN-Basic API.
PCANTP_DEVICE_LAN	0x08	PCAN Gateway devices.

See also: CANTP_Initialize_2016 on page 225, **class method version:** Initialize_2016 on page 99

3.5.8 cantp_statustype

Represents each group of errors a status can hold. These values can be used as flags.

Syntax

C++

```
typedef enum _cantp_statustype {
       PCANTP_STATUSTYPE_OK = 0x00,
       PCANTP_STATUSTYPE_ERR = 0x01,
       PCANTP_STATUSTYPE_BUS = 0x02,
       PCANTP_STATUSTYPE_NET = 0x04,
       PCANTP_STATUSTYPE_INFO = 0x08,
       PCANTP_STATUSTYPE_PCAN = 0x10,
} cantp_statustype;
```

Pascal OO

```
cantp_statustype = (
    PCANTP_STATUSTYPE_OK = $00,
    PCANTP_STATUSTYPE_ERR = $01,
    PCANTP_STATUSTYPE_BUS = $02,
    PCANTP_STATUSTYPE_NET = $04,
    PCANTP_STATUSTYPE_INFO = $08,
    PCANTP_STATUSTYPE_PCAN = $10
);
```

C#

```
[Flags]
public enum cantp_statustype : UInt32
{
    PCANTP_STATUSTYPE_OK = 0x00,
    PCANTP_STATUSTYPE_ERR = 0x01,
    PCANTP_STATUSTYPE_BUS = 0x02,
    PCANTP_STATUSTYPE_NET = 0x04,
    PCANTP_STATUSTYPE_INFO = 0x08,
    PCANTP_STATUSTYPE_PCAN = 0x10,
}
```

C++ / CLR

```
public enum cantp_statustype : UInt32
{
       PCANTP_STATUSTYPE_OK = 0x00,
       PCANTP_STATUSTYPE_ERR = 0x01,
       PCANTP_STATUSTYPE_BUS = 0x02,
       PCANTP_STATUSTYPE_NET = 0x04,
       PCANTP_STATUSTYPE_INFO = 0x08,
       PCANTP_STATUSTYPE_PCAN = 0x10,
};
```

Visual Basic

```
<Flags()>
Public Enum cantp_statustype As UInt32
    PCANTP_STATUSTYPE_OK = &H0
    PCANTP_STATUSTYPE_ERR = &H1
    PCANTP_STATUSTYPE_BUS = &H2
    PCANTP_STATUSTYPE_NET = &H4
    PCANTP_STATUSTYPE_INFO = &H8
    PCANTP_STATUSTYPE_PCAN = &H10
End Enum
```

Values

Name	Value	Description
PCANTP_STATUSTYPE_OK	0x00	No error.
PCANTP_STATUSTYPE_ERR	0x01	General error.
PCANTP_STATUSTYPE_BUS	0x02	Bus status.
PCANTP_STATUSTYPE_NET	0x04	Network status.
PCANTP_STATUSTYPE_INFO	0x08	Extra information.
PCANTP_STATUSTYPE_PCAN	0x10	Encapsulated PCAN-Basic status.

See also: cantp_status on page 60, CANTP_StatusListTypes_2016 on page 242, StatusListTypes_2016 on page 152

3.5.9 cantp_netstatus

Represents the network result of the communication of an CANTP message (used in cantp_status).

Syntax

C++

```
typedef enum _cantp_netstatus {
       PCANTP_NETSTATUS_OK = 0x00,
       PCANTP_NETSTATUS_TIMEOUT_A = 0x01,
       PCANTP_NETSTATUS_TIMEOUT_Bs = 0x02,
       PCANTP_NETSTATUS_TIMEOUT_Cr = 0x03,
       PCANTP_NETSTATUS_WRONG_SN = 0x04,
       PCANTP_NETSTATUS_INVALID_FS = 0x05,
       PCANTP_NETSTATUS_UNEXP_PDU = 0x06,
       PCANTP_NETSTATUS_WFT_OVRN = 0x07,
       PCANTP_NETSTATUS_BUFFER_OVFLW = 0x08,
       PCANTP_NETSTATUS_ERROR = 0x09,
       PCANTP_NETSTATUS_IGNORED = 0x0A,
       PCANTP_NETSTATUS_TIMEOUT_As = 0x0B,
       PCANTP_NETSTATUS_TIMEOUT_Ar = 0x0C,
       PCANTP_NETSTATUS_XMT_FULL = 0x0D,
       PCANTP_NETSTATUS_BUS_ERROR = 0x0E,
       PCANTP_NETSTATUS_NO_MEMORY = 0x0F,
} cantp_netstatus;
```

Pascal OO

```
cantp_netstatus = (
    PCANTP_NETSTATUS_OK = $00,
    PCANTP_NETSTATUS_TIMEOUT_A = $01,
    PCANTP_NETSTATUS_TIMEOUT_Bs = $02,
    PCANTP_NETSTATUS_TIMEOUT_Cr = $03,
    PCANTP_NETSTATUS_WRONG_SN = $04,
    PCANTP_NETSTATUS_INVALID_FS = $05,
    PCANTP_NETSTATUS_UNEXP_PDU = $06,
    PCANTP_NETSTATUS_WFT_OVRN = $07,
    PCANTP_NETSTATUS_BUFFER_OVFLW = $08,
    PCANTP_NETSTATUS_ERROR = $09,
    PCANTP_NETSTATUS_IGNORED = $0A,
    PCANTP_NETSTATUS_TIMEOUT_As = $0B,
    PCANTP_NETSTATUS_TIMEOUT_Ar = $0C,
    PCANTP_NETSTATUS_XMT_FULL = $0D,
    PCANTP_NETSTATUS_BUS_ERROR = $0E,
    PCANTP_NETSTATUS_NO_MEMORY = $0F
);
```

C#

```
public enum cantp_netstatus : UInt32
{
    PCANTP_NETSTATUS_OK = 0x00,
    PCANTP_NETSTATUS_TIMEOUT_A = 0x01,
    PCANTP_NETSTATUS_TIMEOUT_Bs = 0x02,
    PCANTP_NETSTATUS_TIMEOUT_Cr = 0x03,
    PCANTP_NETSTATUS_WRONG_SN = 0x04,
    PCANTP_NETSTATUS_INVALID_FS = 0x05,
    PCANTP_NETSTATUS_UNEXP_PDU = 0x06,
    PCANTP_NETSTATUS_WFT_OVRN = 0x07,
    PCANTP_NETSTATUS_BUFFER_OVFLW = 0x08,
    PCANTP_NETSTATUS_ERROR = 0x09,
    PCANTP_NETSTATUS_IGNORED = 0x0A,
    PCANTP_NETSTATUS_TIMEOUT_As = 0x0B,
    PCANTP_NETSTATUS_TIMEOUT_Ar = 0x0C,
    PCANTP_NETSTATUS_XMT_FULL = 0x0D,
    PCANTP_NETSTATUS_BUS_ERROR = 0x0E,
    PCANTP_NETSTATUS_NO_MEMORY = 0x0F,
}
```

C++ / CLR

```
public enum cantp_netstatus : UInt32
{
       PCANTP_NETSTATUS_OK = 0x00,
       PCANTP_NETSTATUS_TIMEOUT_A = 0x01,
       PCANTP_NETSTATUS_TIMEOUT_Bs = 0x02,
       PCANTP_NETSTATUS_TIMEOUT_Cr = 0x03,
       PCANTP_NETSTATUS_WRONG_SN = 0x04,
       PCANTP_NETSTATUS_INVALID_FS = 0x05,
       PCANTP_NETSTATUS_UNEXP_PDU = 0x06,
       PCANTP_NETSTATUS_WFT_OVRN = 0x07,
       PCANTP_NETSTATUS_BUFFER_OVFLW = 0x08,
       PCANTP_NETSTATUS_ERROR = 0x09,
       PCANTP_NETSTATUS_IGNORED = 0x0A,
       PCANTP_NETSTATUS_TIMEOUT_As = 0x0B,
       PCANTP_NETSTATUS_TIMEOUT_Ar = 0x0C,
       PCANTP_NETSTATUS_XMT_FULL = 0x0D,
       PCANTP_NETSTATUS_BUS_ERROR = 0x0E,
       PCANTP_NETSTATUS_NO_MEMORY = 0x0F,
};
```

Visual Basic

```
Public Enum cantp_netstatus As UInt32
    PCANTP_NETSTATUS_OK = &H0
    PCANTP_NETSTATUS_TIMEOUT_A = &H1
    PCANTP_NETSTATUS_TIMEOUT_Bs = &H2
    PCANTP_NETSTATUS_TIMEOUT_Cr = &H3
    PCANTP_NETSTATUS_WRONG_SN = &H4
    PCANTP_NETSTATUS_INVALID_FS = &H5
    PCANTP_NETSTATUS_UNEXP_PDU = &H6
    PCANTP_NETSTATUS_WFT_OVRN = &H7
    PCANTP_NETSTATUS_BUFFER_OVFLW = &H8
    PCANTP_NETSTATUS_ERROR = &H9
    PCANTP_NETSTATUS_IGNORED = &HA
    PCANTP_NETSTATUS_TIMEOUT_As = &HB
    PCANTP_NETSTATUS_TIMEOUT_Ar = &HC
    PCANTP_NETSTATUS_XMT_FULL = &HD
    PCANTP_NETSTATUS_BUS_ERROR = &HE
```

```
    PCANTP_NETSTATUS_NO_MEMORY = &HF
End Enum
```

Values

Name	Value	Description
PCANTP_NETSTATUS_OK	0x00	No network error.
PCANTP_NETSTATUS_TIMEOUT_A	0x01	Timeout occured between 2 frames transmission (sender and receiver side).
PCANTP_NETSTATUS_TIMEOUT_Bs	0x02	Sender side timeout while waiting for flow control frame.
PCANTP_NETSTATUS_TIMEOUT_Cr	0x03	Receiver side timeout while waiting for consecutive frame.
PCANTP_NETSTATUS_WRONG_SN	0x04	Unexpected sequence number.
PCANTP_NETSTATUS_INVALID_FS	0x05	Invalid or unknown FlowStatus.
PCANTP_NETSTATUS_UNEXP_PDU	0x06	Unexpected protocol data unit.
PCANTP_NETSTATUS_WFT_OVRN	0x07	Reception of flow control WAIT frame that exceeds the maximum counter defined by PCANTP_PARAMETER_WFT_MAX.
PCANTP_NETSTATUS_BUFFER_OVFLW	0x08	Buffer on the receiver side cannot store the data length (server side only).
PCANTP_NETSTATUS_ERROR	0x09	General error.
PCANTP_NETSTATUS_IGNORED	0x0A	Message was invalid and ignored.
PCANTP_NETSTATUS_TIMEOUT_As	0x0B	Sender side timeout while transmitting.
PCANTP_NETSTATUS_TIMEOUT_Ar	0x0C	Receiver side timeout while transmitting.
PCANTP_NETSTATUS_XMT_FULL	0x0D	Transmit queue is full (failed too many times; NON ISO-TP related network results).
PCANTP_NETSTATUS_BUS_ERROR	0x0E	CAN bus error (NON ISO-TP related network results).
PCANTP_NETSTATUS_NO_MEMORY	0x0F	Memory allocation error (NON ISO-TP related network results).

See also: cantp_status on page 60, CANTP_StatusGet_2016 on page 236, StatusGet_2016 on page 150

3.5.10 cantp_busstatus

Represents the status of a CAN bus (used in cantp_status).

Syntax

C++

```
typedef enum _cantp_busstatus {
       PCANTP_BUSSTATUS_OK = 0x00,
       PCANTP_BUSSTATUS_LIGHT = 0x01,
       PCANTP_BUSSTATUS_HEAVY = 0x02,
       PCANTP_BUSSTATUS_WARNING = PCANTP_BUSSTATUS_HEAVY,
       PCANTP_BUSSTATUS_PASSIVE = 0x04,
       PCANTP_BUSSTATUS_OFF = 0x08,
       PCANTP_BUSSTATUS_ANY = PCANTP_BUSSTATUS_LIGHT | PCANTP_BUSSTATUS_HEAVY |
PCANTP_BUSSTATUS_WARNING | PCANTP_BUSSTATUS_PASSIVE | PCANTP_BUSSTATUS_OFF,
} cantp_busstatus;
```

Pascal OO

```
cantp_busstatus = (
    PCANTP_BUSSTATUS_OK = $00,
    PCANTP_BUSSTATUS_LIGHT = $01,
    PCANTP_BUSSTATUS_HEAVY = $02,
    PCANTP_BUSSTATUS_WARNING = PCANTP_BUSSTATUS_HEAVY,
    PCANTP_BUSSTATUS_PASSIVE = $04,
    PCANTP_BUSSTATUS_OFF = $08,
    PCANTP_BUSSTATUS_ANY = UInt32(PCANTP_BUSSTATUS_LIGHT) Or UInt32(PCANTP_BUSSTATUS_HEAVY) Or
UInt32(PCANTP_BUSSTATUS_WARNING) Or UInt32(PCANTP_BUSSTATUS_PASSIVE) Or
UInt32(PCANTP_BUSSTATUS_OFF)
);
```

C#

```
[Flags]
public enum cantp_busstatus : UInt32
{
    PCANTP_BUSSTATUS_OK = 0x00,
    PCANTP_BUSSTATUS_LIGHT = 0x01,
    PCANTP_BUSSTATUS_HEAVY = 0x02,
    PCANTP_BUSSTATUS_WARNING = PCANTP_BUSSTATUS_HEAVY,
    PCANTP_BUSSTATUS_PASSIVE = 0x04,
    PCANTP_BUSSTATUS_OFF = 0x08,
    PCANTP_BUSSTATUS_ANY = PCANTP_BUSSTATUS_LIGHT | PCANTP_BUSSTATUS_HEAVY |
PCANTP_BUSSTATUS_WARNING | PCANTP_BUSSTATUS_PASSIVE | PCANTP_BUSSTATUS_OFF,
}
```

C++ / CLR

```
public enum cantp_busstatus : UInt32
{
       PCANTP_BUSSTATUS_OK = 0x00,
       PCANTP_BUSSTATUS_LIGHT = 0x01,
       PCANTP_BUSSTATUS_HEAVY = 0x02,
       PCANTP_BUSSTATUS_WARNING = PCANTP_BUSSTATUS_HEAVY,
       PCANTP_BUSSTATUS_PASSIVE = 0x04,
       PCANTP_BUSSTATUS_OFF = 0x08,
       PCANTP_BUSSTATUS_ANY = PCANTP_BUSSTATUS_LIGHT | PCANTP_BUSSTATUS_HEAVY |
PCANTP_BUSSTATUS_WARNING | PCANTP_BUSSTATUS_PASSIVE | PCANTP_BUSSTATUS_OFF,
};
```

Visual Basic

```
<Flags()>
Public Enum cantp_busstatus As UInt32
    PCANTP_BUSSTATUS_OK = &H0
    PCANTP_BUSSTATUS_LIGHT = &H1
    PCANTP_BUSSTATUS_HEAVY = &H2
    PCANTP_BUSSTATUS_WARNING = PCANTP_BUSSTATUS_HEAVY
    PCANTP_BUSSTATUS_PASSIVE = &H4
    PCANTP_BUSSTATUS_OFF = &H8
    PCANTP_BUSSTATUS_ANY = PCANTP_BUSSTATUS_LIGHT Or PCANTP_BUSSTATUS_HEAVY Or
PCANTP_BUSSTATUS_WARNING Or PCANTP_BUSSTATUS_PASSIVE Or PCANTP_BUSSTATUS_OFF
End Enum
```

Values

Name	Value	Description
PCANTP_BUSSTATUS_OK	0x0	Bus is in active state.
PCANTP_BUSSTATUS_LIGHT	0x1	Bus error: an error counter reached the 'light' limit.
PCANTP_BUSSTATUS_HEAVY	0x2	Bus error: an error counter reached the 'heavy' limit.
PCANTP_BUSSTATUS_WARNING	0x2	Bus error: an error counter reached the 'warning/heavy' limit.
PCANTP_BUSSTATUS_PASSIVE	0x4	Bus error: the CAN controller is error passive.
PCANTP_BUSSTATUS_OFF	0x8	Bus error: the CAN controller is in bus-off state.
PCANTP_BUSSTATUS_ANY	0xF	Mask for all bus errors.

See also: cantp_status on page 60, CANTP_StatusGet_2016 on page 236, StatusGet_2016 on page 150

3.5.11 cantp_errstatus

Represents a general error (used in cantp_status).

Syntax

C++

```
typedef enum _cantp_errstatus {
       PCANTP_ERRSTATUS_OK = 0x00U,
       PCANTP_ERRSTATUS_NOT_INITIALIZED = 0x01U,
       PCANTP_ERRSTATUS_ALREADY_INITIALIZED = 0x02U,
       PCANTP_ERRSTATUS_NO_MEMORY = 0x03U,
       PCANTP_ERRSTATUS_OVERFLOW = 0x04U,
       PCANTP_ERRSTATUS_NO_MESSAGE = 0x07U,
       PCANTP_ERRSTATUS_PARAM_INVALID_TYPE = 0x08U,
       PCANTP_ERRSTATUS_PARAM_INVALID_VALUE = 0x09U,
       PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED = 0x0DU,
       PCANTP_ERRSTATUS_MAPPING_INVALID = 0x0EU,
       PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED = 0x0FU,
       PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL = 0x10U,
       PCANTP_ERRSTATUS_QUEUE_TX_FULL = 0x11U,
       PCANTP_ERRSTATUS_LOCK_TIMEOUT = 0x12U,
       PCANTP_ERRSTATUS_INVALID_HANDLE = 0x13U,
       PCANTP_ERRSTATUS_UNKNOWN = 0xFFU,
} cantp_errstatus;
```

Pascal OO

```
cantp_errstatus = (
    PCANTP_ERRSTATUS_OK = $00,
    PCANTP_ERRSTATUS_NOT_INITIALIZED = $01,
    PCANTP_ERRSTATUS_ALREADY_INITIALIZED = $02,
    PCANTP_ERRSTATUS_NO_MEMORY = $03,
    PCANTP_ERRSTATUS_OVERFLOW = $04,
    PCANTP_ERRSTATUS_NO_MESSAGE = $07,
    PCANTP_ERRSTATUS_PARAM_INVALID_TYPE = $08,
    PCANTP_ERRSTATUS_PARAM_INVALID_VALUE = $09,
    PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED = $0D,
    PCANTP_ERRSTATUS_MAPPING_INVALID = $0E,
    PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED = $0F,
    PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL = $10,
    PCANTP_ERRSTATUS_QUEUE_TX_FULL = $11,
    PCANTP_ERRSTATUS_LOCK_TIMEOUT = $12,
    PCANTP_ERRSTATUS_INVALID_HANDLE = $13,
    PCANTP_ERRSTATUS_UNKNOWN = $FF
);
```

C#

```
public enum cantp_errstatus : UInt32
{
        PCANTP_ERRSTATUS_OK = 0x00,
        PCANTP_ERRSTATUS_NOT_INITIALIZED = 0x01,
        PCANTP_ERRSTATUS_ALREADY_INITIALIZED = 0x02,
        PCANTP_ERRSTATUS_NO_MEMORY = 0x03,
        PCANTP_ERRSTATUS_OVERFLOW = 0x04,
        PCANTP_ERRSTATUS_NO_MESSAGE = 0x07,
        PCANTP_ERRSTATUS_PARAM_INVALID_TYPE = 0x08,
        PCANTP_ERRSTATUS_PARAM_INVALID_VALUE = 0x09,
        PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED = 0x0D,
        PCANTP_ERRSTATUS_MAPPING_INVALID = 0x0E,
```

```
        PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED = 0x0F,
        PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL = 0x10,
        PCANTP_ERRSTATUS_QUEUE_TX_FULL = 0x11,
        PCANTP_ERRSTATUS_LOCK_TIMEOUT = 0x12,
        PCANTP_ERRSTATUS_INVALID_HANDLE = 0x13,
        PCANTP_ERRSTATUS_UNKNOWN = 0xFF,
}
```

C++ / CLR

```
public enum cantp_errstatus : UInt32
{
       PCANTP_ERRSTATUS_OK = 0x00,
       PCANTP_ERRSTATUS_NOT_INITIALIZED = 0x01,
       PCANTP_ERRSTATUS_ALREADY_INITIALIZED = 0x02,
       PCANTP_ERRSTATUS_NO_MEMORY = 0x03,
       PCANTP_ERRSTATUS_OVERFLOW = 0x04,
       PCANTP_ERRSTATUS_NO_MESSAGE = 0x07,
       PCANTP_ERRSTATUS_PARAM_INVALID_TYPE = 0x08,
       PCANTP_ERRSTATUS_PARAM_INVALID_VALUE = 0x09,
       PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED = 0x0D,
       PCANTP_ERRSTATUS_MAPPING_INVALID = 0x0E,
       PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED = 0x0F,
       PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL = 0x10,
       PCANTP_ERRSTATUS_QUEUE_TX_FULL = 0x11,
       PCANTP_ERRSTATUS_LOCK_TIMEOUT = 0x12,
       PCANTP_ERRSTATUS_INVALID_HANDLE = 0x13,
       PCANTP_ERRSTATUS_UNKNOWN = 0xFF,
};
```

Visual Basic

```
Public Enum cantp_errstatus As UInt32
    PCANTP_ERRSTATUS_OK = &H0
    PCANTP_ERRSTATUS_NOT_INITIALIZED = &H1
    PCANTP_ERRSTATUS_ALREADY_INITIALIZED = &H2
    PCANTP_ERRSTATUS_NO_MEMORY = &H3
    PCANTP_ERRSTATUS_OVERFLOW = &H4
    PCANTP_ERRSTATUS_NO_MESSAGE = &H7
    PCANTP_ERRSTATUS_PARAM_INVALID_TYPE = &H8
    PCANTP_ERRSTATUS_PARAM_INVALID_VALUE = &H9
    PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED = &HD
    PCANTP_ERRSTATUS_MAPPING_INVALID = &HE
    PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED = &HF
    PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL = &H10
    PCANTP_ERRSTATUS_QUEUE_TX_FULL = &H11
    PCANTP_ERRSTATUS_LOCK_TIMEOUT = &H12
    PCANTP_ERRSTATUS_INVALID_HANDLE = &H13
    PCANTP_ERRSTATUS_UNKNOWN = &HFF
End Enum
```

Values

Name	Value	Description
PCANTP_ERRSTATUS_OK	0x00	No error.
PCANTP_ERRSTATUS_NOT_INITIALIZED	0x01	Not Initialized.
PCANTP_ERRSTATUS_ALREADY_INITIALIZED	0x02	Already Initialized.
PCANTP_ERRSTATUS_NO_MEMORY	0x03	Could not obtain memory.
PCANTP_ERRSTATUS_OVERFLOW	0x04	Input buffer overflow.
PCANTP_ERRSTATUS_NO_MESSAGE	0x07	No message available.

Name	Value	Description
PCANTP_ERRSTATUS_PARAM_INVALID_TYPE	0x08	Wrong message parameters.
PCANTP_ERRSTATUS_PARAM_INVALID_VALUE	0x09	Wrong message parameters.
PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED	0x0D	PCANTP mapping not initialized.
PCANTP_ERRSTATUS_MAPPING_INVALID	0x0E	PCANTP mapping parameters are invalid.
PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED	0x0F	PCANTP mapping already defined.
PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL	0x10	Buffer is too small.
PCANTP_ERRSTATUS_QUEUE_TX_FULL	0x11	Transmit queue is full.
PCANTP_ERRSTATUS_LOCK_TIMEOUT	0x12	Failed to get an access to the internal lock.
PCANTP_ERRSTATUS_INVALID_HANDLE	0x13	Invalid cantp_handle.
PCANTP_ERRSTATUS_UNKNOWN	0xff	Unknown/generic error.

See also: cantp_status on page 60, CANTP_StatusGet_2016 on page 236, StatusGet_2016 on page 150

3.5.12 cantp_infostatus

Represents additional status information (used in cantp_status).

Syntax

C++

```
typedef enum _cantp_infostatus {
       PCANTP_INFOSTATUS_OK = 0x00,
       PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED = 0x01,
       PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED = 0x02,
       PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED = 0x04,
       PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED = 0x08,
       PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL = 0x10,
       PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE = 0x20,
} cantp_infostatus;
```

Pascal OO

```
cantp_infostatus = (
    PCANTP_INFOSTATUS_OK = $00,
    PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED = $01,
    PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED = $02,
    PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED = $04,
    PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED = $08,
    PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL = $10,
    PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE = $20
);
```

C#

```
[Flags]
public enum cantp_infostatus : UInt32
{
    PCANTP_INFOSTATUS_OK = 0x00,
    PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED = 0x01,
    PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED = 0x02,
    PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED = 0x04,
    PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED = 0x08,
    PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL = 0x10,
    PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE = 0x20,
}
```

C++ / CLR

```
public enum cantp_infostatus : UInt32
{
        PCANTP_INFOSTATUS_OK = 0x00,
        PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED = 0x01,
        PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED = 0x02,
        PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED = 0x04,
        PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED = 0x08,
        PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL = 0x10,
        PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE = 0x20,
};
```

Visual Basic

```
<Flags()>
Public Enum cantp_infostatus As UInt32
    PCANTP_INFOSTATUS_OK = &H0
    PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED = &H1
    PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED = &H2
    PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED = &H4
    PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED = &H8
    PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL = &H10
    PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE = &H20
End Enum
```

Values

Name	Value	Description
PCANTP_INFOSTATUS_OK	0x00	No extra information.
PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED	0x01	Input was modified by the API.
PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED	0x02	DLC value was modified by the API.
PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED	0x04	Data Length value was modified by the API.
PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED	0x08	FD related flags value were modified by the API.
PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL	0x10	Messages receive queue is full (oldest messages may be lost).
PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE	0x20	Buffer is used by another thread or API.

Note: cantp_infostatus is used as flags. Actual values can be a composition of the listed values

See also: cantp_status below, CANTP_StatusGet_2016 on page 236, StatusGet_2016 on page 150

3.5.13 cantp_status

Represent the PCANTP error and status codes.

Syntax

C++

```
typedef enum _cantp_status {
    PCANTP_STATUS_OK = PCANTP_ERRSTATUS_OK,
    PCANTP_STATUS_NOT_INITIALIZED = PCANTP_ERRSTATUS_NOT_INITIALIZED,
    PCANTP_STATUS_ALREADY_INITIALIZED = PCANTP_ERRSTATUS_ALREADY_INITIALIZED,
    PCANTP_STATUS_NO_MEMORY = PCANTP_ERRSTATUS_NO_MEMORY,
    PCANTP_STATUS_OVERFLOW = PCANTP_ERRSTATUS_OVERFLOW,
    PCANTP_STATUS_NO_MESSAGE = PCANTP_ERRSTATUS_NO_MESSAGE,
    PCANTP_STATUS_PARAM_INVALID_TYPE = PCANTP_ERRSTATUS_PARAM_INVALID_TYPE,
    PCANTP_STATUS_PARAM_INVALID_VALUE = PCANTP_ERRSTATUS_PARAM_INVALID_VALUE,
```

```
    PCANTP_STATUS_MAPPING_NOT_INITIALIZED = PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED,
    PCANTP_STATUS_MAPPING_INVALID = PCANTP_ERRSTATUS_MAPPING_INVALID,
    PCANTP_STATUS_MAPPING_ALREADY_INITIALIZED = PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED,
    PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL = PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL,
    PCANTP_STATUS_QUEUE_TX_FULL = PCANTP_ERRSTATUS_QUEUE_TX_FULL,
    PCANTP_STATUS_LOCK_TIMEOUT = PCANTP_ERRSTATUS_LOCK_TIMEOUT,
    PCANTP_STATUS_HANDLE_INVALID = PCANTP_ERRSTATUS_INVALID_HANDLE,
    PCANTP_STATUS_UNKNOWN = PCANTP_ERRSTATUS_UNKNOWN,
    PCANTP_STATUS_FLAG_BUS_LIGHT = (PCANTP_BUSSTATUS_LIGHT << PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_HEAVY = (PCANTP_BUSSTATUS_HEAVY << PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_WARNING = PCANTP_STATUS_FLAG_BUS_HEAVY,
    PCANTP_STATUS_FLAG_BUS_PASSIVE = (PCANTP_BUSSTATUS_PASSIVE << PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_OFF = (PCANTP_BUSSTATUS_OFF << PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_ANY = (PCANTP_BUSSTATUS_ANY << PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_NETWORK_RESULT = (1 << PCANTP_STATUS_OFFSET_NET),
    PCANTP_STATUS_NETWORK_TIMEOUT_A = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_TIMEOUT_A << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_TIMEOUT_Bs = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_TIMEOUT_Bs << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_TIMEOUT_Cr = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_TIMEOUT_Cr << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_WRONG_SN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_WRONG_SN << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_INVALID_FS = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_INVALID_FS << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_UNEXP_PDU = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_UNEXP_PDU << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_WFT_OVRN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_WFT_OVRN << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_BUFFER_OVFLW = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_BUFFER_OVFLW << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_ERROR = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_ERROR << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_IGNORED = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_IGNORED << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_TIMEOUT_Ar = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_TIMEOUT_Ar << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_TIMEOUT_As = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (PCANTP_NETSTATUS_TIMEOUT_As << (PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_CAUTION_INPUT_MODIFIED = (PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED
          << PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_DLC_MODIFIED = (PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED
          << PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_DATA_LENGTH_MODIFIED =
          (PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED << PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_FD_FLAG_MODIFIED = (PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED
          << PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_RX_QUEUE_FULL = (PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL
          << PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_BUFFER_IN_USE = (PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE
          << PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_FLAG_PCAN_STATUS = 0x80000000U,
    PCANTP_STATUS_MASK_ERROR = 0x000000FFU,
    PCANTP_STATUS_MASK_BUS = 0x00001F00U,
    PCANTP_STATUS_MASK_ISOTP_NET = 0x0003E000U,
    PCANTP_STATUS_MASK_INFO = 0x00FC0000U,
    PCANTP_STATUS_MASK_PCAN = ~PCANTP_STATUS_FLAG_PCAN_STATUS,
} cantp_status;
```

C++ / CLR

```
public enum cantp_status : UInt32
```

```
{
  PCANTP_STATUS_OK = cantp_errstatus::PCANTP_ERRSTATUS_OK,
  PCANTP_STATUS_NOT_INITIALIZED = cantp_errstatus::PCANTP_ERRSTATUS_NOT_INITIALIZED,
  PCANTP_STATUS_ALREADY_INITIALIZED = cantp_errstatus::PCANTP_ERRSTATUS_ALREADY_INITIALIZED,
  PCANTP_STATUS_NO_MEMORY = cantp_errstatus::PCANTP_ERRSTATUS_NO_MEMORY,
  PCANTP_STATUS_OVERFLOW = cantp_errstatus::PCANTP_ERRSTATUS_OVERFLOW,
  PCANTP_STATUS_NO_MESSAGE = cantp_errstatus::PCANTP_ERRSTATUS_NO_MESSAGE,
  PCANTP_STATUS_PARAM_INVALID_TYPE = cantp_errstatus::PCANTP_ERRSTATUS_PARAM_INVALID_TYPE,
  PCANTP_STATUS_PARAM_INVALID_VALUE = cantp_errstatus::PCANTP_ERRSTATUS_PARAM_INVALID_VALUE,
  PCANTP_STATUS_MAPPING_NOT_INITIALIZED =
          cantp_errstatus::PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED,
  PCANTP_STATUS_MAPPING_INVALID = cantp_errstatus::PCANTP_ERRSTATUS_MAPPING_INVALID,
  PCANTP_STATUS_MAPPING_ALREADY_INITIALIZED =
          cantp_errstatus::PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED,
  PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL =
          cantp_errstatus::PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL,
  PCANTP_STATUS_QUEUE_TX_FULL = cantp_errstatus::PCANTP_ERRSTATUS_QUEUE_TX_FULL,
  PCANTP_STATUS_LOCK_TIMEOUT = cantp_errstatus::PCANTP_ERRSTATUS_LOCK_TIMEOUT,
  PCANTP_STATUS_HANDLE_INVALID = cantp_errstatus::PCANTP_ERRSTATUS_INVALID_HANDLE,
  PCANTP_STATUS_UNKNOWN = cantp_errstatus::PCANTP_ERRSTATUS_UNKNOWN,

#pragma region Bus status flags(bits[8..11])
    PCANTP_STATUS_FLAG_BUS_LIGHT = (cantp_busstatus::PCANTP_BUSSTATUS_LIGHT
          << cantp_status_offset::PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_HEAVY = (cantp_busstatus::PCANTP_BUSSTATUS_HEAVY
          << cantp_status_offset::PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_WARNING = PCANTP_STATUS_FLAG_BUS_HEAVY,
    PCANTP_STATUS_FLAG_BUS_PASSIVE = (cantp_busstatus::PCANTP_BUSSTATUS_PASSIVE
          << cantp_status_offset::PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_OFF = (cantp_busstatus::PCANTP_BUSSTATUS_OFF
          << cantp_status_offset::PCANTP_STATUS_OFFSET_BUS),
    PCANTP_STATUS_FLAG_BUS_ANY = (cantp_busstatus::PCANTP_BUSSTATUS_ANY
          << cantp_status_offset::PCANTP_STATUS_OFFSET_BUS),
#pragma endregion

#pragma region Network status(bits[13..17])
    PCANTP_STATUS_FLAG_NETWORK_RESULT = (1 << cantp_status_offset::PCANTP_STATUS_OFFSET_NET),
    PCANTP_STATUS_NETWORK_TIMEOUT_A = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_TIMEOUT_A
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_TIMEOUT_Bs = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_TIMEOUT_Bs
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_TIMEOUT_Cr = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_TIMEOUT_Cr
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_WRONG_SN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_WRONG_SN
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_INVALID_FS = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_INVALID_FS
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_UNEXP_PDU = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_UNEXP_PDU
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_WFT_OVRN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_WFT_OVRN
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_BUFFER_OVFLW = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_BUFFER_OVFLW
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_ERROR = (PCANTP_STATUS_FLAG_NETWORK_RESULT
```

```
          | (cantp_netstatus::PCANTP_NETSTATUS_ERROR
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
    PCANTP_STATUS_NETWORK_IGNORED = (PCANTP_STATUS_FLAG_NETWORK_RESULT
          | (cantp_netstatus::PCANTP_NETSTATUS_IGNORED
          << (cantp_status_offset::PCANTP_STATUS_OFFSET_NET + 1))),
#pragma endregion

#pragma region ISO - TP extra information flags
    PCANTP_STATUS_CAUTION_INPUT_MODIFIED =
          (cantp_infostatus::PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED
          << cantp_status_offset::PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_DLC_MODIFIED =
          (cantp_infostatus::PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED
          << cantp_status_offset::PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_DATA_LENGTH_MODIFIED =
          (cantp_infostatus::PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED
          << cantp_status_offset::PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_FD_FLAG_MODIFIED =
          (cantp_infostatus::PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED
          << cantp_status_offset::PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_RX_QUEUE_FULL =
          (cantp_infostatus::PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL
          << cantp_status_offset::PCANTP_STATUS_OFFSET_INFO),
    PCANTP_STATUS_CAUTION_BUFFER_IN_USE =
          (cantp_infostatus::PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE
          << cantp_status_offset::PCANTP_STATUS_OFFSET_INFO),
#pragma endregion

#pragma region Lower API status code : see also PCANTP_STATUS_xx macros
    PCANTP_STATUS_FLAG_PCAN_STATUS = 0x80000000U,
#pragma endregion

#pragma region Masks to merge / retrieve different PCANTP status by type in a cantp_status
    PCANTP_STATUS_MASK_ERROR = 0x000000FFU,
    PCANTP_STATUS_MASK_BUS = 0x00001F00U,
    PCANTP_STATUS_MASK_ISOTP_NET = 0x0003E000U,
    PCANTP_STATUS_MASK_INFO = 0x00FC0000U,
    PCANTP_STATUS_MASK_PCAN = ~PCANTP_STATUS_FLAG_PCAN_STATUS,
#pragma endregion
};
```

C#

```
public enum cantp_status : UInt32
{
  PCANTP_STATUS_OK = cantp_errstatus.PCANTP_ERRSTATUS_OK,
  PCANTP_STATUS_NOT_INITIALIZED = cantp_errstatus.PCANTP_ERRSTATUS_NOT_INITIALIZED,
  PCANTP_STATUS_ALREADY_INITIALIZED = cantp_errstatus.PCANTP_ERRSTATUS_ALREADY_INITIALIZED,
  PCANTP_STATUS_NO_MEMORY = cantp_errstatus.PCANTP_ERRSTATUS_NO_MEMORY,
  PCANTP_STATUS_OVERFLOW = cantp_errstatus.PCANTP_ERRSTATUS_OVERFLOW,
  PCANTP_STATUS_NO_MESSAGE = cantp_errstatus.PCANTP_ERRSTATUS_NO_MESSAGE,
  PCANTP_STATUS_PARAM_INVALID_TYPE = cantp_errstatus.PCANTP_ERRSTATUS_PARAM_INVALID_TYPE,
  PCANTP_STATUS_PARAM_INVALID_VALUE = cantp_errstatus.PCANTP_ERRSTATUS_PARAM_INVALID_VALUE,
  PCANTP_STATUS_MAPPING_NOT_INITIALIZED =
        cantp_errstatus.PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED,
  PCANTP_STATUS_MAPPING_INVALID = cantp_errstatus.PCANTP_ERRSTATUS_MAPPING_INVALID,
  PCANTP_STATUS_MAPPING_ALREADY_INITIALIZED =
        cantp_errstatus.PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED,
  PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL =
        cantp_errstatus.PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL,
  PCANTP_STATUS_QUEUE_TX_FULL = cantp_errstatus.PCANTP_ERRSTATUS_QUEUE_TX_FULL,
  PCANTP_STATUS_LOCK_TIMEOUT = cantp_errstatus.PCANTP_ERRSTATUS_LOCK_TIMEOUT,
```

```
PCANTP_STATUS_HANDLE_INVALID = cantp_errstatus.PCANTP_ERRSTATUS_INVALID_HANDLE,
PCANTP_STATUS_UNKNOWN = cantp_errstatus.PCANTP_ERRSTATUS_UNKNOWN,

#region Bus status flags (bits [8..11])
PCANTP_STATUS_FLAG_BUS_LIGHT = (cantp_busstatus.PCANTP_BUSSTATUS_LIGHT
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS),
PCANTP_STATUS_FLAG_BUS_HEAVY = (cantp_busstatus.PCANTP_BUSSTATUS_HEAVY
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS),
PCANTP_STATUS_FLAG_BUS_WARNING = PCANTP_STATUS_FLAG_BUS_HEAVY,
PCANTP_STATUS_FLAG_BUS_PASSIVE = (cantp_busstatus.PCANTP_BUSSTATUS_PASSIVE
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS),
PCANTP_STATUS_FLAG_BUS_OFF = (cantp_busstatus.PCANTP_BUSSTATUS_OFF
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS),
PCANTP_STATUS_FLAG_BUS_ANY = (cantp_busstatus.PCANTP_BUSSTATUS_ANY
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS),
#endregion

#region Network status (bits [13..17])
PCANTP_STATUS_FLAG_NETWORK_RESULT = (1 << cantp_status_offset.PCANTP_STATUS_OFFSET_NET),
PCANTP_STATUS_NETWORK_TIMEOUT_A = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_TIMEOUT_A
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_TIMEOUT_Bs = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_TIMEOUT_Bs
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_TIMEOUT_Cr = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_TIMEOUT_Cr
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_WRONG_SN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_WRONG_SN
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_INVALID_FS = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_INVALID_FS
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_UNEXP_PDU = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_UNEXP_PDU
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_WFT_OVRN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_WFT_OVRN
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_BUFFER_OVFLW = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_BUFFER_OVFLW
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_ERROR = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_ERROR
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
PCANTP_STATUS_NETWORK_IGNORED = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      | (cantp_netstatus.PCANTP_NETSTATUS_IGNORED
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1))),
#endregion

#region ISO-TP extra information flags
PCANTP_STATUS_CAUTION_INPUT_MODIFIED =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO),
PCANTP_STATUS_CAUTION_DLC_MODIFIED = (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO),
PCANTP_STATUS_CAUTION_DATA_LENGTH_MODIFIED =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO),
PCANTP_STATUS_CAUTION_FD_FLAG_MODIFIED =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED
```

```
        << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO),
  PCANTP_STATUS_CAUTION_RX_QUEUE_FULL =
        (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL
        << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO),
  PCANTP_STATUS_CAUTION_BUFFER_IN_USE =
        (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE
        << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO),
  #endregion

  #region Lower API status code: see also PCANTP_STATUS_xx macros
  PCANTP_STATUS_FLAG_PCAN_STATUS = 0x80000000U,
  #endregion

  #region Masks to merge/retrieve different PCANTP status by type in a cantp_status
  PCANTP_STATUS_MASK_ERROR = 0x000000FFU,
  PCANTP_STATUS_MASK_BUS = 0x00001F00U,
  PCANTP_STATUS_MASK_ISOTP_NET = 0x0003E000U,
  PCANTP_STATUS_MASK_INFO = 0x00FC0000U,
  PCANTP_STATUS_MASK_PCAN = ~PCANTP_STATUS_FLAG_PCAN_STATUS,
  #endregion
}
```

Pascal OO

```
cantp_status = (
    PCANTP_STATUS_OK = UInt32(PCANTP_ERRSTATUS_OK),
    PCANTP_STATUS_NOT_INITIALIZED = UInt32(PCANTP_ERRSTATUS_NOT_INITIALIZED),
    PCANTP_STATUS_ALREADY_INITIALIZED = UInt32(PCANTP_ERRSTATUS_ALREADY_INITIALIZED),
    PCANTP_STATUS_NO_MEMORY = UInt32(PCANTP_ERRSTATUS_NO_MEMORY),
    PCANTP_STATUS_OVERFLOW = UInt32(PCANTP_ERRSTATUS_OVERFLOW),
    PCANTP_STATUS_NO_MESSAGE = UInt32(PCANTP_ERRSTATUS_NO_MESSAGE),
    PCANTP_STATUS_PARAM_INVALID_TYPE = UInt32(PCANTP_ERRSTATUS_PARAM_INVALID_TYPE),
    PCANTP_STATUS_PARAM_INVALID_VALUE = UInt32(PCANTP_ERRSTATUS_PARAM_INVALID_VALUE),
    PCANTP_STATUS_MAPPING_NOT_INITIALIZED = UInt32(PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED),
    PCANTP_STATUS_MAPPING_INVALID = UInt32(PCANTP_ERRSTATUS_MAPPING_INVALID),
    PCANTP_STATUS_MAPPING_ALREADY_INITIALIZED =
        UInt32(PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED),
    PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL = UInt32(PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL),
    PCANTP_STATUS_QUEUE_TX_FULL = UInt32(PCANTP_ERRSTATUS_QUEUE_TX_FULL),
    PCANTP_STATUS_LOCK_TIMEOUT = UInt32(PCANTP_ERRSTATUS_LOCK_TIMEOUT),
    PCANTP_STATUS_HANDLE_INVALID = UInt32(PCANTP_ERRSTATUS_INVALID_HANDLE),
    PCANTP_STATUS_UNKNOWN = UInt32(PCANTP_ERRSTATUS_UNKNOWN),
    PCANTP_STATUS_FLAG_BUS_LIGHT = UInt32(PCANTP_BUSSTATUS_LIGHT) Shl PCANTP_STATUS_OFFSET_BUS,
    PCANTP_STATUS_FLAG_BUS_HEAVY = UInt32(PCANTP_BUSSTATUS_HEAVY) Shl PCANTP_STATUS_OFFSET_BUS,
    PCANTP_STATUS_FLAG_BUS_WARNING = UInt32(PCANTP_STATUS_FLAG_BUS_HEAVY),
    PCANTP_STATUS_FLAG_BUS_PASSIVE = UInt32(PCANTP_BUSSTATUS_PASSIVE)
        Shl PCANTP_STATUS_OFFSET_BUS,
    PCANTP_STATUS_FLAG_BUS_OFF = UInt32(PCANTP_BUSSTATUS_OFF) Shl PCANTP_STATUS_OFFSET_BUS,
    PCANTP_STATUS_FLAG_BUS_ANY = UInt32(PCANTP_BUSSTATUS_ANY) Shl PCANTP_STATUS_OFFSET_BUS,
    PCANTP_STATUS_FLAG_NETWORK_RESULT = 1 Shl PCANTP_STATUS_OFFSET_NET,
    PCANTP_STATUS_NETWORK_TIMEOUT_A = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_TIMEOUT_A) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_TIMEOUT_Bs = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_TIMEOUT_Bs) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_TIMEOUT_Cr = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_TIMEOUT_Cr) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_WRONG_SN = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_WRONG_SN) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_INVALID_FS = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_INVALID_FS) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_UNEXP_PDU = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_UNEXP_PDU) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
```

```
    PCANTP_STATUS_NETWORK_WFT_OVRN = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_WFT_OVRN) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_BUFFER_OVFLW = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_BUFFER_OVFLW) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_ERROR = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_ERROR) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_NETWORK_IGNORED = UInt32(PCANTP_STATUS_FLAG_NETWORK_RESULT)
        Or (UInt32(PCANTP_NETSTATUS_IGNORED) Shl (PCANTP_STATUS_OFFSET_NET + 1)),
    PCANTP_STATUS_CAUTION_INPUT_MODIFIED = UInt32(PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED)
        Shl PCANTP_STATUS_OFFSET_INFO,
    PCANTP_STATUS_CAUTION_DLC_MODIFIED = UInt32(PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED)
        Shl PCANTP_STATUS_OFFSET_INFO,
    PCANTP_STATUS_CAUTION_DATA_LENGTH_MODIFIED =
        UInt32(PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED) Shl PCANTP_STATUS_OFFSET_INFO,
    PCANTP_STATUS_CAUTION_FD_FLAG_MODIFIED = UInt32(PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED)
        Shl PCANTP_STATUS_OFFSET_INFO,
    PCANTP_STATUS_CAUTION_RX_QUEUE_FULL = UInt32(PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL)
        Shl PCANTP_STATUS_OFFSET_INFO,
    PCANTP_STATUS_CAUTION_BUFFER_IN_USE = UInt32(PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE)
        Shl PCANTP_STATUS_OFFSET_INFO,
    PCANTP_STATUS_FLAG_PCAN_STATUS = UInt32($80000000),
    PCANTP_STATUS_MASK_ERROR = UInt32($000000FF),
    PCANTP_STATUS_MASK_BUS = UInt32($00001F00),
    PCANTP_STATUS_MASK_ISOTP_NET = UInt32($0003E000),
    PCANTP_STATUS_MASK_INFO = UInt32($00FC0000),
    PCANTP_STATUS_MASK_PCAN = Not PCANTP_STATUS_FLAG_PCAN_STATUS
);
```

Visual Basic

```
Public Enum cantp_status As UInt32
  PCANTP_STATUS_OK = cantp_errstatus.PCANTP_ERRSTATUS_OK
  PCANTP_STATUS_NOT_INITIALIZED = cantp_errstatus.PCANTP_ERRSTATUS_NOT_INITIALIZED
  PCANTP_STATUS_ALREADY_INITIALIZED = cantp_errstatus.PCANTP_ERRSTATUS_ALREADY_INITIALIZED
  PCANTP_STATUS_NO_MEMORY = cantp_errstatus.PCANTP_ERRSTATUS_NO_MEMORY
  PCANTP_STATUS_OVERFLOW = cantp_errstatus.PCANTP_ERRSTATUS_OVERFLOW
  PCANTP_STATUS_NO_MESSAGE = cantp_errstatus.PCANTP_ERRSTATUS_NO_MESSAGE
  PCANTP_STATUS_PARAM_INVALID_TYPE = cantp_errstatus.PCANTP_ERRSTATUS_PARAM_INVALID_TYPE
  PCANTP_STATUS_PARAM_INVALID_VALUE = cantp_errstatus.PCANTP_ERRSTATUS_PARAM_INVALID_VALUE
  PCANTP_STATUS_MAPPING_NOT_INITIALIZED =
      cantp_errstatus.PCANTP_ERRSTATUS_MAPPING_NOT_INITIALIZED
  PCANTP_STATUS_MAPPING_INVALID = cantp_errstatus.PCANTP_ERRSTATUS_MAPPING_INVALID
  PCANTP_STATUS_MAPPING_ALREADY_INITIALIZED =
      cantp_errstatus.PCANTP_ERRSTATUS_MAPPING_ALREADY_INITIALIZED
  PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL =
      cantp_errstatus.PCANTP_ERRSTATUS_PARAM_BUFFER_TOO_SMALL
  PCANTP_STATUS_QUEUE_TX_FULL = cantp_errstatus.PCANTP_ERRSTATUS_QUEUE_TX_FULL
  PCANTP_STATUS_LOCK_TIMEOUT = cantp_errstatus.PCANTP_ERRSTATUS_LOCK_TIMEOUT
  PCANTP_STATUS_HANDLE_INVALID = cantp_errstatus.PCANTP_ERRSTATUS_INVALID_HANDLE
  PCANTP_STATUS_UNKNOWN = cantp_errstatus.PCANTP_ERRSTATUS_UNKNOWN

#Region "Bus status flags (bits [8..11])"
  PCANTP_STATUS_FLAG_BUS_LIGHT = (cantp_busstatus.PCANTP_BUSSTATUS_LIGHT
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS)
  PCANTP_STATUS_FLAG_BUS_HEAVY = (cantp_busstatus.PCANTP_BUSSTATUS_HEAVY
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS)
  PCANTP_STATUS_FLAG_BUS_WARNING = PCANTP_STATUS_FLAG_BUS_HEAVY
  PCANTP_STATUS_FLAG_BUS_PASSIVE = (cantp_busstatus.PCANTP_BUSSTATUS_PASSIVE
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS)
  PCANTP_STATUS_FLAG_BUS_OFF = (cantp_busstatus.PCANTP_BUSSTATUS_OFF
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS)
  PCANTP_STATUS_FLAG_BUS_ANY = (cantp_busstatus.PCANTP_BUSSTATUS_ANY
```

```
      << cantp_status_offset.PCANTP_STATUS_OFFSET_BUS)
#End Region

#Region "Network status (bits [13..17])"
  PCANTP_STATUS_FLAG_NETWORK_RESULT = (1 << cantp_status_offset.PCANTP_STATUS_OFFSET_NET)
  PCANTP_STATUS_NETWORK_TIMEOUT_A = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_TIMEOUT_A
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_TIMEOUT_Bs = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_TIMEOUT_Bs
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_TIMEOUT_Cr = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_TIMEOUT_Cr
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_WRONG_SN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_WRONG_SN
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_INVALID_FS = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_INVALID_FS
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_UNEXP_PDU = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_UNEXP_PDU
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_WFT_OVRN = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_WFT_OVRN
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_BUFFER_OVFLW = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_BUFFER_OVFLW
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_ERROR = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_ERROR
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
  PCANTP_STATUS_NETWORK_IGNORED = (PCANTP_STATUS_FLAG_NETWORK_RESULT
      Or (cantp_netstatus.PCANTP_NETSTATUS_IGNORED
      << (cantp_status_offset.PCANTP_STATUS_OFFSET_NET + 1)))
#End Region

#Region "ISO-TP extra information flags"
  PCANTP_STATUS_CAUTION_INPUT_MODIFIED =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_INPUT_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO)
  PCANTP_STATUS_CAUTION_DLC_MODIFIED = (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_DLC_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO)
  PCANTP_STATUS_CAUTION_DATA_LENGTH_MODIFIED =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_DATA_LENGTH_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO)
  PCANTP_STATUS_CAUTION_FD_FLAG_MODIFIED =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_FD_FLAG_MODIFIED
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO)
  PCANTP_STATUS_CAUTION_RX_QUEUE_FULL =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_RX_QUEUE_FULL
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO)
  PCANTP_STATUS_CAUTION_BUFFER_IN_USE =
      (cantp_infostatus.PCANTP_INFOSTATUS_CAUTION_BUFFER_IN_USE
      << cantp_status_offset.PCANTP_STATUS_OFFSET_INFO)
#End Region

#Region "Lower API status code: see also PCANTP_STATUS_xx macros"
  PCANTP_STATUS_FLAG_PCAN_STATUS = &H80000000UI
#End Region

#Region "Masks To merge/retrieve different PCANTP status by type In a cantp_status"
```

```
  PCANTP_STATUS_MASK_ERROR = &HFFUI
  PCANTP_STATUS_MASK_BUS = &H1F00UI
  PCANTP_STATUS_MASK_ISOTP_NET = &H3E000UI
  PCANTP_STATUS_MASK_INFO = &HFC0000UI
  PCANTP_STATUS_MASK_PCAN = Not PCANTP_STATUS_FLAG_PCAN_STATUS
#End Region
End Enum
```

Values

Name	Value	Description
PCANTP_STATUS_OK	0x00000000 (0)	No error.
PCANTP_STATUS_NOT_INITIALIZED	0x00000001 (1)	Not Initialized.
PCANTP_STATUS_ALREADY_INITIALIZED	0x00000002 (2)	Already Initialized.
PCANTP_STATUS_NO_MEMORY	0x00000003 (3)	Could not obtain memory.
PCANTP_STATUS_OVERFLOW	0x00000004 (4)	Input buffer overflow.
PCANTP_STATUS_NO_MESSAGE	0x00000007 (5)	No message available.
PCANTP_STATUS_PARAM_INVALID_TYPE	0x00000008 (8)	Parameter has an invalid or unexpected type.
PCANTP_STATUS_PARAM_INVALID_VALUE	0x00000009 (9)	Parameter has an invalid value.
PCANTP_STATUS_MAPPING_NOT_INITIALIZED	0x0000000D (13)	PCANTP mapping not initialized.
PCANTP_STATUS_MAPPING_INVALID	0x0000000E (14)	PCANTP mapping parameters are invalid.
PCANTP_STATUS_MAPPING_ALREADY_INITIALIZED	0x0000000F (15)	PCANTP mapping already defined.
PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL	0x00000010 (16)	Parameter buffer is too small.
PCANTP_STATUS_QUEUE_TX_FULL	0x00000011 (17)	Transmit queue is full.
PCANTP_STATUS_LOCK_TIMEOUT	0x00000012 (18)	Failed to get an access to the internal lock.
PCANTP_STATUS_HANDLE_INVALID	0x00000013 (19)	Invalid cantp_handle.
PCANTP_STATUS_UNKNOWN	0x000000FF (255)	Unknown/generic error.
PCANTP_STATUS_FLAG_BUS_LIGHT	0x00000100 (256)	PCANTP Channel is in BUS-LIGHT error state.
PCANTP_STATUS_FLAG_BUS_HEAVY	0x00000200 (512)	PCANTP Channel is in BUS-HEAVY error state.
PCANTP_STATUS_FLAG_BUS_WARNING	0x00000200 (512)	PCANTP Channel is in BUS-HEAVY error state.
PCANTP_STATUS_FLAG_BUS_PASSIVE	0x00000400 (1024)	PCANTP Channel is error passive state.
PCANTP_STATUS_FLAG_BUS_OFF	0x00000800 (2048)	PCANTP Channel is in BUS-OFF error state.
PCANTP_STATUS_FLAG_BUS_ANY	0x00000F00 (3840)	Mask for all bus errors.
PCANTP_STATUS_FLAG_NETWORK_RESULT	0x00002000 (8192)	This flag states if one of the following network errors occured with the fetched message.
PCANTP_STATUS_NETWORK_TIMEOUT_A	0x00006000 (24576)	Timeout occured between 2 frames transmission (sender and receiver side).
PCANTP_STATUS_NETWORK_TIMEOUT_Bs	0x0000A000 (40960)	Sender side timeout while waiting for flow control frame.

Name	Value	Description
PCANTP_STATUS_NETWORK_TIMEOUT_Cr	0x0000E000 (57344)	Receiver side timeout while waiting for consecutive frame.
PCANTP_STATUS_NETWORK_WRONG_SN	0x00012000 (73728)	Unexpected sequence number.
PCANTP_STATUS_NETWORK_INVALID_FS	0x00016000 (90112)	Invalid or unknown FlowStatus.
PCANTP_STATUS_NETWORK_UNEXP_PDU	0x0001A000 (106496)	Unexpected protocol data unit.
PCANTP_STATUS_NETWORK_WFT_OVRN	0x0001E000 (122880)	Reception of flow control WAIT frame that exceeds the maximum counter defined by PCANTP_PARAMETER_WFT_MAX
PCANTP_STATUS_NETWORK_BUFFER_OVFLW	0x00022000 (139264)	Buffer on the receiver side cannot store the data length (server side only).
PCANTP_STATUS_NETWORK_ERROR	0x00026000 (155648)	General error.
PCANTP_STATUS_NETWORK_IGNORED	0x0002A000 (172032)	Message was invalid and ignored.
PCANTP_STATUS_NETWORK_TIMEOUT_Ar	0x00046000 (286720)	Sender side timeout while transmitting.
PCANTP_STATUS_NETWORK_TIMEOUT_As	0x00042000 (270336)	Receiver side timeout while transmitting.
PCANTP_STATUS_CAUTION_INPUT_MODIFIED	0x00040000 (262144)	Input was modified.
PCANTP_STATUS_CAUTION_DLC_MODIFIED	0x00080000 (524288)	DLC value of the input was modified.
PCANTP_STATUS_CAUTION_DATA_LENGTH_MODIFIED	0x00100000 (1048576)	Data Length value of the input was modified.
PCANTP_STATUS_CAUTION_FD_FLAG_MODIFIED	0x00200000 (2097152)	FD flags of the input was modified.
PCANTP_STATUS_CAUTION_RX_QUEUE_FULL	0x00400000 (4194304)	Receive queue is full.
PCANTP_STATUS_CAUTION_BUFFER_IN_USE	0x00800000 (8388608)	Buffer is used by another thread or API.
PCANTP_STATUS_FLAG_PCAN_STATUS	0x80000000 (2147483648)	PCAN error flag, remove flag to get a usable PCAN error/status code (cf. PCANBasic API).
PCANTP_STATUS_MASK_ERROR	0x000000FF (255)	Filter by PCANTP_STATUSTYPE_ERR type (see cantp_statustype on page 51).
PCANTP_STATUS_MASK_BUS	0x00001F00 (7936)	Filter by PCANTP_STATUSTYPE_BUS type (see cantp_statustype on page 51).
PCANTP_STATUS_MASK_ISOTP_NET	0x0003E000 (253952)	Filter by PCANTP_STATUSTYPE_NET type (see cantp_statustype on page 51).
PCANTP_STATUS_MASK_INFO	0x00FC0000 (786432)	Filter by PCANTP_STATUSTYPE_INFO type (see cantp_statustype on page 51).
PCANTP_STATUS_MASK_PCAN	0x7fffffff (2147483647)	Filter by PCANTP_STATUSTYPE_PCAN type (see cantp_statustype on page 51).

Remarks

- The PCANTP_STATUS_FLAG_PCAN_STATUS 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_STATUS_MASK_PCAN and the PCAN-Basic (TPCANStatus) error.
- Bits information are distributed bit by bit like it follows:

Status bits																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	000 0000							0000 00						00 000					0 0000					0000 0000							
PCANBasic error flag	Reserved							API extra information						Networking message status					CAN Bus status					PCAN-ISOTP API errors							

3.5.14 cantp_parameter

Represents a PCAN-ISO-TP parameter or a PCAN-ISO-TP value that can be read or set. With some exceptions, a channel must first be initialized before their parameters can be read or set. PCAN-Basic parameters (PCAN_PARAM_xxx) are compatible via casting.

Syntax

C++

```
typedef enum _cantp_parameter {
	PCANTP_PARAMETER_API_VERSION = 0x101,
	PCANTP_PARAMETER_CHANNEL_CONDITION = 0x102,
	PCANTP_PARAMETER_DEBUG = 0x103,
	PCANTP_PARAMETER_RECEIVE_EVENT = 0x104,
	PCANTP_PARAMETER_FRAME_FILTERING = 0x105,
	PCANTP_PARAMETER_CAN_TX_DL = 0x106,
	PCANTP_PARAMETER_CAN_DATA_PADDING = 0x107,
	PCANTP_PARAMETER_CAN_PADDING_VALUE = 0x108,
	PCANTP_PARAMETER_ISO_REV = 0x109,
	PCANTP_PARAMETER_J1939_PRIORITY = 0x10A,
	PCANTP_PARAMETER_MSG_PENDING = 0x10B,
	PCANTP_PARAMETER_BLOCK_SIZE = 0x10C,
	PCANTP_PARAMETER_BLOCK_SIZE_TX = 0x10D,
	PCANTP_PARAMETER_SEPARATION_TIME = 0x10E,
	PCANTP_PARAMETER_SEPARATION_TIME_TX = 0x10F,
	PCANTP_PARAMETER_WFT_MAX = 0x110,
	PCANTP_PARAMETER_TIMEOUT_AS = 0x111,
	PCANTP_PARAMETER_TIMEOUT_AR = 0x112,
	PCANTP_PARAMETER_TIMEOUT_BS = 0x113,
	PCANTP_PARAMETER_TIMEOUT_CR = 0x114,
	PCANTP_PARAMETER_TIMEOUT_TOLERANCE = 0x115,
	PCANTP_PARAMETER_ISO_TIMEOUTS = 0x116,
	PCANTP_PARAMETER_SELFRECEIVE_LATENCY = 0x117,
	PCANTP_PARAMETER_MAX_RX_QUEUE = 0x118,
	PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES = 0x119,
	PCANTP_PARAMETER_FILTER_CAN_ID = 0x11A,
	PCANTP_PARAMETER_SUPPORT_29B_ENHANCED = 0x11B,
	PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL = 0x11C,
	PCANTP_PARAMETER_SUPPORT_29B_MIXED = 0x11D,
	PCANTP_PARAMETER_MSG_CHECK = 0x11E,
	PCANTP_PARAMETER_RESET_HARD = 0x11F,
```

```
        PCANTP_PARAMETER_NETWORK_LAYER_DESIGN = 0x120,
        PCANTP_PARAMETER_HARDWARE_NAME = PCAN_HARDWARE_NAME,
        PCANTP_PARAMETER_DEVICE_ID = PCAN_DEVICE_ID,
        PCANTP_PARAMETER_CONTROLLER_NUMBER = PCAN_CONTROLLER_NUMBER,
        PCANTP_PARAMETER_CHANNEL_FEATURES = PCAN_CHANNEL_FEATURES
} cantp_parameter;
```

C#

```
public enum cantp_parameter : UInt32
{
    PCANTP_PARAMETER_API_VERSION = 0x101,
    PCANTP_PARAMETER_CHANNEL_CONDITION = 0x102,
    PCANTP_PARAMETER_DEBUG = 0x103,
    PCANTP_PARAMETER_RECEIVE_EVENT = 0x104,
    PCANTP_PARAMETER_FRAME_FILTERING = 0x105,
    PCANTP_PARAMETER_CAN_TX_DL = 0x106,
    PCANTP_PARAMETER_CAN_DATA_PADDING = 0x107,
    PCANTP_PARAMETER_CAN_PADDING_VALUE = 0x108,
    PCANTP_PARAMETER_ISO_REV = 0x109,
    PCANTP_PARAMETER_J1939_PRIORITY = 0x10A,
    PCANTP_PARAMETER_MSG_PENDING = 0x10B,
    PCANTP_PARAMETER_BLOCK_SIZE = 0x10C,
    PCANTP_PARAMETER_BLOCK_SIZE_TX = 0x10D,
    PCANTP_PARAMETER_SEPARATION_TIME = 0x10E,
    PCANTP_PARAMETER_SEPARATION_TIME_TX = 0x10F,
    PCANTP_PARAMETER_WFT_MAX = 0x110,
    PCANTP_PARAMETER_TIMEOUT_AS = 0x111,
    PCANTP_PARAMETER_TIMEOUT_AR = 0x112,
    PCANTP_PARAMETER_TIMEOUT_BS = 0x113,
    PCANTP_PARAMETER_TIMEOUT_CR = 0x114,
    PCANTP_PARAMETER_TIMEOUT_TOLERANCE = 0x115,
    PCANTP_PARAMETER_ISO_TIMEOUTS = 0x116,
    PCANTP_PARAMETER_SELFRECEIVE_LATENCY = 0x117,
    PCANTP_PARAMETER_MAX_RX_QUEUE = 0x118,
    PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES = 0x119,
    PCANTP_PARAMETER_FILTER_CAN_ID = 0x11A,
    PCANTP_PARAMETER_SUPPORT_29B_ENHANCED = 0x11B,
    PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL = 0x11C,
    PCANTP_PARAMETER_SUPPORT_29B_MIXED = 0x11D,
    PCANTP_PARAMETER_MSG_CHECK = 0x11E,
    PCANTP_PARAMETER_RESET_HARD = 0x11F,
    PCANTP_PARAMETER_NETWORK_LAYER_DESIGN = 0x120,
    PCANTP_PARAMETER_HARDWARE_NAME = TPCANParameter.PCAN_HARDWARE_NAME,
    PCANTP_PARAMETER_DEVICE_ID = TPCANParameter.PCAN_DEVICE_NUMBER,
    PCANTP_PARAMETER_CONTROLLER_NUMBER = TPCANParameter.PCAN_CONTROLLER_NUMBER,
    PCANTP_PARAMETER_CHANNEL_FEATURES = TPCANParameter.PCAN_CHANNEL_FEATURES
}
```

C++ / CLR

```
public enum cantp_parameter : UInt32
{
        PCANTP_PARAMETER_API_VERSION = 0x101,
        PCANTP_PARAMETER_CHANNEL_CONDITION = 0x102,
        PCANTP_PARAMETER_DEBUG = 0x103,
        PCANTP_PARAMETER_RECEIVE_EVENT = 0x104,
        PCANTP_PARAMETER_FRAME_FILTERING = 0x105,
        PCANTP_PARAMETER_CAN_TX_DL = 0x106,
        PCANTP_PARAMETER_CAN_DATA_PADDING = 0x107,
        PCANTP_PARAMETER_CAN_PADDING_VALUE = 0x108,
        PCANTP_PARAMETER_ISO_REV = 0x109,
        PCANTP_PARAMETER_J1939_PRIORITY = 0x10A,
```

```
        PCANTP_PARAMETER_MSG_PENDING = 0x10B,
        PCANTP_PARAMETER_BLOCK_SIZE = 0x10C,
        PCANTP_PARAMETER_BLOCK_SIZE_TX = 0x10D,
        PCANTP_PARAMETER_SEPARATION_TIME = 0x10E,
        PCANTP_PARAMETER_SEPARATION_TIME_TX = 0x10F,
        PCANTP_PARAMETER_WFT_MAX = 0x110,
        PCANTP_PARAMETER_TIMEOUT_AS = 0x111,
        PCANTP_PARAMETER_TIMEOUT_AR = 0x112,
        PCANTP_PARAMETER_TIMEOUT_BS = 0x113,
        PCANTP_PARAMETER_TIMEOUT_CR = 0x114,
        PCANTP_PARAMETER_TIMEOUT_TOLERANCE = 0x115,
        PCANTP_PARAMETER_ISO_TIMEOUTS = 0x116,
        PCANTP_PARAMETER_SELFRECEIVE_LATENCY = 0x117,
        PCANTP_PARAMETER_MAX_RX_QUEUE = 0x118,
        PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES = 0x119,
        PCANTP_PARAMETER_FILTER_CAN_ID = 0x11A,
        PCANTP_PARAMETER_SUPPORT_29B_ENHANCED = 0x11B,
        PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL = 0x11C,
        PCANTP_PARAMETER_SUPPORT_29B_MIXED = 0x11D,
        PCANTP_PARAMETER_MSG_CHECK = 0x11E,
        PCANTP_PARAMETER_RESET_HARD = 0x11F,
        PCANTP_PARAMETER_NETWORK_LAYER_DESIGN = 0x120,
        PCANTP_PARAMETER_HARDWARE_NAME = (UInt32)TPCANParameter::PCAN_HARDWARE_NAME,
        PCANTP_PARAMETER_DEVICE_ID = (UInt32)TPCANParameter::PCAN_DEVICE_NUMBER,
        PCANTP_PARAMETER_CONTROLLER_NUMBER = (UInt32)TPCANParameter::PCAN_CONTROLLER_NUMBER,
        PCANTP_PARAMETER_CHANNEL_FEATURES = (UInt32)TPCANParameter::PCAN_CHANNEL_FEATURES
};
```

Pascal OO

```
cantp_parameter = (
    PCANTP_PARAMETER_API_VERSION = $101,
    PCANTP_PARAMETER_CHANNEL_CONDITION = $102,
    PCANTP_PARAMETER_DEBUG = $103,
    PCANTP_PARAMETER_RECEIVE_EVENT = $104,
    PCANTP_PARAMETER_FRAME_FILTERING = $105,
    PCANTP_PARAMETER_CAN_TX_DL = $106,
    PCANTP_PARAMETER_CAN_DATA_PADDING = $107,
    PCANTP_PARAMETER_CAN_PADDING_VALUE = $108,
    PCANTP_PARAMETER_ISO_REV = $109,
    PCANTP_PARAMETER_J1939_PRIORITY = $10A,
    PCANTP_PARAMETER_MSG_PENDING = $10B,
    PCANTP_PARAMETER_BLOCK_SIZE = $10C,
    PCANTP_PARAMETER_BLOCK_SIZE_TX = $10D,
    PCANTP_PARAMETER_SEPARATION_TIME = $10E,
    PCANTP_PARAMETER_SEPARATION_TIME_TX = $10F,
    PCANTP_PARAMETER_WFT_MAX = $110,
    PCANTP_PARAMETER_TIMEOUT_AS = $111,
    PCANTP_PARAMETER_TIMEOUT_AR = $112,
    PCANTP_PARAMETER_TIMEOUT_BS = $113,
    PCANTP_PARAMETER_TIMEOUT_CR = $114,
    PCANTP_PARAMETER_TIMEOUT_TOLERANCE = $115,
    PCANTP_PARAMETER_ISO_TIMEOUTS = $116,
    PCANTP_PARAMETER_SELFRECEIVE_LATENCY = $117,
    PCANTP_PARAMETER_MAX_RX_QUEUE = $118,
    PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES = $119,
    PCANTP_PARAMETER_FILTER_CAN_ID = $11A,
    PCANTP_PARAMETER_SUPPORT_29B_ENHANCED = $11B,
    PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL = $11C,
    PCANTP_PARAMETER_SUPPORT_29B_MIXED = $11D,
    PCANTP_PARAMETER_MSG_CHECK = $11E,
    PCANTP_PARAMETER_RESET_HARD = $11F,
```

```
    PCANTP_PARAMETER_NETWORK_LAYER_DESIGN = $120,
    PCANTP_PARAMETER_HARDWARE_NAME = UInt32(PCAN_HARDWARE_NAME),
    PCANTP_PARAMETER_DEVICE_ID = UInt32(PCAN_DEVICE_NUMBER),
    PCANTP_PARAMETER_CONTROLLER_NUMBER = UInt32(PCAN_CONTROLLER_NUMBER),
    PCANTP_PARAMETER_CHANNEL_FEATURES = UInt32(PCAN_CHANNEL_FEATURES)
);
```

Visual Basic

```
Public Enum cantp_parameter As UInt32
    PCANTP_PARAMETER_API_VERSION = &H101
    PCANTP_PARAMETER_CHANNEL_CONDITION = &H102
    PCANTP_PARAMETER_DEBUG = &H103
    PCANTP_PARAMETER_RECEIVE_EVENT = &H104
    PCANTP_PARAMETER_FRAME_FILTERING = &H105
    PCANTP_PARAMETER_CAN_TX_DL = &H106
    PCANTP_PARAMETER_CAN_DATA_PADDING = &H107
    PCANTP_PARAMETER_CAN_PADDING_VALUE = &H108
    PCANTP_PARAMETER_ISO_REV = &H109
    PCANTP_PARAMETER_J1939_PRIORITY = &H10A
    PCANTP_PARAMETER_MSG_PENDING = &H10B
    PCANTP_PARAMETER_BLOCK_SIZE = &H10C
    PCANTP_PARAMETER_BLOCK_SIZE_TX = &H10D
    PCANTP_PARAMETER_SEPARATION_TIME_TX = &H10F
    PCANTP_PARAMETER_WFT_MAX = &H110
    PCANTP_PARAMETER_TIMEOUT_AS = &H111
    PCANTP_PARAMETER_TIMEOUT_AR = &H112
    PCANTP_PARAMETER_TIMEOUT_BS = &H113
    PCANTP_PARAMETER_TIMEOUT_CR = &H114
    PCANTP_PARAMETER_TIMEOUT_TOLERANCE = &H115
    PCANTP_PARAMETER_ISO_TIMEOUTS = &H116
    PCANTP_PARAMETER_SELFRECEIVE_LATENCY = &H117
    PCANTP_PARAMETER_MAX_RX_QUEUE = &H118
    PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES = &H119
    PCANTP_PARAMETER_FILTER_CAN_ID = &H11A,
    PCANTP_PARAMETER_SUPPORT_29B_ENHANCED = &H11B
    PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL = &H11C
    PCANTP_PARAMETER_SUPPORT_29B_MIXED = &H11D
    PCANTP_PARAMETER_MSG_CHECK = &H11E
    PCANTP_PARAMETER_RESET_HARD = &H11F
    PCANTP_PARAMETER_NETWORK_LAYER_DESIGN = &H120
    PCANTP_PARAMETER_HARDWARE_NAME = TPCANParameter.PCAN_HARDWARE_NAME
    PCANTP_PARAMETER_DEVICE_ID = TPCANParameter.PCAN_DEVICE_NUMBER
    PCANTP_PARAMETER_CONTROLLER_NUMBER = TPCANParameter.PCAN_CONTROLLER_NUMBER
    PCANTP_PARAMETER_CHANNEL_FEATURES = TPCANParameter.PCAN_CHANNEL_FEATURES
End Enum
```

Values

Name	Value	Data type	Description
PCANTP_PARAMETER_API_VERSION	0x101	String	API version.
PCANTP_PARAMETER_CHANNEL_CONDITION	0x102	Byte	PCAN-ISO-TP channel condition.
PCANTP_PARAMETER_DEBUG	0x103	Byte	Debug mode.
PCANTP_PARAMETER_RECEIVE_EVENT	0x104	Pointer	PCAN-ISO-TP receive event handler parameter. Data is pointer to a HANDLE created by CreateEvent function.
PCANTP_PARAMETER_FRAME_FILTERING	0x105	Byte	Defines if unsegmented (NON-ISO-TP) CAN frames can be received.
PCANTP_PARAMETER_CAN_TX_DL	0x106	Byte	The maximum Data Length Code (DLC) of the fragmented frames used when transmitting FD messages.

Name	Value	Data type	Description
PCANTP_PARAMETER_CAN_DATA_PADDING	0x107	Byte	ISO-TP CAN frame data handling mode. Define if CAN frame DLC uses padding or not.
PCANTP_PARAMETER_CAN_PADDING_VALUE	0x108	Byte	Value used when CAN Data padding is enabled.
PCANTP_PARAMETER_ISO_REV	0x109	Byte	Defines which revision of ISO 15765-2 to use (see PCANTP_ISO_REV_*).
PCANTP_PARAMETER_J1939_PRIORITY	0x10A	Byte	Priority value (for normal fixed, mixed and enhanced addressing) used when ISO-TP J1939 compliant messages are sent. (default=6).
PCANTP_PARAMETER_MSG_PENDING	0x10B	Byte	Filter for message indication. Determine if pending messages are displayed/hidden.
PCANTP_PARAMETER_BLOCK_SIZE	0x10C	Byte	ISO-TP "BlockSize" (BS) parameter.
PCANTP_PARAMETER_BLOCK_SIZE_TX	0x10D	Int16	Defines the transmit block size parameter (BS_TX).
PCANTP_PARAMETER_SEPARATION_TIME	0x10E	Byte	ISO-TP "SeparationTime" (STmin) parameter.
PCANTP_PARAMETER_SEPARATION_TIME_TX	0x10D	Int16	Defines the transmit seperation time parameter (STmin_TX).
PCANTP_PARAMETER_WFT_MAX	0x110	Int32	ISO-TP "N_WFTmax" parameter.
PCANTP_PARAMETER_TIMEOUT_AS	0x111	Int32	Defines ISO-15765-2:Timeout. Timeout between 2 frames (sender side)
PCANTP_PARAMETER_TIMEOUT_AR	0x112	Int32	Defines ISO-15765-2:Timeout Ar. Timeout between 2 frames (receiver side)
PCANTP_PARAMETER_TIMEOUT_BS	0x113	Int32	Defines ISO-15765-2:Timeout Bs. Sender side timeout while waiting for flow control frame.
PCANTP_PARAMETER_TIMEOUT_CR	0x114	Int32	Defines ISO-15765-2:Timeout Cr. Receiver side timeout while waiting for consecutive frame.
PCANTP_PARAMETER_TIMEOUT_TOLERANCE	0x115	Int32	Defines the tolerence to apply to all timeout as a percentage ([0..100].
PCANTP_PARAMETER_ISO_TIMEOUTS	0x116	Byte	Sets predefined ISO values for timeouts.
PCANTP_PARAMETER_SELFRECEIVE_LATENCY	0x117	Byte	Sets optimization options to improve delay between ISO-TP consecutive frames.
PCANTP_PARAMETER_MAX_RX_QUEUE	0x118	Int16	Defines the maximum number of messages in the Rx queue.
PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES	0x119	Byte	Defines if messages handled by higher layer APIs are still available in this API (default=0).
PCANTP_PARAMETER_FILTER_CAN_ID	0x11A	Byte	Defines if the white-list CAN IDs filtering mechanism is enabled.
PCANTP_PARAMETER_SUPPORT_29B_ENHANCED	0x11B	Byte	Defines if the 29 bit Enhanced Diagnostic CAN identifier is supported (ISO-15765-3:2004, default is false with ISO revision 2016).
PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL	0x11C	Byte	Defines if the 29 bit Fixed Normal addressing CAN identifier is supported (default is true).
PCANTP_PARAMETER_SUPPORT_29B_MIXED	0x11D	Byte	Defines if the 29 bit Mixed addressing CAN identifier is supported (default is true).
PCANTP_PARAMETER_MSG_CHECK	0x11E	cantp_msg *	Checks if the message is valid and can be sent (ex. if a mapping is needed) and corrects input if needed.
PCANTP_PARAMETER_RESET_HARD	0x11F	Byte	Resets the CAN controller and clears Rx/Tx queues without uninitializing mapping and defined settings.
PCANTP_PARAMETER_NETWORK_LAYER_DESIGN	0x120	Byte	Defines if network is full-duplex (default) or half-duplex
PCANTP_PARAMETER_HARDWARE_NAME	0x00E	String	PCAN hardware name parameter.
PCANTP_PARAMETER_DEVICE_ID	0x001	Int32	PCAN-USB device identifier parameter.
PCANTP_PARAMETER_CONTROLLER_NUMBER	0x010	Int32	CAN-Controller number of a PCAN-Channel.
PCANTP_PARAMETER_CHANNEL_FEATURES	0x016	Int32	Capabilities of a PCAN device (FEATURE_***).

Detailed parameters values

PCANTP_PARAMETER_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_HANDLE_NONEBUS channel.

PCANTP_PARAMETER_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.

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

Default value: NA.

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

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

PCANTP_PARAMETER_DEBUG

Access: R W

Description: This parameter is used to control debug mode. If enabled, any received or transmitted CAN frames will be logged in 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. PCANTP_DEBUG_NOTICE, PCANTP_DEBUG_INFO, PCANTP_DEBUG_WARNING, PCANTP_DEBUG_ERROR enables it and adds extra filtering.

	Type	Constant	Value	Description
	Byte	PCANTP_DEBUG_NONE	0	No CAN debug messages are being generated.
	Byte	PCANTP_DEBUG_CAN	1	CAN debug messages are written to log file.
	Byte	PCANTP_DEBUG_NOTICE	0xF4	CAN debug messages are written to log file (only notices, informations, warnings, errors).

	Byte	PCANTP_DEBUG_INFO	0xF3	CAN debug messages are written to log file (only informations, warnings, errors).
	Byte	PCANTP_DEBUG_WARNING	0xF2	CAN debug messages are written to log file (only warnings, errors).
	Byte	PCANTP_DEBUG_ERROR	0xF1	CAN debug messages are written to log file (only errors).

Default value: PCANTP_DEBUG_NONE.

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

PCANTP_PARAMETER_RECEIVE_EVENT

Access: R W

Description: This parameter is used to let the PCAN-ISO-TP 2016 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 on page 259.

Default value: Disabled (0).

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

PCANTP_PARAMETER_FRAME_FILTERING

Access: R W

Description: This parameter is used to define how the API will handle non ISO-TP messages. The default behavior is PCANTP_FRAME_FILTERING_CAN: CAN/CAN FD frames can be received along with ISO-TP messages, but segmented frames composing an ISO-TP message are discarded. If PCANTP_FRAME_FILTERING_ISOTP is set, all non ISO-TP message will be discarded.

Finally, if the parameter is set to PCANTP_FRAME_FILTERING_VERBOSE 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_FRAME_FILTERING_ISOTP, PCANTP_FRAME_FILTERING_CAN, PCANTP_FRAME_FILTERING_VERBOSE.

	Type	Constant	Value	Description
	Byte	PCANTP_FRAME_FILTERING_ISOTP	0x00	Reception of unformatted (NON-ISO-TP) CAN frames is disabled.
	Byte	PCANTP_FRAME_FILTERING_CAN	0x01	Enable reception of unformatted (NON-ISO-TP) CAN frames. Received messages will be treated as either ISO 15765 or as an unformatted CAN frame.
	Byte	PCANTP_FRAME_FILTERING_VERBOSE	0x02	Enable reception of unformatted (NON-ISO-TP) CAN frames received messages will be treated as ISO 15765, unformatted CAN frame, or both (user will be able to read fragmented CAN frames).

Default value: 0x00 (PCANTP_FRAME_FILTERING_ISOTP).

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

PCANTP_PARAMETER_CAN_TX_DL

Access: R W

Description: This parameter is used to define the default maximum Data Length Code (DLC) used when transmitting ISO-TP messages with CAN FD enabled: the fragmented CAN FD frames composing the full CAN ISO-TP message will have at most a length corresponding to that DLC (it depends if the data fit in a lower DLC value). Note that member can_tx_dlc in cantp_mapping or message's can_info.dlc can be used to override this parameter locally.

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

Default value: 8 (0x08).

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

PCANTP_PARAMETER_CAN_DATA_PADDING

Access: R W

Description: This parameter is used to define if the API should use 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.

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

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_HANDLE_NONEBUS channel).

PCANTP_PARAMETER_CAN_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_HANDLE_NONEBUS channel).

PCANTP_PARAMETER_ISO_REV

Access: R W

Description: Define which revision of ISO 15765-2 to use.

Possible values: ISO-15765-2:2004(E) or ISO-15765-2:2016(E).

	Type	Constant	Value	Description
	Byte	PCANTP_ISO_REV_2004	0x01	ISO-15765-2:2004(E)
	Byte	PCANTP_ISO_REV_2016	0x02	ISO-15765-2:2016(E)

Default value: ISO-15765-2:2016(E) (PCANTP_ISO_REV_2016)

PCAN-Device: Any PCAN devices (excluding PCANTP_HANDLE_NONEBUS channel).

PCANTP_PARAMETER_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_HANDLE_NONEBUS channel).

PCANTP_PARAMETER_MSG_PENDING

Access: R W

Description: This parameter is used to define if the API should filter notifications of pending CANTP messages: fragmented CAN frames (either during reception and transmission) are notified by the API with an ISO-TP CANTP message where the message type in the ISO-TP content has a PCANTP_MESSAGE_INDICATION flag. If enabled (default), the function CANTP_Read_2016 (or Read_2016 method) will also return messages with this type. This feature is required by higher layer API like PCAN-UDS.

When receiving or transmitting an ISO-TP message, user will read 2 ISO-TP CANTP messages:

- The first one has the indication flag, stating a message is pending.
- The second does not, stating the message is complete.

Between the 2 messages, users can follow up communication progress with function CANTP_GetMsgProgress_2016.

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

	Type	Constant	Value	Description
	Byte	PCANTP_MSG_PENDING_HIDE	0x00	Messages with the type PCANTP_MESSAGE_INDICATION will be automatically removed from the result of the CANTP_Read_2016 function (or Read_2016 method).
	Byte	PCANTP_MSG_PENDING_SHOW	0x01	Messages with the type PCANTP_MESSAGE_INDICATION can be retrieved from the CANTP_Read_2016 function (or Read_2016 method).

Default value: PCANTP_MSG_PENDING_SHOW

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

PCANTP_PARAMETER_BLOCK_SIZE

Access: R W

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

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

Default value: 10.

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

PCANTP_PARAMETER_BLOCK_SIZE_TX

Access: R W

Description: Sets the transmit block size parameter (BS_TX). This parameter is passthru requirements that overrides and goes against the standard usage. It overrides BS (given by the ECU) when transmitting ISO-TP message.

Possible values: 0x00 (unlimited) to 0xFF, or PCANTP_BLOCK_SIZE_TX_IGNORE (0xFFFF) to ignore the parameter.

Default value: PCANTP_BLOCK_SIZE_TX_IGNORE (0xFFFF) which disables this non-standard feature.

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

PCANTP_PARAMETER_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 has 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_HANDLE_NONEBUS channel).

PCANTP_PARAMETER_SEPARATION_TIME_TX

Access: R W

Description: Sets the transmit seperation time parameter (STmin_TX). This parameter is passthru requirements that overrides and goes against the standard usage. It overrides overrides STmin (given by the ECU) when transmitting ISO-TP message.

Possible values: 0x00 to 0x7F (range from 0 to 127 ms) and 0xF1 to 0xF9 (range from 100 to 900 µs). Or PCANTP_SEPERATION_TIME_TX_IGNORE (0xFFFF) to ignore the parameter.

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: PCANTP_SEPERATION_TIME_TX_IGNORE (0xFFFF) which disables this non-standard feature.

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

PCANTP_PARAMETER_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.

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

Default value: PCANTP_WFT_MAX_DEFAULT (0x10).

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

Access: R W

Description: **ISO-15765-2 – "Timeout As".** It defines the maximum time to wait for a frame while transmitting (sender side timeout).

Possible values: 32 bits unsigned integer, time unit is microsecond. There are some predefined values:

	Type	Constant	Value	Description
	Int32	PCANTP_TIMEOUT_AS_ISO_15765_2	1000000	Default value for Timeout As in µs
	Int32	PCANTP_TIMEOUT_AS_ISO_15765_4	25000	OBDII value for Timeout As in µs

Default value: 1 000 000 µs (PCANTP_TIMEOUT_AS_ISO_15765_2).

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

PCANTP_PARAMETER_TIMEOUT_AR

Access: R W

Description: ISO-15765-2 – "Timeout Ar". It defines the maximum time to wait for a frame while receiving (receiver side timeout).

Possible values: 32 bits unsigned integer, time unit is microsecond. There are some predefined values:

	Type	Constant	Value	Description
	Int32	PCANTP_TIMEOUT_AR_ISO_15765_2	1000000	Default value for Timeout Ar in µs
	Int32	PCANTP_TIMEOUT_AR_ISO_15765_4	25000	OBDII value for Timeout Ar in µs

Default value: 1 000 000 µs (PCANTP_TIMEOUT_AR_ISO_15765_2).

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

PCANTP_PARAMETER_TIMEOUT_BS

Access: R W

Description: ISO-15765-2 - "Timeout BS". Sender side timeout while waiting for a flow control frame.

Possible values: 32 bits unsigned integer, time unit is microsecond. There are some predefined values:

	Type	Constant	Value	Description
	Int32	PCANTP_TIMEOUT_BS_ISO_15765_2	1000000	Default value for Timeout Bs in µs
	Int32	PCANTP_TIMEOUT_BS_ISO_15765_4	75000	OBDII value for Timeout Bs in µs

Default value: 1 000 000 µs (PCANTP_TIMEOUT_BS_ISO_15765_2).

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

PCANTP_PARAMETER_TIMEOUT_CR

Access: R W

Description: ISO-15765-2 - "Timeout CR". Receiver side timeout while waiting for a consecutive frame.

Possible values: 8 bits unsigned integer, time unit is microsecond. There are some predefined values:

	Type	Constant	Value	Description
	Int32	PCANTP_TIMEOUT_CR_ISO_15765_2	1000000	Default value for Timeout Cr in µs
	Int32	PCANTP_TIMEOUT_CR_ISO_15765_4	150000	OBDII value for Timeout Cr in µs

Default value: 1 000 000 µs (PCANTP_TIMEOUT_CR_ISO_15765_2**).**

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

PCANTP_PARAMETER_TIMEOUT_TOLERANCE

Access: R W

Description: The tolerence to apply to all timeout as a percentage.

Possible values: 32 bits unsigned integer 0% (0) to 100% (100).

Default value: 0% (PCANTP_TIMEOUT_TOLERANCE).

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

PCANTP_PARAMETER_ISO_TIMEOUTS

Access: R W

Description: Set predefined ISO values for timeouts.

Possible values:

	Type	Constant	Value	Description
	Byte	PCANTP_ISO_TIMEOUTS_15765_2	0	Sets timeouts according to ISO-15765-2
	Byte	PCANTP_ISO_TIMEOUTS_15765_4	1	Sets timeouts according to ISO-15765-4 (OBDII)

Default value: 0 (PCANTP_ISO_TIMEOUTS_15765_2).

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

PCANTP_PARAMETER_SELFRECEIVE_LATENCY

Access: R W

Description: Set optimization options to improve delay between ISO-TP consecutive frames. ECUs have different policies regarding the respect of Minimum Separation Time. When communicating with "strict" ECU, optimization should be disabled: API will ensure that the minimum time has passed before transmitting another frame, this results in slightly slower communication. On the opposite, an optimized setting will try to predict the duration between the order to transmit a frame and the time the frame is actually written on the CAN bus; since this is a prediction (which also depends on the bus load) it is most probable that the minimum separation time may not be respected (on average the difference with STmin value is less than 1 ms).

Possible values:

	Type	Constant	Value	Description
	Byte	PCANTP_SELFRECEIVE_LATENCY_NONE	0	No optimization (use this parameter if ECU requires strict respect of Minimum Separation Time).
	Byte	PCANTP_SELFRECEIVE_LATENCY_LIGHT	1	Fragmented self-receive frame mechanism is ignored when STmin is set to 0xF3 and lower (<300µs)
	Byte	PCANTP_SELFRECEIVE_LATENCY_OPTIMIZED	2	As LIGHT value plus optimize self-receive latency by predicting the time to effectively write frames on bus.

Default value: 1 (PCANTP_SELFRECEIVE_LATENCY_LIGHT).

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

PCANTP_PARAMETER_MAX_RX_QUEUE

Access: R W

Description: Define the maximum number of messages in the Rx queue.

Possible values: 0x0 to 0xFFFF.

Default value: 32767 (PCANTP_MAX_RX_QUEUE_DEFAULT).

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

PCANTP_PARAMETER_KEEP_HIGHER_LAYER_MESSAGES

Access: R W

Description: Define if messages handled by higher layer APIs are still available in this API. The feature is disabled by default. For instance, when using PCAN-UDS, any ISO-TP message that corresponds to a UDS request/response will be available in PCAN-UDS API but not in PCAN-ISO-TP API.

Possible values: 0 (false) or 1 (true).

Default value: 0 (false).

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

PCANTP_PARAMETER_FILTER_CAN_ID

Access: R W

Description: Defines if the white-list CAN IDs filtering mechanism is enabled. Enabling this feature can help speed up messages processing when connected to a CAN bus with moderate or heavy bus load. Consider enabling the feature, if ISO-TP communications fail with network timeout errors.

Possible values: 0 (false) or 1 (true).

Default value: 0 (false).

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

PCANTP_PARAMETER_SUPPORT_29B_ENHANCED

Access: R W

Description: Defines if the 29 bit Enhanced Diagnostic CAN identifier should be supported or not (see ISO-15765-3:2004). Since ISO-15765:2016, ISO-15765-3:2004 was moved to ISO-14229-3 but this specific addressing is no longer referenced. Previous 2004 prototypes CANTP_Initialize and CANTP_InitializeFD will enable this setting by default for backward compatibility reasons.

Possible values: 0 (false) or 1 (true).

Default value: 0 (false).

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

PCANTP_PARAMETER_SUPPORT_29B_FIXED_NORMAL

Access: R W

Description: Defines if the 29 bit Fixed Normal CAN identifier should be supported or not (see ISO-15765-2).

Possible values: 0 (false) or 1 (true).

Default value: 1 (true).

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

PCANTP_PARAMETER_SUPPORT_29B_MIXED

Access: R W

Description: Defines if the 29 bit Mixed CAN identifier should be supported or not (see ISO-15765-2).

Possible values: 0 (false) or 1 (true).

Default value: 1 (true).

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

PCANTP_PARAMETER_CHECK_MSG

Access: R

Description: Checks if the message is valid and can be sent (ex. if a mapping is needed) and corrects input if needed.

Possible values: pointer to a cantp_msg structure (**cantp_msg*).**

Default value: NA.

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

PCANTP_PARAMETER_RESET_HARD

Access: W

Description: Resets the CAN controller and clears internal reception and transmission queues. This parameter provides a way to uninitialize then initialize the CAN channel without losing any configured mappings and ISO-TP related settings.

Possible values: 1.

Default value: NA.

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

PCANTP_PARAMETER_NETWORK_LAYER_DESIGN

Access: R W

Description: Set the network layer design. Possible values are full duplex, which is the default design with no restriction, and half duplex which limits ISO-TP communications: only a single transmission/reception can happen at a time between two ISO-TP nodes.

Possible values: PCANTP_NETWORK_LAYER_FULL_DUPLEX (0),
PCANTP_NETWORK_LAYER_HALF_DUPLEX (1).

Default value: PCANTP_NETWORK_LAYER_FULL_DUPLEX (0).

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

PCANTP_PARAMETER_HARDWARE_NAME

Access: R

Description: This parameter is used to retrieve the name of the hardware represented by a PCAN channel. This is useful when an application wants to differentiate between several models of the same device, e.g. a PCAN-USB and a PCAN-USB Pro.

Possible values: The value is a null-terminated string which contains the name of the hardware specified by the given PCAN channel. The length of this text will have a maximum length of 32 bytes (null termination included).

Default value: N/A.

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

PCANTP_PARAMETER_DEVICE_ID

Access: R W

Description: This parameter is used on PCAN hardware to distinguish between 2 (or more) devices of the same type connected to the same computer. This value is persistent, i.e. the identifier will not be lost after disconnecting and connecting again a device.

Possible values: According with the firmware version, this value can be a number in the range [1..255] or [1..4294967295]. If the firmware has a resolution of one byte and the specified value is bigger, the value will be truncated.

Default value: If this parameter was never set before, the value is the maximum value possible for the used resolution. For 8-bits: 255 (0xFF), for 32 bits: 429496729 (0xFFFFFFFF).

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

Remarks:

- This parameter was originally called PCANTP_PARAMETER_DEVICE_NUMBER.

PCANTP_PARAMETER_CONTROLLER_NUMBER

Access: R

Description: This parameter is a zero-based index used to identify the CAN controllers built in a hardware. This parameter is useful when it is needed to communicate with a specific physical channel on a multichannel CAN Hardware, e.g. "0" or "1" on a PCAN-USB Pro device.

Possible values: A number in the range [0..n-1], where n is the number of physical channels on the device being used.

Default value: N/A.

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

PCANTP_PARAMETER_CHANNEL_FEATURES

Access: R

Description: This value is used to read the particularities of a PCAN Channel.

Possible values: The value can be one of the following values or a combination of them:

	Type	Constant	Value	Description
	UInt32	FEATURE_FD_CAPABLE	1	This value indicates that the hardware represented by a PCAN Channel is FD capable (it supports flexible data rate).
	UInt32	FEATURE_DELAY_CAPABLE	2	This value indicates that the hardware represented by a PCAN Channel allows the configuration of a delay between sending frames.
	Uint32	FEATURE_IO_CAPABLE	4	This value indicates that the hardware represented by a PCAN Channel supports I/O functionality for electronic circuits (USB-Chip devices)

Default value: A value of 0, indicating "no special features".

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

3.5.15 cantp_msgtype

Represents the type of a CANTP message (see field "type" in cantp_msg on page 28).

Syntax

C++

```
typedef enum _cantp_msgtype {
       PCANTP_MSGTYPE_NONE = 0,
       PCANTP_MSGTYPE_CAN = 1,
       PCANTP_MSGTYPE_CANFD = 2,
       PCANTP_MSGTYPE_ISOTP = 4,
       PCANTP_MSGTYPE_FRAME = PCANTP_MSGTYPE_CAN | PCANTP_MSGTYPE_CANFD,
       PCANTP_MSGTYPE_ANY = PCANTP_MSGTYPE_FRAME | PCANTP_MSGTYPE_ISOTP | 0xFFFFFFFF
} cantp_msgtype;
```

Pascal OO

```
cantp_msgtype = (
    PCANTP_MSGTYPE_NONE = 0,
    PCANTP_MSGTYPE_CAN = 1,
    PCANTP_MSGTYPE_CANFD = 2,
    PCANTP_MSGTYPE_ISOTP = 4,
    PCANTP_MSGTYPE_FRAME = UInt32(PCANTP_MSGTYPE_CAN) Or UInt32(PCANTP_MSGTYPE_CANFD),
    PCANTP_MSGTYPE_ANY = UInt32(PCANTP_MSGTYPE_FRAME) Or UInt32(PCANTP_MSGTYPE_ISOTP)
            Or UInt32($FFFFFFFF)
);
```

C#

```
[Flags]
public enum cantp_msgtype : UInt32
{
    PCANTP_MSGTYPE_NONE = 0,
    PCANTP_MSGTYPE_CAN = 1,
    PCANTP_MSGTYPE_CANFD = 2,
    PCANTP_MSGTYPE_ISOTP = 4,
    PCANTP_MSGTYPE_FRAME = PCANTP_MSGTYPE_CAN | PCANTP_MSGTYPE_CANFD,
    PCANTP_MSGTYPE_ANY = PCANTP_MSGTYPE_FRAME | PCANTP_MSGTYPE_ISOTP | 0xFFFFFFFF
}
```

C++ / CLR

```
public enum cantp_msgtype : UInt32
{
       PCANTP_MSGTYPE_NONE = 0,
       PCANTP_MSGTYPE_CAN = 1,
       PCANTP_MSGTYPE_CANFD = 2,
       PCANTP_MSGTYPE_ISOTP = 4,
       PCANTP_MSGTYPE_FRAME = PCANTP_MSGTYPE_CAN | PCANTP_MSGTYPE_CANFD,
       PCANTP_MSGTYPE_ANY = PCANTP_MSGTYPE_FRAME | PCANTP_MSGTYPE_ISOTP | 0xFFFFFFFF
};
```

Visual Basic

```
<Flags()>
Public Enum cantp_msgtype As UInt32
    PCANTP_MSGTYPE_NONE = 0
    PCANTP_MSGTYPE_CAN = 1
    PCANTP_MSGTYPE_CANFD = 2
    PCANTP_MSGTYPE_ISOTP = 4
    PCANTP_MSGTYPE_FRAME = PCANTP_MSGTYPE_CAN Or PCANTP_MSGTYPE_CANFD
    PCANTP_MSGTYPE_ANY = PCANTP_MSGTYPE_FRAME Or PCANTP_MSGTYPE_ISOTP Or &HFFFFFFFFUI
End Enum
```

Values

Name	Value	Description
PCANTP_MSGTYPE_NONE	0x00000000	Uninitialized message (data is NULL).
PCANTP_MSGTYPE_CAN	0x00000001	Standard CAN frame.
PCANTP_MSGTYPE_CANFD	0x00000002	CAN frame with FD support.
PCANTP_MSGTYPE_ISOTP	0x00000004	ISO-TP message (ISO:15765).
PCANTP_MSGTYPE_FRAME	0x00000003	Frame only: unsegmented messages.
PCANTP_MSGTYPE_ANY	0xFFFFFFFF	Any supported message type.

See also: cantp_msg on page 28.

3.5.16 cantp_msgflag

Represents the flags common to all types of cantp_msg (see field cantp_msg.msgdata.flags on page 23).

Syntax

C++

```
typedef enum _cantp_msgflag {
       PCANTP_MSGFLAG_NONE = 0,
       PCANTP_MSGFLAG_LOOPBACK = 1,
       PCANTP_MSGFLAG_ISOTP_FRAME = 2,
} cantp_msgflag;
```

Pascal OO

```
cantp_msgflag = (
    PCANTP_MSGFLAG_NONE = 0,
    PCANTP_MSGFLAG_LOOPBACK = 1,
    PCANTP_MSGFLAG_ISOTP_FRAME = 2
);
```

C#

```
public enum cantp_msgflag : UInt32
{
    PCANTP_MSGFLAG_NONE = 0,
    PCANTP_MSGFLAG_LOOPBACK = 1,
    PCANTP_MSGFLAG_ISOTP_FRAME = 2,
}
```

C++ / CLR

```
public enum cantp_msgflag : UInt32
{
       PCANTP_MSGFLAG_NONE = 0,
       PCANTP_MSGFLAG_LOOPBACK = 1,
       PCANTP_MSGFLAG_ISOTP_FRAME = 2,
};
```

Visual Basic

```
Public Enum cantp_msgflag As UInt32
    PCANTP_MSGFLAG_NONE = 0
    PCANTP_MSGFLAG_LOOPBACK = 1
    PCANTP_MSGFLAG_ISOTP_FRAME = 2
End Enum
```

Values

Name	Value	Description
PCANTP_MSGFLAG_NONE	0	No flag.
PCANTP_MSGFLAG_LOOPBACK	1	Message is the confirmation of a transmitted message.
PCANTP_MSGFLAG_ISOTP_FRAME	2	Message is a frame of a segmented ISO-TP message.

See also: cantp_msg on page 28, cantp_msgdata on page 22

3.5.17 cantp_can_msgtype

Represents the flags of a CAN or CAN FD message (must be used as flags for ex. EXTENDED|FD|BRS.) (see field cantp_msg.can_info.can_msgtype on page 18).

Syntax

C++

```
#define PCAN_MESSAGE_STANDARD        0x00U
#define PCAN_MESSAGE_RTR             0x01U
#define PCAN_MESSAGE_EXTENDED        0x02U
#define PCAN_MESSAGE_FD              0x04U
#define PCAN_MESSAGE_BRS             0x08U
#define PCAN_MESSAGE_ESI             0x10U
#define PCAN_MESSAGE_ERRFRAME        0x40U
#define PCAN_MESSAGE_STATUS          0x80U
typedef enum _cantp_can_msgtype {
       PCANTP_CAN_MSGTYPE_STANDARD = PCAN_MESSAGE_STANDARD,
       PCANTP_CAN_MSGTYPE_RTR = PCAN_MESSAGE_RTR,
       PCANTP_CAN_MSGTYPE_EXTENDED = PCAN_MESSAGE_EXTENDED,
       PCANTP_CAN_MSGTYPE_FD = PCAN_MESSAGE_FD,
       PCANTP_CAN_MSGTYPE_BRS = PCAN_MESSAGE_BRS,
       PCANTP_CAN_MSGTYPE_ESI = PCAN_MESSAGE_ESI,
       PCANTP_CAN_MSGTYPE_ERRFRAME = PCAN_MESSAGE_ERRFRAME,
       PCANTP_CAN_MSGTYPE_STATUS = PCAN_MESSAGE_STATUS,
       PCANTP_CAN_MSGTYPE_ECHO = PCAN_MESSAGE_ECHO,
       PCANTP_CAN_MSGTYPE_FLAG_INFO = (PCAN_MESSAGE_ERRFRAME | PCAN_MESSAGE_STATUS)
} cantp_can_msgtype;
```

Pascal OO

```
cantp_can_msgtype = (
    PCANTP_CAN_MSGTYPE_STANDARD = UInt32(PCAN_MESSAGE_STANDARD),
    PCANTP_CAN_MSGTYPE_RTR = UInt32(PCAN_MESSAGE_RTR),
    PCANTP_CAN_MSGTYPE_EXTENDED = UInt32(PCAN_MESSAGE_EXTENDED),
    PCANTP_CAN_MSGTYPE_FD = UInt32(PCAN_MESSAGE_FD),
    PCANTP_CAN_MSGTYPE_BRS = UInt32(PCAN_MESSAGE_BRS),
    PCANTP_CAN_MSGTYPE_ESI = UInt32(PCAN_MESSAGE_ESI),
    PCANTP_CAN_MSGTYPE_ERRFRAME = UInt32(PCAN_MESSAGE_ERRFRAME),
    PCANTP_CAN_MSGTYPE_STATUS = UInt32(PCAN_MESSAGE_STATUS),
    PCANTP_CAN_MSGTYPE_ECHO = UInt32(PCAN_MESSAGE_ECHO),
    PCANTP_CAN_MSGTYPE_FLAG_INFO = (UInt32(PCAN_MESSAGE_ERRFRAME)
            Or UInt32(PCAN_MESSAGE_STATUS))
);
```

C#

```
public enum cantp_can_msgtype : UInt32
{
    PCANTP_CAN_MSGTYPE_STANDARD = TPCANMessageType.PCAN_MESSAGE_STANDARD,
    PCANTP_CAN_MSGTYPE_RTR = TPCANMessageType.PCAN_MESSAGE_RTR,
```

```
    PCANTP_CAN_MSGTYPE_EXTENDED = TPCANMessageType.PCAN_MESSAGE_EXTENDED,
    PCANTP_CAN_MSGTYPE_FD = TPCANMessageType.PCAN_MESSAGE_FD,
    PCANTP_CAN_MSGTYPE_BRS = TPCANMessageType.PCAN_MESSAGE_BRS,
    PCANTP_CAN_MSGTYPE_ESI = TPCANMessageType.PCAN_MESSAGE_ESI,
    PCANTP_CAN_MSGTYPE_ERRFRAME = TPCANMessageType.PCAN_MESSAGE_ERRFRAME,
    PCANTP_CAN_MSGTYPE_STATUS = TPCANMessageType.PCAN_MESSAGE_STATUS,
    PCANTP_CAN_MSGTYPE_ECHO = TPCANMessageType.PCAN_MESSAGE_ECHO,
    PCANTP_CAN_MSGTYPE_FLAG_INFO = (TPCANMessageType.PCAN_MESSAGE_ERRFRAME
            | TPCANMessageType.PCAN_MESSAGE_STATUS)
}
```

C++ / CLR

```
public enum cantp_can_msgtype : UInt32
{
        PCANTP_CAN_MSGTYPE_STANDARD = (UInt32)TPCANMessageType::PCAN_MESSAGE_STANDARD,
        PCANTP_CAN_MSGTYPE_RTR = (UInt32)TPCANMessageType::PCAN_MESSAGE_RTR,
        PCANTP_CAN_MSGTYPE_EXTENDED = (UInt32)TPCANMessageType::PCAN_MESSAGE_EXTENDED,
        PCANTP_CAN_MSGTYPE_FD = (UInt32)TPCANMessageType::PCAN_MESSAGE_FD,
        PCANTP_CAN_MSGTYPE_BRS = (UInt32)TPCANMessageType::PCAN_MESSAGE_BRS,
        PCANTP_CAN_MSGTYPE_ESI = (UInt32)TPCANMessageType::PCAN_MESSAGE_ESI,
        PCANTP_CAN_MSGTYPE_ERRFRAME = (UInt32)TPCANMessageType::PCAN_MESSAGE_ERRFRAME,
        PCANTP_CAN_MSGTYPE_STATUS = (UInt32)TPCANMessageType::PCAN_MESSAGE_STATUS,
        PCANTP_CAN_MSGTYPE_ECHO = (UInt32)TPCANMessageType::PCAN_MESSAGE_ECHO,
        PCANTP_CAN_MSGTYPE_FLAG_INFO = ((UInt32)TPCANMessageType::PCAN_MESSAGE_ERRFRAME
              | (UInt32)TPCANMessageType::PCAN_MESSAGE_STATUS)
};
```

Visual Basic

```
Public Enum cantp_can_msgtype As UInt32
    PCANTP_CAN_MSGTYPE_STANDARD = TPCANMessageType.PCAN_MESSAGE_STANDARD
    PCANTP_CAN_MSGTYPE_RTR = TPCANMessageType.PCAN_MESSAGE_RTR
    PCANTP_CAN_MSGTYPE_EXTENDED = TPCANMessageType.PCAN_MESSAGE_EXTENDED
    PCANTP_CAN_MSGTYPE_FD = TPCANMessageType.PCAN_MESSAGE_FD
    PCANTP_CAN_MSGTYPE_BRS = TPCANMessageType.PCAN_MESSAGE_BRS
    PCANTP_CAN_MSGTYPE_ESI = TPCANMessageType.PCAN_MESSAGE_ESI
    PCANTP_CAN_MSGTYPE_ERRFRAME = TPCANMessageType.PCAN_MESSAGE_ERRFRAME
    PCANTP_CAN_MSGTYPE_STATUS = TPCANMessageType.PCAN_MESSAGE_STATUS
    PCANTP_CAN_MSGTYPE_ECHO = TPCANMessageType.PCAN_MESSAGE_ECHO
    PCANTP_CAN_MSGTYPE_FLAG_INFO = (TPCANMessageType.PCAN_MESSAGE_ERRFRAME
            Or TPCANMessageType.PCAN_MESSAGE_STATUS)
End Enum
```

Values

Name	Value	Description
PCANTP_CAN_MSGTYPE_STANDARD	0x00	The PCAN message is a CAN Standard Frame (11-bit identifier).
PCANTP_CAN_MSGTYPE_RTR	0x01	The PCAN message is a CAN Remote-Transfer-Request Frame.
PCANTP_CAN_MSGTYPE_EXTENDED	0x02	The PCAN message is a CAN Extended Frame (29-bit identifier).
PCANTP_CAN_MSGTYPE_FD	0x04	The PCAN message represents a FD frame in terms of CiA Specs.
PCANTP_CAN_MSGTYPE_BRS	0x08	The PCAN message represents a FD bit rate switch (CAN data at a higher bit rate).
PCANTP_CAN_MSGTYPE_ESI	0x10	The PCAN message represents a FD error state indicator (CAN FD transmitter was error active).
PCANTP_CAN_MSGTYPE_ERRFRAME	0x40	The PCAN message represents an error frame.
PCANTP_CAN_MSGTYPE_STATUS	0x80	The PCAN message represents a PCAN status message.
PCANTP_CAN_MSGTYPE_ECHO	0x20	The PCAN message represents a self-received (Tx-loopback) message.
PCANTP_CAN_MSGTYPE_FLAG_INFO	0xC0	Mask filtering error frame messages and status messages.

Note: PCANTP_CAN_MSGTYPE_ECHO replaced PCANTP_CAN_MSGTYPE_SELFRECEIVE, which is now set as deprecated.

See also: cantp_msg on page 28, cantp_can_info on page 18.

3.5.18 cantp_isotp_msgtype

Represents the addressing format of an ISO-TP message (see field cantp_msg.msgdata.isotp.netaddrinfo.msgtype on page 19).

Syntax

C++

```
typedef enum _cantp_isotp_msgtype {
       PCANTP_ISOTP_MSGTYPE_UNKNOWN = 0x00,
       PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC = 0x01,
       PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC = 0x02,
       PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX = 0x10,
       PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX = 0x20,
       PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION = (0x10 | 0x20),
       PCANTP_ISOTP_MSGTYPE_MASK_INDICATION = 0x0F
} cantp_isotp_msgtype;
```

Pascal OO

```
cantp_isotp_msgtype = (
    PCANTP_ISOTP_MSGTYPE_UNKNOWN = $00,
    PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC = $01,
    PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC = $02,
    PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX = $10,
    PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX = $20,
    PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION = ($10 Or $20),
    PCANTP_ISOTP_MSGTYPE_MASK_INDICATION = $0F
);
```

C#

```
public enum cantp_isotp_msgtype : UInt32
{
    PCANTP_ISOTP_MSGTYPE_UNKNOWN = 0x00,
    PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC = 0x01,
    PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC = 0x02,
    PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX = 0x10,
    PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX = 0x20,
    PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION = (0x10 | 0x20),
    PCANTP_ISOTP_MSGTYPE_MASK_INDICATION = 0x0F
}
```

C++ / CLR

```
public enum cantp_isotp_msgtype : UInt32
{
       PCANTP_ISOTP_MSGTYPE_UNKNOWN = 0x00,
       PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC = 0x01,
       PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC = 0x02,
       PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX = 0x10,
       PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX = 0x20,
       PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION = (0x10 | 0x20),
       PCANTP_ISOTP_MSGTYPE_MASK_INDICATION = 0x0F
```

```
};
```

Visual Basic

```
Public Enum cantp_isotp_msgtype As UInt32
        PCANTP_ISOTP_MSGTYPE_UNKNOWN = &H0
        PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC = &H1
        PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC = &H2
        PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX = &H10
        PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX = &H20
        PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION = (&H10 Or &H20)
        PCANTP_ISOTP_MSGTYPE_MASK_INDICATION = &HF
End Enum
```

Values

Name	Value	Description
PCANTP_ISOTP_MSGTYPE_UNKNOWN	0x00	Unknown (non-ISO-TP) message.
PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC	0x01	Diagnostic message (request or confirmation).
PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC	0x02	Remote Diagnostic message (request or confirmation).
PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX	0x10	Multi-Frame Message is being received.
PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX	0x20	Multi-Frame Message is being transmitted.
PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION	0x30	Multi-Frame Message is being communicated (Tx or Rx).
PCANTP_ISOTP_MSGTYPE_MASK_INDICATION	0x0F	Mask to remove Indication flags.

See also: cantp_msg on page 28, cantp_msgdata_isotp on page 26, cantp_netaddrinfo on page 19

3.5.19 cantp_isotp_format

Represents the addressing format of an ISO-TP message.

See also: cantp_msg.msgdata.isotp.netaddrinfo.format on page 19

Syntax

C++

```
typedef enum _cantp_isotp_format {
       PCANTP_ISOTP_FORMAT_UNKNOWN = 0xFF,
       PCANTP_ISOTP_FORMAT_NONE = 0x00,
       PCANTP_ISOTP_FORMAT_NORMAL = 0x01,
       PCANTP_ISOTP_FORMAT_FIXED_NORMAL = 0x02,
       PCANTP_ISOTP_FORMAT_EXTENDED = 0x03,
       PCANTP_ISOTP_FORMAT_MIXED = 0x04,
       PCANTP_ISOTP_FORMAT_ENHANCED = 0x05,
} cantp_isotp_format;
```

Pascal OO

```
cantp_isotp_format = (
    PCANTP_ISOTP_FORMAT_UNKNOWN = $FF,
    PCANTP_ISOTP_FORMAT_NONE = $00,
    PCANTP_ISOTP_FORMAT_NORMAL = $01,
    PCANTP_ISOTP_FORMAT_FIXED_NORMAL = $02,
    PCANTP_ISOTP_FORMAT_EXTENDED = $03,
    PCANTP_ISOTP_FORMAT_MIXED = $04,
    PCANTP_ISOTP_FORMAT_ENHANCED = $05
);
```

C#

```
public enum cantp_isotp_format : UInt32
{
    PCANTP_ISOTP_FORMAT_UNKNOWN = 0xFF,
    PCANTP_ISOTP_FORMAT_NONE = 0x00,
    PCANTP_ISOTP_FORMAT_NORMAL = 0x01,
    PCANTP_ISOTP_FORMAT_FIXED_NORMAL = 0x02,
    PCANTP_ISOTP_FORMAT_EXTENDED = 0x03,
    PCANTP_ISOTP_FORMAT_MIXED = 0x04,
    PCANTP_ISOTP_FORMAT_ENHANCED = 0x05,
}
```

C++ / CLR

```
public enum cantp_isotp_format : UInt32
{
       PCANTP_ISOTP_FORMAT_UNKNOWN = 0xFF,
       PCANTP_ISOTP_FORMAT_NONE = 0x00,
       PCANTP_ISOTP_FORMAT_NORMAL = 0x01,
       PCANTP_ISOTP_FORMAT_FIXED_NORMAL = 0x02,
       PCANTP_ISOTP_FORMAT_EXTENDED = 0x03,
       PCANTP_ISOTP_FORMAT_MIXED = 0x04,
       PCANTP_ISOTP_FORMAT_ENHANCED = 0x05,
};
```

Visual Basic

```
Public Enum cantp_isotp_format As UInt32
    PCANTP_ISOTP_FORMAT_UNKNOWN = &HFF
    PCANTP_ISOTP_FORMAT_NONE = &H0
    PCANTP_ISOTP_FORMAT_NORMAL = &H1
    PCANTP_ISOTP_FORMAT_FIXED_NORMAL = &H2
    PCANTP_ISOTP_FORMAT_EXTENDED = &H3
    PCANTP_ISOTP_FORMAT_MIXED = &H4
    PCANTP_ISOTP_FORMAT_ENHANCED = &H5
End Enum
```

Values

Name	Value	Description
PCANTP_ISOTP_FORMAT_UNKNOWN	0xFF	Unknown adressing format.
PCANTP_ISOTP_FORMAT_NONE	0x00	Unsegmented CAN frame.
PCANTP_ISOTP_FORMAT_NORMAL	0x01	Normal adressing format from ISO 15765-2.
PCANTP_ISOTP_FORMAT_FIXED_NORMAL	0x02	Fixed normal adressing format from ISO 15765-2.
PCANTP_ISOTP_FORMAT_EXTENDED	0x03	Extended adressing format from ISO 15765-2.
PCANTP_ISOTP_FORMAT_MIXED	0x04	Mixed adressing format from ISO 15765-2.
PCANTP_ISOTP_FORMAT_ENHANCED	0x05	Enhanced adressing format from ISO 15765-3.

See also: cantp_msg on page 28, cantp_msgdata_isotp on page 26, cantp_netaddrinfo on page 19.

3.5.20 cantp_isotp_addressing

Represents the type of target of an ISO-TP message.

See also: cantp_msg.msgdata.isotp.netaddrinfo.target_type on page 19

Syntax

C++

```
typedef enum _cantp_isotp_addressing {
       PCANTP_ISOTP_ADDRESSING_UNKNOWN = 0x00,
       PCANTP_ISOTP_ADDRESSING_PHYSICAL = 0x01,
       PCANTP_ISOTP_ADDRESSING_FUNCTIONAL = 0x02,
} cantp_isotp_addressing;
```

Pascal OO

```
cantp_isotp_addressing = (
    PCANTP_ISOTP_ADDRESSING_UNKNOWN = $00,
    PCANTP_ISOTP_ADDRESSING_PHYSICAL = $01,
    PCANTP_ISOTP_ADDRESSING_FUNCTIONAL = $02
);
```

C#

```
public enum cantp_isotp_addressing : UInt32
{
    PCANTP_ISOTP_ADDRESSING_UNKNOWN = 0x00,
    PCANTP_ISOTP_ADDRESSING_PHYSICAL = 0x01,
    PCANTP_ISOTP_ADDRESSING_FUNCTIONAL = 0x02,
}
```

C++ / CLR

```
public enum cantp_isotp_addressing : UInt32
{
       PCANTP_ISOTP_ADDRESSING_UNKNOWN = 0x00,
       PCANTP_ISOTP_ADDRESSING_PHYSICAL = 0x01,
       PCANTP_ISOTP_ADDRESSING_FUNCTIONAL = 0x02,
};
```

Visual Basic

```
Public Enum cantp_isotp_addressing As UInt32
    PCANTP_ISOTP_ADDRESSING_UNKNOWN = &H0
    PCANTP_ISOTP_ADDRESSING_PHYSICAL = &H1
    PCANTP_ISOTP_ADDRESSING_FUNCTIONAL = &H2
End Enum
```

Values

Name	Value	Description
PCANTP_ISOTP_ADDRESSING_UNKNOWN	0x00	Unknown adressing format.
PCANTP_ISOTP_ADDRESSING_PHYSICAL	0x01	Physical addressing ("peer to peer").
PCANTP_ISOTP_ADDRESSING_FUNCTIONAL	0x02	Functional addressing ("peer to any").

See also: cantp_msg on page 28, cantp_msgdata_isotp on page 26, cantp_netaddrinfo on page 19

3.5.21 cantp_option

Represents the options of a message (mainly supported for ISO-TP message) (see field cantp_msg.msgdata.options on page 22).

Syntax

C++

```
typedef enum _cantp_option {
        PCANTP_OPTION_FRAME_FILTERING = PCANTP_PARAMETER_FRAME_FILTERING,
        PCANTP_OPTION_CAN_DATA_PADDING = PCANTP_PARAMETER_CAN_DATA_PADDING,
        PCANTP_OPTION_CAN_PADDING_VALUE = PCANTP_PARAMETER_CAN_PADDING_VALUE,
        PCANTP_OPTION_J1939_PRIORITY = PCANTP_PARAMETER_J1939_PRIORITY,
        PCANTP_OPTION_MSG_PENDING = PCANTP_PARAMETER_MSG_PENDING,
        PCANTP_OPTION_BLOCK_SIZE = PCANTP_PARAMETER_BLOCK_SIZE,
        PCANTP_OPTION_BLOCK_SIZE_TX = PCANTP_PARAMETER_BLOCK_SIZE_TX,
        PCANTP_OPTION_SEPARATION_TIME = PCANTP_PARAMETER_SEPARATION_TIME,
        PCANTP_OPTION_SEPARATION_TIME_TX = PCANTP_PARAMETER_SEPARATION_TIME_TX,
        PCANTP_OPTION_WFT_MAX = PCANTP_PARAMETER_WFT_MAX,
        PCANTP_OPTION_SELFRECEIVE_LATENCY = PCANTP_PARAMETER_SELFRECEIVE_LATENCY
} cantp_option;
```

Pascal OO

```
cantp_option = (
    PCANTP_OPTION_FRAME_FILTERING = UInt32(PCANTP_PARAMETER_FRAME_FILTERING),
    PCANTP_OPTION_CAN_DATA_PADDING = UInt32(PCANTP_PARAMETER_CAN_DATA_PADDING),
    PCANTP_OPTION_CAN_PADDING_VALUE = UInt32(PCANTP_PARAMETER_CAN_PADDING_VALUE),
    PCANTP_OPTION_J1939_PRIORITY = UInt32(PCANTP_PARAMETER_J1939_PRIORITY),
    PCANTP_OPTION_MSG_PENDING = UInt32(PCANTP_PARAMETER_MSG_PENDING),
    PCANTP_OPTION_BLOCK_SIZE = UInt32(PCANTP_PARAMETER_BLOCK_SIZE),
    PCANTP_OPTION_BLOCK_SIZE_TX = UInt32(PCANTP_PARAMETER_BLOCK_SIZE_TX),
    PCANTP_OPTION_SEPARATION_TIME = UInt32(PCANTP_PARAMETER_SEPARATION_TIME),
    PCANTP_OPTION_SEPARATION_TIME_TX = UInt32(PCANTP_PARAMETER_SEPARATION_TIME_TX),
    PCANTP_OPTION_WFT_MAX = UInt32(PCANTP_PARAMETER_WFT_MAX),
    PCANTP_OPTION_SELFRECEIVE_LATENCY = UInt32(PCANTP_PARAMETER_SELFRECEIVE_LATENCY)
);
```

C#

```
public enum cantp_option : UInt32
{
    PCANTP_OPTION_FRAME_FILTERING = cantp_parameter.PCANTP_PARAMETER_FRAME_FILTERING,
    PCANTP_OPTION_CAN_DATA_PADDING = cantp_parameter.PCANTP_PARAMETER_CAN_DATA_PADDING,
    PCANTP_OPTION_CAN_PADDING_VALUE = cantp_parameter.PCANTP_PARAMETER_CAN_PADDING_VALUE,
    PCANTP_OPTION_J1939_PRIORITY = cantp_parameter.PCANTP_PARAMETER_J1939_PRIORITY,
    PCANTP_OPTION_MSG_PENDING = cantp_parameter.PCANTP_PARAMETER_MSG_PENDING,
    PCANTP_OPTION_BLOCK_SIZE = cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE,
    PCANTP_OPTION_BLOCK_SIZE_TX = cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE_TX,
    PCANTP_OPTION_SEPARATION_TIME = cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME,
    PCANTP_OPTION_SEPARATION_TIME_TX = cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME_TX,
    PCANTP_OPTION_WFT_MAX = cantp_parameter.PCANTP_PARAMETER_WFT_MAX,
    PCANTP_OPTION_SELFRECEIVE_LATENCY = cantp_parameter.PCANTP_PARAMETER_SELFRECEIVE_LATENCY
}
```

C++ / CLR

```
public enum cantp_option : UInt32
{
      PCANTP_OPTION_FRAME_FILTERING = cantp_parameter::PCANTP_PARAMETER_FRAME_FILTERING,
      PCANTP_OPTION_CAN_DATA_PADDING = cantp_parameter::PCANTP_PARAMETER_CAN_DATA_PADDING,
      PCANTP_OPTION_CAN_PADDING_VALUE = cantp_parameter::PCANTP_PARAMETER_CAN_PADDING_VALUE,
      PCANTP_OPTION_J1939_PRIORITY = cantp_parameter::PCANTP_PARAMETER_J1939_PRIORITY,
      PCANTP_OPTION_MSG_PENDING = cantp_parameter::PCANTP_PARAMETER_MSG_PENDING,
      PCANTP_OPTION_BLOCK_SIZE = cantp_parameter::PCANTP_PARAMETER_BLOCK_SIZE,
      PCANTP_OPTION_BLOCK_SIZE_TX = cantp_parameter::PCANTP_PARAMETER_BLOCK_SIZE_TX,
      PCANTP_OPTION_SEPARATION_TIME = cantp_parameter::PCANTP_PARAMETER_SEPARATION_TIME,
```

```
        PCANTP_OPTION_SEPARATION_TIME_TX = cantp_parameter::PCANTP_PARAMETER_SEPARATION_TIME_TX,
        PCANTP_OPTION_WFT_MAX = cantp_parameter::PCANTP_PARAMETER_WFT_MAX,
        PCANTP_OPTION_SELFRECEIVE_LATENCY = cantp_parameter::PCANTP_PARAMETER_SELFRECEIVE_LATENCY
};
```

Visual Basic

```
Public Enum cantp_option As UInt32
    PCANTP_OPTION_FRAME_FILTERING = cantp_parameter.PCANTP_PARAMETER_FRAME_FILTERING
    PCANTP_OPTION_CAN_DATA_PADDING = cantp_parameter.PCANTP_PARAMETER_CAN_DATA_PADDING
    PCANTP_OPTION_CAN_PADDING_VALUE = cantp_parameter.PCANTP_PARAMETER_CAN_PADDING_VALUE
    PCANTP_OPTION_J1939_PRIORITY = cantp_parameter.PCANTP_PARAMETER_J1939_PRIORITY
    PCANTP_OPTION_MSG_PENDING = cantp_parameter.PCANTP_PARAMETER_MSG_PENDING
    PCANTP_OPTION_BLOCK_SIZE = cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE
    PCANTP_OPTION_BLOCK_SIZE_TX = cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE_TX
    PCANTP_OPTION_SEPARATION_TIME = cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME
    PCANTP_OPTION_SEPARATION_TIME_TX = cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME_TX
    PCANTP_OPTION_WFT_MAX = cantp_parameter.PCANTP_PARAMETER_WFT_MAX
    PCANTP_OPTION_SELFRECEIVE_LATENCY = cantp_parameter.PCANTP_PARAMETER_SELFRECEIVE_LATENCY
End Enum
```

Values

Name	Value	Data type	Description
PCANTP_OPTION_FRAME_FILTERING	0x105	Byte	Defines if unsegmented (NON-ISO-TP) CAN frames can be received.
PCANTP_OPTION_CAN_DATA_PADDING	0x107	Byte	Defines if CAN frame DLC uses padding or not.
PCANTP_OPTION_CAN_PADDING_VALUE	0x108	Byte	Defines the value used for CAN data padding.
PCANTP_OPTION_J1939_PRIORITY	0x10A	Byte	Defines the default priority value for normal fixed, mixed and enhanced addressing.
PCANTP_OPTION_MSG_PENDING	0x10B	Byte	Defines if pending messages are displayed/hidden.
PCANTP_OPTION_BLOCK_SIZE	0x10C	Byte	Defines the block size parameter (BS).
PCANTP_OPTION_BLOCK_SIZE_TX	0x10D	Int16	Defines the transmit block size parameter (BS_TX).
PCANTP_OPTION_SEPARATION_TIME	0x10E	Byte	Defines the seperation time parameter (STmin).
PCANTP_OPTION_SEPARATION_TIME_TX	0x10F	Int16	Defines the transmit seperation time parameter (STmin_TX).
PCANTP_OPTION_WFT_MAX	0x110	Int32	Defines the Wait Frame Transmissions parameter.
PCANTP_OPTION_SELFRECEIVE_LATENCY	0x117	Byte	Sets optimization options to improve delay between ISO-TP consecutive frames.

See cantp_parameter on page 70 for a description of the options.

See also: cantp_msgoption on page 16, CANTP_MsgDataInitOptions_2016 on page 254, **class method version:** MsgDataInitOptions_2016 on page 194, getOption_2016 on page 216, setOption_2016 on page 218

3.5.22 cantp_msgprogress_state

Represents the status for a message whose transmission is in progress.

Syntax

C++

```
typedef enum _cantp_msgprogress_state {
        PCANTP_MSGPROGRESS_STATE_QUEUED = 0,
        PCANTP_MSGPROGRESS_STATE_PROCESSING = 1,
        PCANTP_MSGPROGRESS_STATE_COMPLETED = 2,
        PCANTP_MSGPROGRESS_STATE_UNKNOWN = 3
} cantp_msgprogress_state;
```

Pascal OO

```
cantp_msgprogress_state = (
    PCANTP_MSGPROGRESS_STATE_QUEUED = 0,
    PCANTP_MSGPROGRESS_STATE_PROCESSING = 1,
    PCANTP_MSGPROGRESS_STATE_COMPLETED = 2,
    PCANTP_MSGPROGRESS_STATE_UNKNOWN = 3
);
```

C#

```
public enum cantp_msgprogress_state : UInt32
{
    PCANTP_MSGPROGRESS_STATE_QUEUED = 0,
    PCANTP_MSGPROGRESS_STATE_PROCESSING = 1,
    PCANTP_MSGPROGRESS_STATE_COMPLETED = 2,
    PCANTP_MSGPROGRESS_STATE_UNKNOWN = 3
}
```

C++ / CLR

```
public enum cantp_msgprogress_state : UInt32
{
       PCANTP_MSGPROGRESS_STATE_QUEUED = 0,
       PCANTP_MSGPROGRESS_STATE_PROCESSING = 1,
       PCANTP_MSGPROGRESS_STATE_COMPLETED = 2,
       PCANTP_MSGPROGRESS_STATE_UNKNOWN = 3
};
```

Visual Basic

```
Public Enum cantp_msgprogress_state As UInt32
        PCANTP_MSGPROGRESS_STATE_QUEUED = 0
        PCANTP_MSGPROGRESS_STATE_PROCESSING = 1
        PCANTP_MSGPROGRESS_STATE_COMPLETED = 2
        PCANTP_MSGPROGRESS_STATE_UNKNOWN = 3
End Enum
```

Values

Name	Value	Description
PCANTP_MSGPROGRESS_STATE_QUEUED	0	Message is not yet handled.
PCANTP_MSGPROGRESS_STATE_PROCESSING	1	Message is being processed (received or transmitted).
PCANTP_MSGPROGRESS_STATE_COMPLETED	2	Message is completed.
PCANTP_MSGPROGRESS_STATE_UNKNOWN	3	Message is unknown/not found.

See also: cantp_msgprogress on page 31, CANTP_GetMsgProgress_2016 on page 238, GetMsgProgress_2016 on page 143

3.5.23 cantp_msgdirection

Represents the direction of a message's communication.

Syntax

C++

```
typedef enum _cantp_msgdirection {
       PCANTP_MSGDIRECTION_RX = 0,
       PCANTP_MSGDIRECTION_TX = 1,
```

```
} cantp_msgdirection;
```

Pascal OO

```
cantp_msgdirection = (
    PCANTP_MSGDIRECTION_RX = 0,
    PCANTP_MSGDIRECTION_TX = 1
);
```

C#

```
public enum cantp_msgdirection : UInt32
{
        PCANTP_MSGDIRECTION_RX = 0,
        PCANTP_MSGDIRECTION_TX = 1,
}
```

C++ / CLR

```
public enum cantp_msgdirection : UInt32
{
       PCANTP_MSGDIRECTION_RX = 0,
       PCANTP_MSGDIRECTION_TX = 1,
};
```

Visual Basic

```
Public Enum cantp_msgdirection As UInt32
        PCANTP_MSGDIRECTION_RX = 0
        PCANTP_MSGDIRECTION_TX = 1
End Enum
```

Values

Name	Value	Description
PCANTP_MSGDIRECTION_RX	0	Message is being received.
PCANTP_MSGDIRECTION_TX	1	Message is being transmitted.

See also: CANTP_GetMsgProgress_2016 on page 238, **class method version:** GetMsgProgress_2016 on page 143

3.6 Methods

The methods defined for the classes CanTpApi (on page 14) and TCanTpApi (on page 15) 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

	Methods	Description
S	Initialize_2016	Initializes a PCANTP channel.
S	InitializeFD_2016	Initializes a PCANTP channel with CAN-FD support.
S	Uninitialize_2016	Uninitializes a PCANTP channel.

Configuration

	Methods	Description
S	SetValue_2016	Sets a configuration or information value within a PCANTP channel.
S	AddMapping_2016	Configures the ISO-TP mapping between a CAN ID and an ISO-TP network addressing information.
S	RemoveMapping_2016	Removes a previously configured ISO-TP mapping.
S	RemoveMappings_2016	Removes previously configured ISO-TP mappings.
S	AddFiltering_2016	Adds an entry to the CAN-ID white-list filtering.
S	RemoveFiltering_2016	Removes an entry from the CAN-ID white-list filtering.

Information

	Methods	Description
S	GetValue_2016	Retrieves information from a PCANTP channel.
S	StatusGet_2016	Retrieves the value of a cantp_status subtype.
S	GetErrorText_2016	Gets a descriptive text for an error code.
S	GetCanBusStatus_2016	Gets information about the internal BUS status of a PCANTP Channel.
S	GetMsgProgress_2016	Gets progress information on a specific message.
S	StatusListTypes_2016	Lists the subtypes contained in the PCANTP status.
S	GetMappings_2016	Retrieves all the mappings defined for a PCANTP channel.
S	StatusIsOk_2016	Check if the status is an error status or not.

Communication

	Methods	Description
S	Read_2016	Reads a CAN message from the receive queue of a PCANTP channel.
S	Write_2016	Transmits a CAN message using a connected PCANTP channel.
S	Reset_2016	Resets the receive and transmit queues of a PCANTP channel.

Messages handling

	Methods	Description
S	MsgEqual_2016	Checks if two CANTP messages are equal.
S	MsgCopy_2016	Copies a CANTP message to another buffer.
S	MsgDlcToLength_2016	Converts a CAN DLC to its corresponding length.
S	MsgLengthToDlc_2016	Converts a data length to a corresponding CAN DLC.
S	MsgDataAlloc_2016	Allocates a CANTP message based on the given type.

	Methods	Description
	MsgDataInit_2016	Initializes an allocated CANTP message.
	MsgDataInitOptions_2016	Initializes several options for the CANTP message that will override the channel's parameter(s).
	MsgDataFree_2016	Deallocates a CANTP message.

Helper (C# and VB specifics methods)

	Methods	Description
	allocProgressBuffer_2016	In a progress structure, allocate a buffer to receive a copy of the pending message.
	freeProgressBuffer_2016	Free the buffer receiving the pending message in a progress object.
	getProgressBuffer_2016	Get the current pending message of a progress structure.
	getFlags_2016	Get the flags of a message.
	setLength_2016	Set the length of a message.
	getLength_2016	Get the length of a message in a safe way.
	setData_2016	Set the data of a message.
	getData_2016	Get the data of a message.
	getNetStatus_2016	Get the netstatus of a message in a safe way.
	getOption_2016	Get an option of a message.
	setOption_2016	Modifies an option of a message.
	getOptionsNumber_2016	Get the number of options of a message.
	setNetaddrinfo_2016	Set the network address information of an ISO-TP message.
	getNetaddrinfo_2016	Get the network address information of an ISO-TP message.

3.6.1 Initialize_2016

Initializes a PCANTP channel based on a PCANTP handle (without CAN-FD support).

Overloads

	Method	Description
	Initialize_2016(cantp_handle, cantp_baudrate)	Initializes a Plug and Play PCANTP channel.
	Initialize_2016(cantp_handle, cantp_baudrate, cantp_hwtype, UInt32, UInt16)	Initializes a Non-Plug-and Play PCANTP channel.

Plain function version: CANTP_Initialize_2016 on page 225

3.6.2 Initialize_2016(cantp_handle, cantp_baudrate)

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

Syntax

Pascal OO

```
class function Initialize_2016(
  channel: cantp_handle;
  baudrate: cantp_baudrate
): cantp_status; overload;
```

C#

```
public static cantp_status Initialize_2016(
        cantp_handle channel,
        cantp_baudrate baudrate);
```

C++ / CLR

```
static cantp_status Initialize_2016(
       cantp_handle channel,
       cantp_baudrate baudrate);
```

Visual Basic

```
Public Shared Function Initialize_2016(
    ByVal channel As cantp_handle,
    ByVal baudrate As cantp_baudrate) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
baudrate	The speed for the communication (see cantp_baudrate on page 46).

Returns

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

PCANTP_STATUS_ALREADY_INITIALIZED	Indicates that the desired PCANTP channel is already in use.
PCANTP_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error.

Remarks

The Initialize_2016 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_HANDLE_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 SetValue_2016 on page 110).

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

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

Example

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

C#

```
cantp_status result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
        cantp_baudrate.PCANTP_BAUDRATE_500K);
```

```
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Initialization failed", "Error");
else
        MessageBox.Show("PCAN-PCI (Ch-2) was initialized", "Success");

// All initialized channels are released
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS);
```

C++/CLR

```
cantp_status result;
// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi_2016::Initialize_2016(PCANTP_HANDLE_PCIBUS2, PCANTP_BAUDRATE_500K);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Initialization failed", "Error");
else
       MessageBox::Show("PCAN-PCI (Ch-2) was initialized", "Success");

// All initialized channels are released
CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_NONEBUS);
```

Visual Basic

```
Dim result As cantp_status

' The Plug And Play channel (PCAN-PCI) Is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    cantp_baudrate.PCANTP_BAUDRATE_500K)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Initialization failed", "Error")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized", "Success")
End If

' All initialized channels are released
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS)
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Plug and Play channel (PCAN-PCI) is initialized.
  result := TCanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'PCAN-PCI (Ch-2) was initialized', 'Success', MB_OK);
  end;

  // All initialized channels are released
  TCanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS);
end;
```

Plain function version: CANTP_Initialize_2016 on page 225

See also: InitializeFD_2016 on page 105, Uninitialize_2016 on page 107, Understanding PCAN-ISO-TP on page 7

3.6.3 Initialize_2016(cantp_handle, cantp_baudrate, cantp_hwtype, UInt32, UInt16)

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

Syntax

Pascal OO

```
class function Initialize_2016(
    channel: cantp_handle;
    baudrate: cantp_baudrate;
    hw_type: cantp_hwtype;
    io_port: UInt32;
    interrupt: UInt16
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Initialize_2016")]
public static extern cantp_status Initialize_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_baudrate baudrate,
    [MarshalAs(UnmanagedType.U4)]
    cantp_hwtype hw_type,
    UInt32 io_port,
    UInt16 interrupt);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Initialize_2016")]
static cantp_status Initialize_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_baudrate baudrate,
       [MarshalAs(UnmanagedType::U4)]
       cantp_hwtype hw_type,
       UInt32 io_port,
       UInt16 interrupt);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Initialize_2016")>
Public Shared Function Initialize_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal baudrate As cantp_baudrate,
    <MarshalAs(UnmanagedType.U4)>
    ByVal hw_type As cantp_hwtype,
    ByVal io_port As UInt32,
    ByVal interrupt As UInt16) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
baudrate	The speed for the communication (see cantp_baudrate on page 46)
hw_type	Non plug-n-play: the type of hardware (see cantp_hwtype on page 48)
io_port	Non plug-n-play: the I/O address for the parallel port.
interrupt	Non plug-n-play: interrupt number of the parallel port.

Returns

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

PCANTP_STATUS_ALREADY_INITIALIZED	Indicates that the desired PCANTP channel is already in use.
PCANTP_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error.

Remarks

The Initialize_2016 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_HANDLE_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 SetValue_2016 on page 110).

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

Example

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

C#

```
cantp_status result;

// The Non-Plug and Play channel (PCAN-DNG) is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_DNGBUS1,
        cantp_baudrate.PCANTP_BAUDRATE_500K, cantp_hwtype.PCANTP_HWTYPE_DNG_SJA, 0x378, 7);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Initialization failed", "Error");
else
        MessageBox.Show("PCAN-DNG (Ch-1) was initialized", "Success");

// All initialized channels are released.
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS);
```

C++/CLR

```
cantp_status result;
```

```
// The Non-Plug and Play channel (PCAN-DNG) is initialized.
result = CanTpApi_2016::Initialize_2016(PCANTP_HANDLE_DNGBUS1, PCANTP_BAUDRATE_500K,
       PCANTP_HWTYPE_DNG_SJA, 0x378, 7);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Initialization failed", "Error");
else
       MessageBox::Show("PCAN-DNG (Ch-1) was initialized", "Success");

// All initialized channels are released.
CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_NONEBUS);
```

Visual Basic

```
Dim result As cantp_status

' The Non-Plug And Play channel (PCAN-DNG) Is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_DNGBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K, cantp_hwtype.PCANTP_HWTYPE_DNG_SJA, &H378, 7)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Initialization failed", "Error")
Else
    MessageBox.Show("PCAN-DNG (Ch-1) was initialized", "Success")
End If

' All initialized channels are released.
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS)
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Non-Plug and Play channel (PCAN-DNG) is initialized.
  result := TCanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_DNGBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K,
    cantp_hwtype.PCANTP_HWTYPE_DNG_SJA, $378, 7);
  if NOT(TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)) then
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'PCAN-DNG (Ch-1) was initialized', 'Success', MB_OK);
  end;

  // All initialized channels are released.
  TCanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS);
end;
```

Plain function version: CANTP_Initialize_2016 on page 225

See also: InitializeFD_2016 on page 105, Uninitialize_2016 on page 107, Understanding PCAN-ISO-TP on page 7

3.6.4 InitializeFD_2016

Initializes a PCANTP channel based on a CANTP handle (including CAN-FD support).

Syntax

Pascal OO

```
class function InitializeFD_2016(
    channel: cantp_handle;
    const bitrate_fd: cantp_bitrate
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_InitializeFD_2016")]
public static extern cantp_status InitializeFD_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.LPStr)]
    cantp_bitrate bitrate_fd);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_InitializeFD_2016")]
static cantp_status InitializeFD_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::LPStr)]
       cantp_bitrate bitrate_fd);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_InitializeFD_2016")>
Public Shared Function InitializeFD_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByVal bitrate_fd As cantp_bitrate) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a FD capable PCAN Channel (see cantp_handle on page 37)
bitrate_fd	The speed for the communication (see FD Bit Rate Parameter Definitions on page 35, see cantp_bitrate on page 34)

Returns

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

PCANTP_STATUS_ALREADY_INITIALIZED	Indicates that the desired PCANTP channel is already in use.
PCANTP_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error.

Remarks

The InitializeFD_2016 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_HANDLE_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 SetValue_2016 on page 110).

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-USB hardware).

C#

```
cantp_status result;

// The Plug and Play channel (PCAN-USB) is initialized @500kbps/2Mbps.
result = CanTpApi.InitializeFD_2016(cantp_handle.PCANTP_HANDLE_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 (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Initialization failed", "Error");
else
        MessageBox.Show("PCAN-USB (Ch-2) was initialized", "Success");

// All initialized channels are released.
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS);
```

C++/CLR

```
cantp_status result;

// The Plug and Play channel (PCAN-USB) is initialized @500kbps/2Mbps.
result = CanTpApi_2016::InitializeFD_2016(PCANTP_HANDLE_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 (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Initialization failed", "Error");
else
       MessageBox::Show("PCAN-USB (Ch-2) was initialized", "Success");

// All initialized channels are released.
CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_NONEBUS);
```

Visual Basic

```
Dim result As cantp_status

' The Plug And Play channel (PCAN-USB) Is initialized @500kbps/2Mbps.
result = CanTpApi.InitializeFD_2016(cantp_handle.PCANTP_HANDLE_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 Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Initialization failed", "Error")
```

```
Else
    MessageBox.Show("PCAN-USB (Ch-2) was initialized", "Success")
End If

' All initialized channels are released.
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS)
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Plug and Play channel (PCAN-USB) is initialized @500kbps/2Mbps.
  result := TCanTpApi.InitializeFD_2016(cantp_handle.PCANTP_HANDLE_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 NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'PCAN-USB (Ch-2) was initialized', 'Success', MB_OK);
  end;

  // All initialized channels are released.
  TCanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_NONEBUS);
end;
```

Plain function version: CANTP_InitializeFD_2016 on page 226

See Also: Uninitialize_2016 below, Initialize_2016 on page 99, Understanding PCAN-ISO-TP on page 7, FD Bit Rate Parameter Definitions on page 35

3.6.5 Uninitialize_2016

Uninitializes an already initialized PCANTP channel.

Syntax

Pascal OO

```
class function Uninitialize_2016(
    channel: cantp_handle
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Uninitialize_2016")]
public static extern cantp_status Uninitialize_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Uninitialize_2016")]
static cantp_status Uninitialize_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Uninitialize_2016")>
Public Shared Function Uninitialize_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)

Returns

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

PCANTP_STATUS_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 PCANTP 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 PCANTP_HANDLE_NONEBUS which instructs the API to search for all channels initialized by the calling application and release them all. This option causes 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#

```
cantp_status result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
        cantp_baudrate.PCANTP_BAUDRATE_500K);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Initialization failed", "Error");
else
        MessageBox.Show("PCAN-PCI (Ch-2) was initialized", "Success");

// Uninitialize PCI channel 2
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Uninitialization failed", "Error");
else
        MessageBox.Show("PCAN-PCI (Ch-2) was uninitialized", "Success");
```

C++/CLR

```
cantp_status result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CanTpApi_2016::Initialize_2016(PCANTP_HANDLE_PCIBUS2, PCANTP_BAUDRATE_500K);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
```

```
        MessageBox::Show("Initialization failed", "Error");
else
        MessageBox::Show("PCAN-PCI (Ch-2) was initialized", "Success");

// Uninitialize PCI channel 2
CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_PCIBUS2);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
        MessageBox::Show("Uninitialization failed", "Error");
else
        MessageBox::Show("PCAN-PCI (Ch-2) was uninitialized", "Success");
```

Visual Basic

```
Dim result As cantp_status

' The Plug And Play channel (PCAN-PCI) Is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    cantp_baudrate.PCANTP_BAUDRATE_500K)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Initialization failed", "Error")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initialized", "Success")
End If

' Uninitialize PCI channel 2
CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Uninitialization failed", "Error")
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was uninitialized", "Success")
End If
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Plug and Play channel (PCAN-PCI) is initialized.
  result := TCanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'PCAN-PCI (Ch-2) was initialized', 'Success', MB_OK);
  end;

  // Uninitialize PCI channel 2
  TCanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Uninitialization failed', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'PCAN-PCI (Ch-2) was uninitialized', 'Success', MB_OK);
  end;
end;
```

Plain function version: CANTP_Uninitialize_2016 on page 227

See also: Initialize_2016 on page 99, InitializeFD_2016 on page 105

3.6.6 SetValue_2016

Sets a configuration or information value within a PCANTP channel.

Overloads

	Method	Description
	SetValue_2016(cantp_handle, cantp_parameter, UInt32, UInt32)	Sets a configuration or information numeric value within a PCANTP channel.
	SetValue_2016(cantp_handle, cantp_parameter, Byte[], UInt32)	Sets a configuration or information with an array of bytes within a PCANTP channel.
	SetValue_2016(cantp_handle, cantp_parameter, String, UInt32)	Sets a configuration or information string value within a PCANTP channel.

Plain function version: CANTP_SetValue_2016 on page 228

3.6.7 SetValue_2016(cantp_handle, cantp_parameter, UInt32, UInt32)

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

Syntax

Pascal OO

```
class function SetValue_2016(
    channel: cantp_handle;
    parameter: cantp_parameter;
    numericBuffer: PLongWord;
    buffer_length: UInt32
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue_2016")]
public static extern cantp_status SetValue_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_parameter parameter,
    ref UInt32 NumericBuffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue_2016")]
static cantp_status SetValue_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_parameter parameter,
       UInt32 %NumericBuffer,
       UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_SetValue_2016")>
Public Shared Function SetValue_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal parameter As cantp_parameter,
    ByRef NumericBuffer As UInt32,
    ByVal BufferLength As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
parameter	The code of the value to be set (see cantp_parameter on page 70)
NumericBuffer	The buffer containing the numeric value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

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

PCANTP_STAUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the method are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length.

Remarks

Use the method SetValue_2016 to set configuration information or environment values of a PCANTP channel.

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

More information about the parameters and values can be found in Detailed parameters values on page 75. 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_2016 on the channel PCANTP_HANDLE_PCIBUS2 to enable debug mode.

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

C#

```
cantp_status result;
UInt32 iBuffer = 0;

// Enables CAN DEBUG mode.
iBuffer = CanTpApi.PCANTP_DEBUG_CAN;
result = CanTpApi.SetValue_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
        cantp_parameter.PCANTP_PARAMETER_DEBUG, ref iBuffer, sizeof(UInt32));
if (result != cantp_status.PCANTP_STATUS_OK)
        MessageBox.Show("Failed to set value");
else
        MessageBox.Show("Value changed successfully");
```

C++/CLR

```
cantp_status result;
UInt32 iBuffer = 0;

// Enables CAN DEBUG mode.
iBuffer = CanTpApi_2016::PCANTP_DEBUG_CAN;
result = CanTpApi_2016::SetValue_2016(PCANTP_HANDLE_PCIBUS2, PCANTP_PARAMETER_DEBUG, iBuffer,
       sizeof(UInt32));
if (result != PCANTP_STATUS_OK)
       MessageBox::Show("Failed to set value");
else
       MessageBox::Show("Value changed successfully");
```

Visual Basic

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

' Enables CAN DEBUG mode.
iBuffer = CanTpApi.PCANTP_DEBUG_CAN
result = CanTpApi.SetValue_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    cantp_parameter.PCANTP_PARAMETER_DEBUG, iBuffer, Convert.ToUInt32(Len(iBuffer)))
If result <> cantp_status.PCANTP_STATUS_OK Then
    MessageBox.Show("Failed to set value")
Else
    MessageBox.Show("Value changed successfully")
End If
```

Pascal OO

```
var
  result: cantp_status;
  iBuffer: UInt32;
begin
  iBuffer := 0;

  // Enables CAN DEBUG mode.
  iBuffer := TCanTpApi.PCANTP_DEBUG_CAN;
  result := TCanTpApi.SetValue_2016(cantp_handle.PCANTP_HANDLE_PCIBUS2,
    cantp_parameter.PCANTP_PARAMETER_DEBUG, PLongWord(@iBuffer),
    sizeof(UInt32));
  if result <> cantp_status.PCANTP_STATUS_OK then
  begin
    MessageBox(0, 'Failed to set value', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'Value changed successfully', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_SetValue_2016 on page 228

See also: GetValue_2016 on page 131, cantp_parameter on page 70, Detailed parameters values on page 75

3.6.8 SetValue_2016(cantp_handle, cantp_parameter, String, UInt32)

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

Syntax

Pascal OO

```
class function SetValue_2016(
    channel: cantp_handle;
    parameter: cantp_parameter;
    StringBuffer: PAnsiChar;
    buffer_length: UInt32
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue_2016")]
public static extern cantp_status SetValue_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_parameter parameter,
    [MarshalAs(UnmanagedType.LPStr, SizeParamIndex = 3)]
    string StringBuffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue_2016")]
static cantp_status SetValue_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_parameter parameter,
       [MarshalAs(UnmanagedType::LPStr, SizeParamIndex = 3)]
       String ^StringBuffer,
       UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_SetValue_2016")>
Public Shared Function SetValue_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal parameter As cantp_parameter,
    <MarshalAs(UnmanagedType.LPStr, SizeParamIndex:=3)>
    ByVal StringBuffer As String,
    ByVal BufferLength As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
parameter	The code of the value to be set (see cantp_parameter on page 70).
StringBuffer	The buffer containing the value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

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

PCANTP_STAUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the method are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length.

Remarks

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

Plain function version: CANTP_SetValue_2016 on page 228

See also: GetValue_2016 on page 131, cantp_parameter on page 70, Detailed parameters values on page 75

3.6.9 SetValue_2016(cantp_handle, cantp_parameter, Byte[], UInt32)

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

Syntax

Pascal OO

```
class function SetValue_2016(
    channel: cantp_handle;
    parameter: cantp_parameter;
    byteBuffer: PByte;
    buffer_length: UInt32
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue_2016")]
public static extern cantp_status SetValue_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_parameter parameter,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
    Byte[] Buffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_SetValue_2016")]
static cantp_status SetValue_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_parameter parameter,
       [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 3)]
       array<Byte>^ Buffer,
       UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_SetValue_2016")>
Public Shared Function SetValue_2016(
```

```
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal parameter As cantp_parameter,
    ByVal Buffer As Byte(),
    ByVal BufferLength As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
parameter	The code of the value to be set (see cantp_parameter on page 70)
Buffer	The buffer with the value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

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

PCANTP_STAUS_NOT_INITIALIZED	Indicates that the given PCANTP channel it was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the method are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length.

Remarks

Use the method SetValue_2016 to set configuration information or environment values of a PCANTP channel.

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

More information about the parameters and values can be found in Detailed parameters values on page 75. 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_2016 on the channel PCANTP_HANDLE_USBBUS1 to define the use of unlimited block size during ISO-TP communication.

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

C#

```
cantp_status result;

// Defines unlimited blocksize.
result = CanTpApi.SetValue_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
        cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE, new byte[] { 0 }, 1);
if (result != cantp_status.PCANTP_STATUS_OK)
        MessageBox.Show("Failed to set value");
else
        MessageBox.Show("Value changed successfully");
```

C++/CLR

```
cantp_status result;
```

```
// Defines unlimited blocksize.
result = CanTpApi_2016::SetValue_2016(PCANTP_HANDLE_USBBUS1, PCANTP_PARAMETER_BLOCK_SIZE,
       gcnew array<Byte> { 0 }, 1);
if (result != PCANTP_STATUS_OK)
       MessageBox::Show("Failed to set value");
else
       MessageBox::Show("Value changed successfully");
```

Visual Basic

```
Dim result As cantp_status
Dim buffer_array(2) As Byte

' Defines unlimited blocksize.
buffer_array(0) = 0

result = CanTpApi.SetValue_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE, buffer_array, 1)
If result <> cantp_status.PCANTP_STATUS_OK Then
    MessageBox.Show("Failed to set value")
Else
    MessageBox.Show("Value changed successfully")
End If
```

Pascal OO

```
var
  result: cantp_status;
  bufferArray: array [0 .. 1] of Byte;
begin

  // Defines unlimited blocksize.
  result := TCanTpApi.SetValue_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_parameter.PCANTP_PARAMETER_BLOCK_SIZE, PLongWord(@bufferArray), 1);
  if result <> cantp_status.PCANTP_STATUS_OK then
  begin
    MessageBox(0, 'Failed to set value', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'Value changed successfully', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_SetValue_2016 on page 228

See also: GetValue_2016 on page 131, cantp_parameter on page 70, Detailed parameters values on page 75

3.6.10 AddMapping_2016

Adds a user-defined PCANTP mapping between CAN ID and ISOTP Network Address Information within a PCANTP channel.

Syntax

Pascal OO

```
class function AddMapping_2016(
    channel: cantp_handle;
    mapping: Pcantp_mapping
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_AddMapping_2016")]
public static extern cantp_status AddMapping_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    ref cantp_mapping mapping);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_AddMapping_2016")]
       static cantp_status AddMapping_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       cantp_mapping %mapping);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_AddMapping_2016")>
Public Shared Function AddMapping_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByRef mapping As cantp_mapping) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
mapping	Mapping to be added. (see cantp_mapping on page 20)

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_ALREADY_INITIALIZED	A mapping with the same CAN ID already exists.
PCANTP_STATUS_PARAM_INVALID_VALUE	Mapping is not valid regarding ISO-TP standard.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory to define mapping.

Remarks

The cantp_mapping structure is described on page 20.

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#

```
cantp_handle channel = cantp_handle.PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = new cantp_mapping();
cantp_mapping response_mapping;

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
```

```
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype =
       cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type =
       cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Defines a second mapping to allow communication from Destination 0x13 to Source 0xF1.
response_mapping = request_mapping;
response_mapping.can_id = request_mapping.can_id_flow_ctrl;
response_mapping.can_id_flow_ctrl = request_mapping.can_id;
response_mapping.netaddrinfo.source_addr = request_mapping.netaddrinfo.target_addr;
response_mapping.netaddrinfo.target_addr = request_mapping.netaddrinfo.source_addr;

// Add request mapping
result = CanTpApi.AddMapping_2016(channel, ref request_mapping);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Failed to add request mapping.", "Error");
// Add response mapping
result = CanTpApi.AddMapping_2016(channel, ref response_mapping);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Failed to add response mapping.", "Error");
```

C++ / CLR

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = {};
cantp_mapping response_mapping;

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype = PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type = PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Defines a second mapping to allow communication from Destination 0x13 to Source 0xF1.
response_mapping = request_mapping;
response_mapping.can_id = request_mapping.can_id_flow_ctrl;
response_mapping.can_id_flow_ctrl = request_mapping.can_id;
response_mapping.netaddrinfo.source_addr = request_mapping.netaddrinfo.target_addr;
response_mapping.netaddrinfo.target_addr = request_mapping.netaddrinfo.source_addr;

// Add request mapping
result = CanTpApi_2016::AddMapping_2016(channel, request_mapping);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Failed to add request mapping.", "Error");
// Add response mapping
result = CanTpApi_2016::AddMapping_2016(channel, response_mapping);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Failed to add response mapping.", "Error");
```

Visual Basic

```
Dim channel As cantp_handle = cantp_handle.PCANTP_HANDLE_USBBUS1
Dim result As cantp_status
Dim request_mapping As cantp_mapping
Dim response_mapping As cantp_mapping

' Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = &HD1
request_mapping.can_id_flow_ctrl = &HD2
request_mapping.netaddrinfo.source_addr = &HF1
request_mapping.netaddrinfo.target_addr = &H13
request_mapping.netaddrinfo.extension_addr = &H52
request_mapping.can_msgtype = cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD
request_mapping.netaddrinfo.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED
request_mapping.netaddrinfo.msgtype =
    cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC
request_mapping.netaddrinfo.target_type =
    cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL

' Defines a second mapping to allow communication from Destination 0x13 to Source 0xF1.
response_mapping = request_mapping
response_mapping.can_id = request_mapping.can_id_flow_ctrl
response_mapping.can_id_flow_ctrl = request_mapping.can_id
response_mapping.netaddrinfo.source_addr = request_mapping.netaddrinfo.target_addr
response_mapping.netaddrinfo.target_addr = request_mapping.netaddrinfo.source_addr

' Add request mapping
result = CanTpApi.AddMapping_2016(channel, request_mapping)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Failed to add request mapping.", "Error")
End If

' Add response mapping
result = CanTpApi.AddMapping_2016(channel, response_mapping)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Failed to add response mapping.", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  channel: cantp_handle;
  request_mapping: cantp_mapping;
  response_mapping: cantp_mapping;
begin
  channel := cantp_handle.PCANTP_HANDLE_USBBUS1;

  // Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
  request_mapping.can_id := $D1;
  request_mapping.can_id_flow_ctrl := $D2;
  request_mapping.netaddrinfo.source_addr := $F1;
  request_mapping.netaddrinfo.target_addr := $13;
  request_mapping.netaddrinfo.extension_addr := $52;
  request_mapping.can_msgtype := cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD;
  request_mapping.netaddrinfo.format :=
    cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED;
  request_mapping.netaddrinfo.msgtype :=
    cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
  request_mapping.netaddrinfo.target_type :=
    cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;
```

```
  // Defines a second mapping to allow communication from Destination 0x13 to Source 0xF1.
  response_mapping := request_mapping;
  response_mapping.can_id := request_mapping.can_id_flow_ctrl;
  response_mapping.can_id_flow_ctrl := request_mapping.can_id;
  response_mapping.netaddrinfo.source_addr :=
    request_mapping.netaddrinfo.target_addr;
  response_mapping.netaddrinfo.target_addr :=
    request_mapping.netaddrinfo.source_addr;

  // Add request mapping
  result := TCanTpApi.AddMapping_2016(channel, @request_mapping);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Failed to add request mapping.', 'Error', MB_OK);
  end;
  // Add response mapping
  result := TCanTpApi.AddMapping_2016(channel, @response_mapping);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Failed to add response mapping.', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_AddMapping_2016 on page 229

See also: cantp_mapping on page 20, RemoveMapping_2016 below, RemoveMappings_2016 on page 123

3.6.11 RemoveMapping_2016

Removes a user-defined PCANTP mapping on a channel using a unique mapping identifier.

Syntax

Pascal OO

```
class function RemoveMapping_2016(
    channel: cantp_handle;
    uid: Pointer
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveMapping_2016")]
public static extern cantp_status RemoveMapping_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    UIntPtr uid);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveMapping_2016")]
static cantp_status RemoveMapping_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       UIntPtr uid);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_RemoveMapping_2016")>
Public Shared Function RemoveMapping_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByVal uid As UIntPtr) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
uid	Unique identifier of the mapping to remove.

Returns

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

PCANTP_STATUS_MAPPING_NOT_INITIALIZED	The PCANTP mapping to remove is not in the mapping list.

Example

The following example shows the use of the method RemoveMapping_2016 on the PCANTP channel USB 1. It adds a mapping and removes it using unique identifier.

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

C#

```
cantp_handle channel = cantp_handle.PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = new cantp_mapping();

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype =
       cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type =
       cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Add request mapping
result = CanTpApi.AddMapping_2016(channel, ref request_mapping);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
       MessageBox.Show("Failed to add request mapping.", "Error");

// Remove request mapping
result = CanTpApi.RemoveMapping_2016(channel, request_mapping.uid);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
       MessageBox.Show("Failed to remove mapping.", "Error");
```

C++ / CLR

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = {};

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype = PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type = PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Add request mapping
result = CanTpApi_2016::AddMapping_2016(channel, request_mapping);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Failed to add request mapping.", "Error");

// Remove request mapping
result = CanTpApi_2016::RemoveMapping_2016(channel, request_mapping.uid);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Failed to remove mapping.", "Error");
```

Visual Basic

```
Dim channel As cantp_handle
channel = cantp_handle.PCANTP_HANDLE_USBBUS1
Dim result As cantp_status
Dim request_mapping As cantp_mapping

' Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = &HD1
request_mapping.can_id_flow_ctrl = &HD2
request_mapping.netaddrinfo.source_addr = &HF1
request_mapping.netaddrinfo.target_addr = &H13
request_mapping.netaddrinfo.extension_addr = &H52
request_mapping.can_msgtype = cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD
request_mapping.netaddrinfo.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED
request_mapping.netaddrinfo.msgtype =
    cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC
request_mapping.netaddrinfo.target_type =
    cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL

' Add request mapping
result = CanTpApi.AddMapping_2016(channel, request_mapping)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Failed to add request mapping.", "Error")
End If

' Remove request mapping
result = CanTpApi.RemoveMapping_2016(channel, request_mapping.uid)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Failed to remove mapping.", "Error")
End If
```

Pascal OO

```
var
```

```
  result: cantp_status;
  channel: cantp_handle;
  request_mapping: cantp_mapping;
begin
  channel := cantp_handle.PCANTP_HANDLE_USBBUS1;

  // Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
  request_mapping.can_id := $D1;
  request_mapping.can_id_flow_ctrl := $D2;
  request_mapping.netaddrinfo.source_addr := $F1;
  request_mapping.netaddrinfo.target_addr := $13;
  request_mapping.netaddrinfo.extension_addr := $52;
  request_mapping.can_msgtype := cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD;
  request_mapping.netaddrinfo.format :=
    cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED;
  request_mapping.netaddrinfo.msgtype :=
    cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
  request_mapping.netaddrinfo.target_type :=
    cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;

  // Add request mapping
  result := TCanTpApi.AddMapping_2016(channel, @request_mapping);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Failed to add request mapping.', 'Error', MB_OK);
  end;

  // Remove request mapping
  result := TCanTpApi.RemoveMapping_2016(channel, request_mapping.uid);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Failed to remove mapping.', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_RemoveMapping_2016 on page 231

See also: cantp_mapping on page 20, RemoveMappings_2016 below, AddMapping_2016 on page 116

3.6.12 RemoveMappings_2016

Removes all user-defined PCANTP mappings corresponding to a CAN ID.

Syntax

Pascal OO

```
class function RemoveMappings_2016(
    channel: cantp_handle;
    can_id: UInt32
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveMappings_2016")]
public static extern cantp_status RemoveMappings_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    UInt32 can_id);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveMappings_2016")]
       static cantp_status RemoveMappings_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       UInt32 can_id);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_RemoveMappings_2016")>
Public Shared Function RemoveMappings_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByVal can_id As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
can_id	The mapped CAN Identifier to search for that identifies the mapping to remove.

Returns

The return value is a cantp_status code. PCANTP_ STATUS_OK is returned on success. The typical error in case of failure is:

PCANTP_STATUS_MAPPING_NOT_INITIALIZED	The PCANTP CANID to remove is not specified in a mapping.

Example

The following example shows the definition and removal of a mapping using 0xD1 CANID.

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

C#

```
cantp_handle channel = cantp_handle.PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = new cantp_mapping();

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype =
       cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type =
       cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Add request mapping
result = CanTpApi.AddMapping_2016(channel, ref request_mapping);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
       MessageBox.Show("Failed to add request mapping.", "Error");

// Remove request mapping using CANID
```

```
result = CanTpApi.RemoveMappings_2016(channel, request_mapping.can_id);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
       MessageBox.Show("Failed to remove mapping.", "Error");
```

C++ / CLR

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = {};

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype = PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type = PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Add request mapping
result = CanTpApi_2016::AddMapping_2016(channel, request_mapping);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Failed to add request mapping.", "Error");

// Remove request mapping using CANID
result = CanTpApi_2016::RemoveMappings_2016(channel, request_mapping.can_id);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Failed to remove mapping.", "Error");
```

Visual Basic

```
Dim channel As cantp_handle
channel = cantp_handle.PCANTP_HANDLE_USBBUS1
Dim result As cantp_status
Dim request_mapping As cantp_mapping

' Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = &HD1
request_mapping.can_id_flow_ctrl = &HD2
request_mapping.netaddrinfo.source_addr = &HF1
request_mapping.netaddrinfo.target_addr = &H13
request_mapping.netaddrinfo.extension_addr = &H52
request_mapping.can_msgtype = cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD
request_mapping.netaddrinfo.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED
request_mapping.netaddrinfo.msgtype =
    cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC
request_mapping.netaddrinfo.target_type =
    cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL

' Add request mapping
result = CanTpApi.AddMapping_2016(channel, request_mapping)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Failed to add request mapping.", "Error")
End If

' Remove request mapping using CANID
result = CanTpApi.RemoveMappings_2016(channel, request_mapping.can_id)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Failed to remove mapping.", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  channel: cantp_handle;
  request_mapping: cantp_mapping;
begin
  channel := cantp_handle.PCANTP_HANDLE_USBBUS1;

  // Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
  request_mapping.can_id := $D1;
  request_mapping.can_id_flow_ctrl := $D2;
  request_mapping.netaddrinfo.source_addr := $F1;
  request_mapping.netaddrinfo.target_addr := $13;
  request_mapping.netaddrinfo.extension_addr := $52;
  request_mapping.can_msgtype := cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD;
  request_mapping.netaddrinfo.format :=
    cantp_isotp_format.PCANTP_ISOTP_FORMAT_MIXED;
  request_mapping.netaddrinfo.msgtype :=
    cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
  request_mapping.netaddrinfo.target_type :=
    cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;

  // Add request mapping
  result := TCanTpApi.AddMapping_2016(channel, @request_mapping);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Failed to add request mapping.', 'Error', MB_OK);
  end;

  // Remove request mapping using CANID
  result := TCanTpApi.RemoveMappings_2016(channel, request_mapping.can_id);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Failed to remove mapping.', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_RemoveMappings_2016 on page 232

See also: cantp_mapping on page 20, RemoveMapping_2016 on page 120, AddMapping_2016 on page 116

3.6.13 AddFiltering_2016

Adds an entry to the CAN-ID white-list filtering.

Syntax

Pascal OO

```
class function AddFiltering_2016(
    channel: cantp_handle;
    can_id_from: UInt32;
    can_id_to: UInt32;
    ignore_can_msgtype: Boolean;
    can_msgtype: cantp_can_msgtype
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_AddFiltering_2016")]
```

```
public static extern cantp_status AddFiltering_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    UInt32 can_id_from,
    UInt32 can_id_to,
    [MarshalAs(UnmanagedType.U1)]
    bool ignore_can_msgtype,
    [MarshalAs(UnmanagedType.U4)]
    cantp_can_msgtype can_msgtype);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_AddFiltering_2016")]
static cantp_status AddFiltering_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       UInt32 can_id_from,
       UInt32 can_id_to,
       [MarshalAs(UnmanagedType::U1)]
       bool ignore_can_msgtype,
       [MarshalAs(UnmanagedType::U4)]
       cantp_can_msgtype can_msgtype);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_AddFiltering_2016")>
Public Shared Function AddFiltering_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByVal can_id_from As UInt32,
    ByVal can_id_to As UInt32,
    <MarshalAs(UnmanagedType.U1)>
    ByVal ignore_can_msgtype As Boolean,
    <MarshalAs(UnmanagedType.U4)>
    ByVal can_msgtype As cantp_can_msgtype) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
can_id_from	The lowest CAN ID wanted to be received.
can_id_to	The highest CAN ID wanted to be received.
ignore_can_msgtype	States if filter should check the CAN message type.
can_msgtype	If ignore_can_msgtype is false, the value states which types of CAN frame should be allowed (see cantp_can_msgtype on page 88).

Returns

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

PCANTP_STATUS_NO_MEMORY	Memory allocation error when add element in the white list.

Example

The following example shows the use of the method AddFiltering_2016 on the channel PCANTP_HANDLE_USBBUS1. It adds a filter from 0xD1 can identifier to 0xD2 can identifier for standard messages.

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

C#

```
cantp_status result;
result = CanTpApi.AddFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
       MessageBox.Show("Error adding filter.", "Error");
```

C++ / CLR

```
cantp_status result;
result = CanTpApi_2016::AddFiltering_2016(PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       PCANTP_CAN_MSGTYPE_STANDARD);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Error adding filter.", "Error");
```

Visual Basic

```
Dim result As cantp_status
result = CanTpApi.AddFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, &HD1, &HD2, False,
    cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Error adding filter.", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
begin
  result := TCanTpApi.AddFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, $D1,
    $D2, false, cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Error adding filter.', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_AddFiltering_2016 on page 233

See also: RemoveFiltering_2016 below

3.6.14 RemoveFiltering_2016

Removes an entry from the CAN-ID white-list filtering.

Syntax

Pascal OO

```
class function RemoveFiltering_2016(
    channel: cantp_handle;
    can_id_from: UInt32;
    can_id_to: UInt32;
    ignore_can_msgtype: Boolean;
    can_msgtype: cantp_can_msgtype
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveFiltering_2016")]
```

```
public static extern cantp_status RemoveFiltering_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    UInt32 can_id_from,
    UInt32 can_id_to,
    [MarshalAs(UnmanagedType.U1)]
    bool ignore_can_msgtype,
    [MarshalAs(UnmanagedType.U4)]
    cantp_can_msgtype can_msgtype);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_RemoveFiltering_2016")]
static cantp_status RemoveFiltering_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       UInt32 can_id_from,
       UInt32 can_id_to,
       [MarshalAs(UnmanagedType::U1)]
       bool ignore_can_msgtype,
       [MarshalAs(UnmanagedType::U4)]
       cantp_can_msgtype can_msgtype);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_RemoveFiltering_2016")>
Public Shared Function RemoveFiltering_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByVal can_id_from As UInt32,
    ByVal can_id_to As UInt32,
    <MarshalAs(UnmanagedType.U1)>
    ByVal ignore_can_msgtype As Boolean,
    <MarshalAs(UnmanagedType.U4)>
    ByVal can_msgtype As cantp_can_msgtype) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
can_id_from	The lowest CAN ID wanted to be removed (see AddFiltering_2016 on page 126).
can_id_to	The highest CAN ID wanted to be removed (see AddFiltering_2016 on page 126).
ignore_can_msgtype	ignore_can_msgtype boolean of the filter to remove (see AddFiltering_2016 on page 126).
can_msgtype	can_msgtype of the filter to remove (see AddFiltering_2016 on page 126).

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	The filter to remove is not in the filtering list.

Example

The following example shows the use of the method RemoveFiltering_2016 on the channel PCANTP_HANDLE_USBBUS1. This example adds a filter from 0xD1 can identifier to 0xD2 can identifier then removes it.

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

C#

```
cantp_status result;
result = CanTpApi.AddFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
        cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Error adding filter.", "Error");

// Remove the previously added filter
result = CanTpApi.RemoveFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
        cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
        MessageBox.Show("Error removing filter.", "Error");
```

C++ / CLR

```
cantp_status result;
result = CanTpApi_2016::AddFiltering_2016(PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       PCANTP_CAN_MSGTYPE_STANDARD);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Error adding filter.", "Error");

// Remove the previously added filter
result = CanTpApi_2016::RemoveFiltering_2016(PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       PCANTP_CAN_MSGTYPE_STANDARD);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("Error removing filter.", "Error");
```

Visual Basic

```
Dim result As cantp_status
result = CanTpApi.AddFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, &HD1, &HD2, False,
    cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Error adding filter.", "Error")
End If

' Remove the previously added filter
result = CanTpApi.RemoveFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, &HD1, &HD2, False,
    cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Error removing filter.", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
begin
  result := TCanTpApi.AddFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, $D1,
    $D2, false, cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Error adding filter.', 'Error', MB_OK);
  end;

  // Remove the previously added filter
  result := TCanTpApi.RemoveFiltering_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    $D1, $D2, false, cantp_can_msgtype.PCANTP_CAN_MSGTYPE_STANDARD);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'Error removing filter.', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_RemoveFiltering_2016 on page 234

See also: AddFiltering_2016 on page 126

3.6.15 GetValue_2016

Retrieves information from a PCAN channel.

Overloads

	Method	Description
	GetValue_2016(cantp_handle, cantp_parameter, UInt32, UInt32)	Retrieves information from a PCANTP channel in numeric form.
	GetValue_2016(cantp_handle, cantp_parameter, Byte[], UInt32)	Retrieves information from a PCANTP channel in byte array form.
	GetValue_2016(cantp_handle, cantp_parameter, String, UInt32)	Retrieves information from a PCANTP channel in text form.

Plain function version: CANTP_GetValue_2016 on page 235

3.6.16 GetValue_2016(cantp_handle, cantp_parameter, String, UInt32)

Retrieves information from a PCANTP channel in text form.

Syntax

Pascal OO

```
class function GetValue_2016(
    channel: cantp_handle;
    parameter: cantp_parameter;
    StringBuffer: PAnsiChar;
    buffer_length: UInt32
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue_2016")]
public static extern cantp_status GetValue_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_parameter parameter,
    StringBuilder StringBuffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue_2016")]
       static cantp_status GetValue_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_parameter parameter,
       StringBuilder ^StringBuffer,
       UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetValue_2016")>
Public Shared Function GetValue_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal parameter As cantp_parameter,
    ByVal StringBuffer As StringBuilder,
    ByVal BufferLength As UInt32) As cantp_status

End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
parameter	The code of the value to retrieve (see cantp_parameter on page 70)
StringBuffer	The buffer to return the required string value.
BufferLength	The length in bytes of the given buffer.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the method are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length (see cantp_parameter on page 70).

Example

The following example shows the use of the method GetValue_2016 to retrieve the version of the ISO-TP API. Depending on the result, a message will be shown to the user.

C#

```
cantp_status result;
StringBuilder BufferString;

// Get API version.
BufferString = new StringBuilder(255);
result = CanTpApi.GetValue_2016(cantp_handle.PCANTP_HANDLE_NONEBUS,
cantp_parameter.PCANTP_PARAMETER_API_VERSION, BufferString, 255);
if (result != cantp_status.PCANTP_STATUS_OK)
        MessageBox.Show("Failed to get value");
else
        MessageBox.Show(BufferString.ToString());
```

C++ / CLR

```
cantp_status result;
StringBuilder^ BufferString;

// Get API version.
BufferString = gcnew StringBuilder(255);
result = CanTpApi_2016::GetValue_2016(PCANTP_HANDLE_NONEBUS, PCANTP_PARAMETER_API_VERSION,
       BufferString, 255);
if (result != PCANTP_STATUS_OK)
       MessageBox::Show("Failed to get value");
```

```
else
        MessageBox::Show(BufferString->ToString());
```

Visual Basic

```
Dim result As cantp_status
Dim BufferString As StringBuilder

' Get API version.
BufferString = New StringBuilder(255)
result = CanTpApi.GetValue_2016(cantp_handle.PCANTP_HANDLE_NONEBUS,
    cantp_parameter.PCANTP_PARAMETER_API_VERSION, BufferString, 255)
If result <> cantp_status.PCANTP_STATUS_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(BufferString.ToString())
End If
```

Pascal OO

```
var
  result: cantp_status;
  BufferString: array [0 .. 254] of ansichar;
begin
  // Get API version.
  result := TCanTpApi.GetValue_2016(cantp_handle.PCANTP_HANDLE_NONEBUS,
    cantp_parameter.PCANTP_PARAMETER_API_VERSION, BufferString, 255);
  if result <> cantp_status.PCANTP_STATUS_OK then
  begin
    MessageBox(0, 'Failed to get value', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, PWideChar(String(BufferString)), 'Info', MB_OK);
  end;
end;
```

Plain function version: CANTP_GetValue_2016 on page 235

See also: SetValue_2016 on page 110, cantp_parameter on page 70, Detailed parameters values on page 75

3.6.17 GetValue_2016(cantp_handle, cantp_parameter, UInt32, UInt32)

Retrieves information from a PCAN channel in numeric form.

Syntax

Pascal OO

```
class function GetValue_2016(
    channel: cantp_handle;
    parameter: cantp_parameter;
    numericBuffer: PLongWord;
    buffer_length: UInt32
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue_2016")]
public static extern cantp_status GetValue_2016(
    [MarshalAs(UnmanagedType.U4)]
```

```
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_parameter parameter,
    out UInt32 NumericBuffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue_2016")]
       static cantp_status GetValue_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_parameter parameter,
       UInt32 %NumericBuffer,
       UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetValue_2016")>
Public Shared Function GetValue_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal parameter As cantp_parameter,
    ByRef NumericBuffer As UInt32,
    ByVal BufferLength As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
parameter	The code of the value to retrieve (see cantp_parameter on page 70).
NumericBuffer	The buffer to return the required numeric value.
BufferLength	The length in bytes of the given buffer.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the method are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length (see cantp_parameter on page 70).

Example

The following example shows the use of the method GetValue_2016 on the channel PCANTP_HANDLE_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#

```
cantp_handle CanChannel = cantp_handle.PCANTP_HANDLE_USBBUS1;
cantp_status result;
UInt32 iBuffer = 0;
```

```
// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue_2016(CanChannel, cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME,
        out iBuffer, sizeof(UInt32));
if (result != cantp_status.PCANTP_STATUS_OK)
        MessageBox.Show("Failed to get value");
else
        MessageBox.Show(iBuffer.ToString());
```

C++ / CLR

```
cantp_handle CanChannel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
UInt32 iBuffer = 0;

// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi_2016::GetValue_2016(CanChannel, PCANTP_PARAMETER_SEPARATION_TIME, iBuffer,
       sizeof(UInt32));
if (result != PCANTP_STATUS_OK)
       MessageBox::Show("Failed to get value");
else
       MessageBox::Show(iBuffer.ToString());
```

Visual Basic

```
Dim CanChannel As cantp_handle = cantp_handle.PCANTP_HANDLE_USBBUS1
Dim result As cantp_status
Dim iBuffer As UInt32 = 0

' Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue_2016(CanChannel, cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME,
    iBuffer, Len(iBuffer))
If result <> cantp_status.PCANTP_STATUS_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(iBuffer.ToString())
End If
```

Pascal OO

```
var
  result: cantp_status;
  CanChannel: cantp_handle;
  iBuffer: UInt32;
begin
  CanChannel := cantp_handle.PCANTP_HANDLE_USBBUS1;

  // Gets the value of the ISO-TP Separation Time (STmin) parameter.
  iBuffer := 0;
  result := TCanTpApi.GetValue_2016(CanChannel,
    cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME, PLongWord(@iBuffer),
    sizeof(iBuffer));
  if result <> cantp_status.PCANTP_STATUS_OK then
  begin
    MessageBox(0, 'Failed to get value', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, PWideChar(Format('%d', [Integer(iBuffer)])), 'Info', MB_OK);
  end;
end;
```

Plain function version: CANTP_GetValue_2016 on page 235

See also: SetValue_2016 on page 110, cantp_parameter on page 70, Detailed parameters values on page 75

3.6.18 GetValue_2016(cantp_handle, cantp_parameter, Byte[], UInt32)

Retrieves information from a PCAN channel in a byte array.

Syntax

Pascal OO

```
class function GetValue_2016(
    channel: cantp_handle;
    parameter: cantp_parameter;
    byteBuffer: PByte;
    buffer_length: UInt32
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue_2016")]
public static extern cantp_status GetValue_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.U4)]
    cantp_parameter parameter,
    [MarshalAs(UnmanagedType.LPArray)]
    [Out] Byte[] Buffer,
    UInt32 BufferLength);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetValue_2016")]
static cantp_status GetValue_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::U4)]
       cantp_parameter parameter,
       [MarshalAs(UnmanagedType::LPArray)]
       [Out] array<Byte>^ Buffer,
       UInt32 BufferLength);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetValue_2016")>
Public Shared Function GetValue_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.U4)>
    ByVal parameter As cantp_parameter,
    <MarshalAs(UnmanagedType.LPArray)>
    <Out> ByVal Buffer As Byte(),
    ByVal BufferLength As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
parameter	The code of the value to retrieve (see cantp_parameter on page 70).

Buffer	The buffer containing the array value to retrieve.
BufferLength	The length in bytes of the given buffer.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the method are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length (see cantp_parameter on page 70).

Example

The following example shows the use of the method GetValue_2016 on the channel PCANTP_HANDLE_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#

```
cantp_handle CanChannel = cantp_handle.PCANTP_HANDLE_USBBUS1;
cantp_status result;
uint bufferLength = 2;
byte[] bufferArray = new byte[bufferLength];

// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue_2016(CanChannel, cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME,
        bufferArray, sizeof(byte) * bufferLength);
if (result != cantp_status.PCANTP_STATUS_OK)
        MessageBox.Show("Failed to get value");
else
        MessageBox.Show(bufferArray[0].ToString());
```

C++ / CLR

```
cantp_handle CanChannel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
UInt32 bufferLength = 2;
array<Byte>^ bufferArray = gcnew array<Byte>(bufferLength);

// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi_2016::GetValue_2016(CanChannel, PCANTP_PARAMETER_SEPARATION_TIME,
       bufferArray, sizeof(Byte) * bufferLength);
if (result != PCANTP_STATUS_OK)
       MessageBox::Show("Failed to get value");
else
       MessageBox::Show(bufferArray[0].ToString());
```

Visual Basic

```
Dim CanChannel As cantp_handle = cantp_handle.PCANTP_HANDLE_USBBUS1
Dim result As cantp_status
Dim bufferLength As UInt32 = 2
Dim bufferArray(bufferLength) As Byte

' Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CanTpApi.GetValue_2016(CanChannel, cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME,
```

```
    bufferArray, bufferLength)
If result <> cantp_status.PCANTP_STATUS_OK Then
    MessageBox.Show("Failed to get value")
Else
    MessageBox.Show(bufferArray(0).ToString())
End If
```

Pascal OO

```
var
  result: cantp_status;
  CanChannel: cantp_handle;
  bufferLength: UInt32;
  bufferArray: array [0 .. 2] of Byte;
begin
  CanChannel := cantp_handle.PCANTP_HANDLE_USBBUS1;
  bufferLength := 2;

  // Gets the value of the ISO-TP Separation Time (STmin) parameter.
  result := TCanTpApi.GetValue_2016(CanChannel,
    cantp_parameter.PCANTP_PARAMETER_SEPARATION_TIME, PByte(@bufferArray),
    sizeof(Byte) * bufferLength);
  if result <> cantp_status.PCANTP_STATUS_OK then
  begin
    MessageBox(0, 'Failed to get value', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, PWideChar(bufferArray[0].ToString()), 'Info', MB_OK);
  end;
end;
```

Plain function version: CANTP_GetValue_2016 on page 235

See also: SetValue_2016 on page 110, cantp_parameter on page 70, Detailed parameters values on page 75

3.6.19 GetErrorText_2016

Gets a descriptive text for a given cantp_status error code.

Syntax

Pascal OO

```
class function GetErrorText_2016(
    error: cantp_status;
    language: UInt16;
    StringBuffer: PAnsiChar;
    bufferSize: UInt32
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetErrorText_2016")]
public static extern cantp_status GetErrorText_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_status error,
    UInt16 language,
    StringBuilder StringBuffer,
    UInt32 bufferSize);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetErrorText_2016")]
static cantp_status GetErrorText_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_status error,
       UInt16 language,
       StringBuilder ^StringBuffer,
       UInt32 BufferSize);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetErrorText_2016")>
Public Shared Function GetErrorText_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal err As cantp_status,
    ByVal language As UInt16,
    ByVal StringBuffer As StringBuilder,
    ByVal BufferSize As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
error	A cantp_status error code (see cantp_status on page 60).
language	The current languages available for translation are: Neutral (0x00), German (0x07), English (0x09), Spanish (0x0A), Italian (0x10) and French (0x0C).
StringBuffer	A buffer for a null-terminated char array.
bufferSize	Buffer length in bytes.

Returns

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

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

Remarks

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

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

Note: If the buffer is too small for the resulting text, the error 0x80008000 (PCANTP_STATUS_MASK_PCAN | PCAN_ERROR_ILLPARAMVAL) is returned. Even when only short texts are being currently returned, a text within this method 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_2016 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#

```
StringBuilder str_msg = new StringBuilder(256);
cantp_status result;
cantp_status error_result;
error_result = CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1);
result = CanTpApi.GetErrorText_2016(error_result, 0x0, str_msg, 256);
if (CanTpApi.StatusIsOk_2016(result))
        MessageBox.Show(str_msg.ToString(), "Error on uninitialized");
```

C++ / CLR

```
StringBuilder^ str_msg = gcnew StringBuilder(256);
cantp_status result;
cantp_status error_result;
error_result = CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_USBBUS1);
result = CanTpApi_2016::GetErrorText_2016(error_result, 0x0, str_msg, 256);
if (CanTpApi_2016::StatusIsOk_2016(result))
       MessageBox::Show(str_msg->ToString(), "Error on uninitialized");
```

Visual Basic

```
Dim str_msg As StringBuilder
str_msg = New StringBuilder(256)
Dim result As cantp_status
Dim error_result As cantp_status
error_result = CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1)
result = CanTpApi.GetErrorText_2016(error_result, &H0, str_msg, 256)
If CanTpApi.StatusIsOk_2016(result) Then
    MessageBox.Show(str_msg.ToString(), "Error on uninitialized")
End If
```

Pascal OO

```
var
  result: cantp_status;
  error_result: cantp_status;
  str_msg: array [0 .. 255] of ansichar;
begin
  error_result := TCanTpApi.Uninitialize_2016
    (cantp_handle.PCANTP_HANDLE_USBBUS1);
  result := TCanTpApi.GetErrorText_2016(error_result, $0, str_msg, 256);
  if TCanTpApi.StatusIsOk_2016(result) then
  begin
    MessageBox(0, PWideChar(String(str_msg)), 'Error on uninitialized', MB_OK);
  end;
end;
```

Plain function version: CANTP_GetErrorText_2016 on page 239

See also: cantp_status on page 60

3.6.20 GetCanBusStatus_2016

Gets information about the internal BUS status of a PCANTP channel.

Syntax

Pascal OO

```
class function GetCanBusStatus_2016(
    channel: cantp_handle
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetCanBusStatus_2016")]
public static extern cantp_status GetCanBusStatus_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetCanBusStatus_2016")]
static cantp_status GetCanBusStatus_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetCanBusStatus_2016")>
Public Shared Function GetCanBusStatus_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)

Returns

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

PCANTP_STATUS_OK	Indicates that the status of the given PCANTP channel is OK.
PCANTP_STATUS_FLAG_BUS_LIGHT	Indicates a bus error within the given PCANTP channel. The hardware is in bus-light status.
PCANTP_STATUS_FLAG_BUS_HEAVY	Indicates a bus error within the given PCANTP channel. The hardware is in bus-heavy status.
PCANTP_STATUS_FLAG_BUS_OFF	Indicates a bus error within the given PCANTP channel. The hardware is in bus-off status.
PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.

Remarks

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

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

Example

The following example shows the use of the method GetCanBusStatus_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;
result = CanTpApi.GetCanBusStatus_2016(cantp_handle.PCANTP_HANDLE_PCIBUS1);

// Checks the status of the PCI channel.
switch (result)
{
        case cantp_status.PCANTP_STATUS_FLAG_BUS_LIGHT:
                MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...", "Success");
                break;
        case cantp_status.PCANTP_STATUS_FLAG_BUS_HEAVY:
                MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...", "Success");
                break;
        case cantp_status.PCANTP_STATUS_FLAG_BUS_OFF:
                MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...", "Success");
                break;
        case cantp_status.PCANTP_STATUS_OK:
                MessageBox.Show("PCAN-PCI (Ch-1): Status is OK", "Success");
                break;
        default:
                // An error occurred.
                MessageBox.Show("Failed to retrieve status", "Error");
                break;
}
```

C++ / CLR

```
cantp_status result;
result = CanTpApi_2016::GetCanBusStatus_2016(PCANTP_HANDLE_PCIBUS1);

// Checks the status of the PCI channel.
switch (result)
{
case PCANTP_STATUS_FLAG_BUS_LIGHT:
       MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...", "Success");
       break;
case PCANTP_STATUS_FLAG_BUS_HEAVY:
       MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...", "Success");
       break;
case PCANTP_STATUS_FLAG_BUS_OFF:
       MessageBox::Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...", "Success");
       break;
case PCANTP_STATUS_OK:
       MessageBox::Show("PCAN-PCI (Ch-1): Status is OK", "Success");
       break;
default:
       // An error occurred.
       MessageBox::Show("Failed to retrieve status", "Error");
       break;
}
```

Visual Basic

```
Dim result As cantp_status
result = CanTpApi.GetCanBusStatus_2016(cantp_handle.PCANTP_HANDLE_PCIBUS1)
```

```
' Checks the status of the PCI channel.
Select Case (result)
    Case cantp_status.PCANTP_STATUS_FLAG_BUS_LIGHT
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...", "Success")
    Case cantp_status.PCANTP_STATUS_FLAG_BUS_HEAVY
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...", "Success")
    Case cantp_status.PCANTP_STATUS_FLAG_BUS_OFF
        MessageBox.Show("PCAN-PCI (Ch-1): Handling a BUS-OFF status...", "Success")
    Case cantp_status.PCANTP_STATUS_OK
        MessageBox.Show("PCAN-PCI (Ch-1): Status is OK", "Success")
    Case Else
        ' An error occurred.
        MessageBox.Show("Failed to retrieve status", "Error")
End Select
```

Pascal OO

```
var
  result: cantp_status;
begin
  result := TCanTpApi.GetCanBusStatus_2016(cantp_handle.PCANTP_HANDLE_PCIBUS1);

  // Checks the status of the PCI channel.
  Case (result) of
    cantp_status.PCANTP_STATUS_FLAG_BUS_LIGHT:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...',
        'Success', MB_OK);
    cantp_status.PCANTP_STATUS_FLAG_BUS_HEAVY:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...',
        'Success', MB_OK);
    cantp_status.PCANTP_STATUS_FLAG_BUS_OFF:
      MessageBox(0, 'PCAN-PCI (Ch-1): Handling a BUS-OFF status...',
        'Success', MB_OK);
    cantp_status.PCANTP_STATUS_OK:
      MessageBox(0, 'PCAN-PCI (Ch-1): Status is OK', 'Success', MB_OK);
  else
    // An error occurred.
    MessageBox(0, 'Failed to retrieve status', 'Error', MB_OK);
  End;
end;
```

Plain function version: CANTP_GetCanBusStatus_2016 on page 240

See also: cantp_status on page 60

3.6.21 GetMsgProgress_2016

Gets progress information on a specific message.

Syntax

Pascal OO

```
class function GetMsgProgress_2016(
    channel: cantp_handle;
    msg_buffer: Pcantp_msg;
    direction: cantp_msgdirection;
    var msgprogress_buffer: cantp_msgprogress
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetMsgProgress_2016")]
public static extern cantp_status GetMsgProgress_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [In]ref cantp_msg msg_buffer,
    [MarshalAs(UnmanagedType.U4)]
    cantp_msgdirection direction,
    out cantp_msgprogress msgprogress_buffer);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetMsgProgress_2016")]
static cantp_status GetMsgProgress_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [In] cantp_msg %msg_buffer,
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgdirection direction,
       cantp_msgprogress %msgprogress_buffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetMsgProgress_2016")>
Public Shared Function GetMsgProgress_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByRef msg_buffer As cantp_msg,
    <MarshalAs(UnmanagedType.U4)>
    ByVal direction As cantp_msgdirection,
    ByRef msgprogress_buffer As cantp_msgprogress) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
msg_buffer	A cantp_msg structure buffer matching the message to look for (see cantp_msg on page 28).
direction	The expected direction (incoming/outgoing) of the message (see cantp_msgdirection on page 96).
msgprogress_buffer	A cantp_msgprogress structure buffer to store the progress information (see cantp_msgprogress on page 31).

Returns

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

PCANTP_STATUS_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_STATUS_PARAM_INVALID_VALUE	The cantp_msg message or the cantp_msgprogress buffer is invalid.
PCANTP_STATUS_NO_MESSAGE	The message is unknown.
PCANTP_STATUS_LOCK_TIMEOUT	Internal lock timeout while searching the message within internal queues.

Example

The following example shows the use of the method GetMsgProgress_2016 when receiving a loopback message on the PCANTP channel USB 1. Depending on the result, progress will be shown to the user.

Note: It is assumed that the channel was already initialized and a heavy ISOTP message has been sent.

C#

```
cantp_status result;
cantp_msg loopback_msg = new cantp_msg();

// Read transmission confirmation.
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out loopback_msg);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)
        && (cantp_msgtype.PCANTP_MSGTYPE_ISOTP & loopback_msg.type)
        == cantp_msgtype.PCANTP_MSGTYPE_ISOTP
        && ((loopback_msg.Msgdata_isotp_Copy.netaddrinfo.msgtype
        & cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX)
        == cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX))
{
    // The ISOTP loopback message is being received, wait and show progress
    cantp_msgprogress progress = new cantp_msgprogress();
    do
    {
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
                ref loopback_msg, cantp_msgdirection.PCANTP_MSGDIRECTION_TX, out progress);
        MessageBox.Show("RX Progress on loopback message: " + progress.percentage, "Success");
    } while (progress.state == cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING);
}
else
{
     MessageBox.Show("Read error: " + result.ToString(), "Error");
}
```

C++ / CLR

```
cantp_status result;
cantp_msg loopback_msg = {};

// Read transmission confirmation.
result = CanTpApi_2016::Read_2016(PCANTP_HANDLE_USBBUS1, loopback_msg);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK)
       && (PCANTP_MSGTYPE_ISOTP & loopback_msg.type) == PCANTP_MSGTYPE_ISOTP
       && ((loopback_msg.msgdata.isotp->netaddrinfo.msgtype
       & PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX)
       == PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX))
{
       // The ISOTP loopback message is being received, wait and show progress
       cantp_msgprogress progress = {};
       do
       {
              result = CanTpApi_2016::GetMsgProgress_2016(PCANTP_HANDLE_USBBUS1, loopback_msg,
                     PCANTP_MSGDIRECTION_TX, progress);
              MessageBox::Show("RX Progress on loopback message: " + progress.percentage,
                     "Success");
       } while (progress.state == PCANTP_MSGPROGRESS_STATE_PROCESSING);
}
else
{
       MessageBox::Show(String::Format("Read error: {0}", (int)result), "Error");
}
```

Visual Basic

```
Dim result As cantp_status
Dim loopback_msg As cantp_msg

' Read transmission confirmation.
```

```
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, loopback_msg)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) And
        (cantp_msgtype.PCANTP_MSGTYPE_ISOTP And loopback_msg.type) =
        cantp_msgtype.PCANTP_MSGTYPE_ISOTP And
        (loopback_msg.Msgdata_isotp_Copy.netaddrinfo.msgtype And
        cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX) =
        cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX Then

    ' The ISOTP loopback message Is being received, wait And show progress
    Dim progress As cantp_msgprogress
    Do
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, loopback_msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_TX, progress)
        MessageBox.Show("RX Progress on loopback message: " + progress.percentage.ToString(),
            "Success")
    Loop While progress.state = cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING
Else
    MessageBox.Show("Read error: " + result.ToString(), "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  loopback_msg: cantp_msg;
  progress: cantp_msgprogress;
begin

      // Read transmission confirmation.
      result := TCanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
        loopback_msg);
      if (TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) AND
        (cantp_msgtype(UInt32(cantp_msgtype.PCANTP_MSGTYPE_ISOTP) and
        UInt32(loopback_msg.typem)) = cantp_msgtype.PCANTP_MSGTYPE_ISOTP) AND
        (cantp_isotp_msgtype(UInt32(loopback_msg.msgdata_isotp.netaddrinfo.
        msgtype) and
        UInt32(cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX))
        = cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX)) then
      begin

        // The ISOTP loopback message is being received, wait and show progress
        repeat
          result := TCanTpApi.GetMsgProgress_2016
            (cantp_handle.PCANTP_HANDLE_USBBUS1, @loopback_msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_TX, progress);
          MessageBox(0,
            PWideChar(format('RX Progress on loopback message: %d%%',
            [Integer(progress.percentage)])), 'Success', MB_OK);
        until progress.state <> cantp_msgprogress_state.
          PCANTP_MSGPROGRESS_STATE_PROCESSING;
      end
      else
      begin
        MessageBox(0, PWideChar(format('Read error: %d', [Integer(result)])),
          'Error', MB_OK);
      end;
end;
```

Plain function version: CANTP_GetMsgProgress_2016 on page 238

See also: cantp_msgprogress on page 31, cantp_msgprogress_state on page 95

3.6.22 GetMappings_2016

Retrieves all the mappings defined for a given PCANTP channel.

Syntax

Pascal OO

```
class function GetMappings_2016(
    channel: cantp_handle;
    buffer: Pcantp_mapping;
    buffer_length: PUInt32
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetMappings_2016")]
public static extern cantp_status GetMappings_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
    [Out] cantp_mapping[] buffer,
    ref UInt32 buffer_length);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_GetMappings_2016")]
static cantp_status GetMappings_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 2)]
       [Out] array<cantp_mapping>^ buffer,
       UInt32 %buffer_length);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_GetMappings_2016")>
Public Shared Function GetMappings_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    <MarshalAs(UnmanagedType.LPArray, SizeParamIndex:=2)>
    <Out> ByVal buffer As cantp_mapping(),
    ByRef buffer_length As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
buffer	A buffer to store an array of cantp_mapping (see cantp_mapping on page 20).
buffer_length	(In) The number of cantp_mapping element the buffer can store. (Out) The actual number of elements copied in the buffer.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	The buffer or the size is invalid.
PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL	The given buffer is to small to store all mappings.

Example

The following example shows the use of the method GetMappings_2016 on PCANTP_HANDLE_USBBUS1. It displays all mappings added on the channel.

Note: It is assumed that the channel and some mappings were already initialized.

C#

```
cantp_status result;
UInt32 count = 256;
cantp_mapping[] mappings = new cantp_mapping[256];
result = CanTpApi.GetMappings_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, mappings, ref count);
if (CanTpApi.StatusIsOk_2016(result))
{
    for (int i = 0; i < count; i++)
    {
        Console.WriteLine("mappings[" + i + "]:");
        Console.WriteLine("\t- can id: " + mappings[i].can_id);
        Console.WriteLine("\t- can id flow control: " + mappings[i].can_id_flow_ctrl);
        Console.WriteLine("\t- can message type: " + mappings[i].can_msgtype);
        Console.WriteLine("\t- extension address: " + mappings[i].netaddrinfo.extension_addr);
        Console.WriteLine("\t- addressing format: " + mappings[i].netaddrinfo.format);
        Console.WriteLine("\t- isotp message type: " + mappings[i].netaddrinfo.msgtype);
        Console.WriteLine("\t- source address: " + mappings[i].netaddrinfo.source_addr);
        Console.WriteLine("\t- target address: " + mappings[i].netaddrinfo.target_addr);
        Console.WriteLine("\t- target type: " + mappings[i].netaddrinfo.target_type);
    }
}
else
{
    Console.WriteLine("Failed to get mappings: " + result.ToString());
}
```

C++ / CLR

```
cantp_status result;
UInt32 count = 256;
array<cantp_mapping>^ mappings = gcnew array<cantp_mapping>(count);
result = CanTpApi_2016::GetMappings_2016(PCANTP_HANDLE_USBBUS1, mappings, count);
if (CanTpApi_2016::StatusIsOk_2016(result))
{
       for (int i = 0; i < count; i++)
       {
              Console::WriteLine("mappings[" + i + "]:");
              Console::WriteLine("\t- can id: " + mappings[i].can_id);
              Console::WriteLine("\t- can id flow control: " + mappings[i].can_id_flow_ctrl);
              Console::WriteLine("\t- can message type: {0}", (int)mappings[i].can_msgtype);
              Console::WriteLine("\t- extension address: " +
                     mappings[i].netaddrinfo.extension_addr);
              Console::WriteLine("\t- addressing format: {0}",
                     (int)mappings[i].netaddrinfo.format);
              Console::WriteLine("\t- isotp message type: {0}",
                     (int)mappings[i].netaddrinfo.msgtype);
              Console::WriteLine("\t- source address: " + mappings[i].netaddrinfo.source_addr);
              Console::WriteLine("\t- target address: " + mappings[i].netaddrinfo.target_addr);
              Console::WriteLine("\t- target type: {0}",
                      (int)mappings[i].netaddrinfo.target_type);
       }
}
else
{
```

```
        Console::WriteLine("Failed to get mappings: {0}", (int)result);
}
```

Visual Basic

```
Dim result As cantp_status
Dim count As UInt32 = 256
Dim mappings(count) As cantp_mapping
result = CanTpApi.GetMappings_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, mappings, count)
If CanTpApi.StatusIsOk_2016(result) Then
    For i As UInt32 = 0 To count - 1
        Console.WriteLine("mappings[" + i.ToString() + "]:")
        Console.WriteLine("    - can id: " + mappings(i).can_id.ToString())
        Console.WriteLine("    - can id flow control: " +
            mappings(i).can_id_flow_ctrl.ToString())
        Console.WriteLine("    - can message type: " + mappings(i).can_msgtype.ToString())
        Console.WriteLine("    - extension address: " +
            mappings(i).netaddrinfo.extension_addr.ToString())
        Console.WriteLine("    - addressing format: " +
            mappings(i).netaddrinfo.format.ToString())
        Console.WriteLine("    - isotp message type: " +
            mappings(i).netaddrinfo.msgtype.ToString())
        Console.WriteLine("    - source address: " +
            mappings(i).netaddrinfo.source_addr.ToString())
        Console.WriteLine("    - target address: " +
            mappings(i).netaddrinfo.target_addr.ToString())
        Console.WriteLine("    - target type: " +
            mappings(i).netaddrinfo.target_type.ToString())
    Next
Else
    Console.WriteLine("Failed to get mappings: " + result)
End If
```

Pascal OO

```
var
  result: cantp_status;
  i: UInt32;
  count: UInt32;
  mappings: array [0 .. 256] of cantp_mapping;
begin
  count := 256;
  result := TCanTpApi.GetMappings_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    @mappings, @count);
  if TCanTpApi.StatusIsOk_2016(result) then
  begin

    for i := 0 to count - 1 do
    begin
      WriteLn(format('mappings[%d]:', [Integer(i)]));
      WriteLn(format('    - can id: %d', [Integer(mappings[i].can_id)]));
      WriteLn(format('    - can id flow control: %d',
        [Integer(mappings[i].can_id_flow_ctrl)]));
      WriteLn(format('    - can message type: %d',
        [Integer(mappings[i].can_msgtype)]));
      WriteLn(format('    - extension address: %d',
        [Integer(mappings[i].netaddrinfo.extension_addr)]));
      WriteLn(format('    - addressing format: %d',
        [Integer(mappings[i].netaddrinfo.format)]));
      WriteLn(format('    - isotp message type: %d',
        [Integer(mappings[i].netaddrinfo.msgtype)]));
      WriteLn(format('    - source address: %d',
```

```
        [Integer(mappings[i].netaddrinfo.source_addr)]));
      WriteLn(format('    - target address: %d',
        [Integer(mappings[i].netaddrinfo.target_addr)]));
      WriteLn(format('    - target type: %d',
        [Integer(mappings[i].netaddrinfo.target_type)]));
    end;
  end
  else
  begin
    WriteLn(format('Failed to get mappings: %d', [Integer(result)]));
  end;
end;
```

Plain function version: CANTP_GetMappings_2016 on page 241

See also: cantp_mapping on page 20, AddMapping_2016 on page 116

3.6.23 StatusGet_2016

Retrieves the value of a cantp_status subtype (like cantp_errstatus, cantp_busstatus, etc.).

Syntax

Pascal OO

```
class function StatusGet_2016(
    const status: cantp_status;
    const typest: cantp_statustype
): UInt32;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_StatusGet_2016")]
public static extern UInt32 StatusGet_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_status status,
    [MarshalAs(UnmanagedType.U4)]
    cantp_statustype type);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_StatusGet_2016")]
static UInt32 StatusGet_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_status status,
       [MarshalAs(UnmanagedType::U4)]
       cantp_statustype type);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_StatusGet_2016")>
Public Shared Function StatusGet_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal status As cantp_status,
    <MarshalAs(UnmanagedType.U4)>
    ByVal type As cantp_statustype) As UInt32
End Function
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).
type	The type of status to filter (see cantp_statustype on page 51).

Returns

The return is the value of the enumeration matching the requested type.

Example

The following example shows the use of the method StatusGet_2016 on a status from Uninitialize_2016 on an uninitialized channel. The goal is to generate a PCANTP_STATUS_NOT_INITIALIZED status.

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

C#

```
cantp_status result = CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1);

// Check general error status: should be PCANTP_STATUS_NOT_INITIALIZED (=1)
UInt32 general_error = CanTpApi.StatusGet_2016(result, cantp_statustype.PCANTP_STATUSTYPE_ERR);
if ((cantp_errstatus)general_error != cantp_errstatus.PCANTP_ERRSTATUS_OK)
{
    MessageBox.Show("General error code on uninitialized: " + general_error, "Success");
}

// Check network error status: should be PCANTP_STATUS_OK
UInt32 network_error = CanTpApi.StatusGet_2016(result, cantp_statustype.PCANTP_STATUSTYPE_NET);
if ((cantp_netstatus)network_error != cantp_netstatus.PCANTP_NETSTATUS_OK)
{
    MessageBox.Show("Network error! Code: " + network_error, "Error");
}
```

C++ / CLR

```
cantp_status result = CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_USBBUS1);

// Check general error status: should be PCANTP_STATUS_NOT_INITIALIZED (=1)
UInt32 general_error = CanTpApi_2016::StatusGet_2016(result, PCANTP_STATUSTYPE_ERR);
if ((cantp_errstatus)general_error != PCANTP_ERRSTATUS_OK)
{
       MessageBox::Show("General error code on uninitialized: " + general_error, "Success");
}

// Check network error status: should be PCANTP_STATUS_OK
UInt32 network_error = CanTpApi_2016::StatusGet_2016(result, PCANTP_STATUSTYPE_NET);
if ((cantp_netstatus)network_error != PCANTP_NETSTATUS_OK)
{
       MessageBox::Show("Network error! Code: " + network_error, "Error");
}
```

Visual Basic

```
Dim result As cantp_status
result = CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1)

' Check general error status: should be PCANTP_STATUS_NOT_INITIALIZED (=1)
Dim general_error As UInt32
general_error = CanTpApi.StatusGet_2016(result, cantp_statustype.PCANTP_STATUSTYPE_ERR)
If general_error <> cantp_errstatus.PCANTP_ERRSTATUS_OK Then
```

```
    MessageBox.Show("General error code on uninitialized: " + general_error.ToString(),
        "Success")
End If

' Check network error status: should be PCANTP_STATUS_OK
Dim network_error As UInt32
network_error = CanTpApi.StatusGet_2016(result, cantp_statustype.PCANTP_STATUSTYPE_NET)
If network_error <> cantp_netstatus.PCANTP_NETSTATUS_OK Then
    MessageBox.Show("Network error! Code: " + network_error.ToString(), "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  general_error: UInt32;
  network_error: UInt32;
begin
  result := TCanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1);

  // Check general error status: should be PCANTP_STATUS_NOT_INITIALIZED (=1)
  general_error := TCanTpApi.StatusGet_2016(result,
    cantp_statustype.PCANTP_STATUSTYPE_ERR);
  if cantp_errstatus(general_error) <> cantp_errstatus.PCANTP_ERRSTATUS_OK then
  begin
    MessageBox(0, PWideChar(format('General error code on uninitialized: %d',
      [Integer(general_error)])), 'Success', MB_OK);
  end;

  // Check network error status: should be PCANTP_STATUS_OK
  network_error := TCanTpApi.StatusGet_2016(result,
    cantp_statustype.PCANTP_STATUSTYPE_NET);
  if cantp_netstatus(network_error) <> cantp_netstatus.PCANTP_NETSTATUS_OK then
  begin
    MessageBox(0, PWideChar(format('Network error! Code: %d',
      [Integer(network_error)])), 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_StatusGet_2016 on page 236

See also: cantp_status on page 60, cantp_statustype on page 51

3.6.24 StatusListTypes_2016

Lists the subtypes contained in the PCANTP status.

Syntax

Pascal OO

```
class function StatusListTypes_2016(
    const status: cantp_status
): cantp_statustype;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_StatusListTypes_2016")]
public static extern cantp_statustype StatusListTypes_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_status status);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_StatusListTypes_2016")]
       static cantp_statustype StatusListTypes_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_status status);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_StatusListTypes_2016")>
Public Shared Function StatusListTypes_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal status As cantp_status) As cantp_statustype
End Function
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).

Returns

An aggregation of cantp_statustype values.

Example

The following example shows the use of the method StatusListTypes_2016 on the channel PCANTP_HANDLE_USBBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;
cantp_statustype statustype;
result = CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    statustype = CanTpApi.StatusListTypes_2016(result);

    // Expected type: general error (=1)
    MessageBox.Show("Uninitialize error type: " + statustype, "Error");
}
```

C++ / CLR

```
cantp_status result;
cantp_statustype statustype;
result = CanTpApi_2016::Uninitialize_2016(PCANTP_HANDLE_USBBUS1);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
{
       statustype = CanTpApi_2016::StatusListTypes_2016(result);

       // Expected type: general error (=1)
       MessageBox::Show(String::Format("Uninitialize error type: {0}", (int)statustype),
              "Error");
}
```

Visual Basic

```
Dim result As cantp_status
Dim statustype As cantp_statustype
```

```
result = CanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    statustype = CanTpApi.StatusListTypes_2016(result)

    ' Expected type: general error (=1)
    MessageBox.Show("Uninitialize error type: " + statustype.ToString(), "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  statustype: cantp_statustype;
begin
  result := TCanTpApi.Uninitialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    statustype := TCanTpApi.StatusListTypes_2016(result);

    // Expected type: general error (=1)
    MessageBox(0, PWideChar(format('Uninitialize error type: %d',
      [Integer(statustype)])), 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_StatusListTypes_2016 on page 242

See also: cantp_statustype on page 51, cantp_status on page 60

3.6.25 StatusIsOk_2016

Checks if a cantp_status matches an expected result (default is PCANTP_STATUS_OK).

Overloads

	Method	Description
	StatusIsOk_2016(cantp_status)	Check if the cantp_status matches OK (PCANTP_STATUS_OK) in a non-strict mode.
	StatusIsOk_2016(cantp_status, cantp_status)	Checks if a cantp_status matches the expected result in a non-strict mode.
	StatusIsOk_2016(cantp_status, cantp_status, boolean)	Checks if a cantp_status matches an expected result.

See also: cantp_status on page 60

Plain function version: CANTP_StatusIsOk_2016 on page 237

3.6.26 StatusIsOk_2016(cantp_status)

Check if the status matches OK (PCANTP_STATUS_OK) in a non-strict mode.

Syntax

Pascal OO

```
class function StatusIsOk_2016(
    const status: cantp_status
): boolean; overload;
```

C#

```
public static bool StatusIsOk_2016(
    cantp_status status
);
```

C++ / CLR

```
static bool StatusIsOk_2016(
      cantp_status status,
);
```

Visual Basic

```
Public Shared Function StatusIsOk_2016(
    ByVal status As cantp_status
    ) As Boolean
End Function
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).

Returns

The return value is true if the status matches PCANTP_STATUS_OK. Comparison is not strict meaning that differences like bus or extra information flags are ignored.

Remarks

When comparing a cantp_status, it is preferred to use StatusIsOk_2016 instead of comparing it with the "==" operator because StatusIsOk_2016 can remove information flag.

Example

The following example shows the use of the method StatusIsOk_2016 after initializing the channel PCANTP_HANDLE_USBBUS1.

C#

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
if (CanTpApi.StatusIsOk_2016(result))
    MessageBox.Show("PCAN-USB (Ch-1) was initialized", "Success");
else
    MessageBox.Show("Initialization failed", "Error");
```

C++ / CLR

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CanTpApi_2016::Initialize_2016(PCANTP_HANDLE_USBBUS1, PCANTP_BAUDRATE_500K);
if (CanTpApi_2016::StatusIsOk_2016(result))
      MessageBox::Show("PCAN-USB (Ch-1) was initialized", "Success");
else
      MessageBox::Show("Initialization failed", "Error");
```

Visual Basic

```
Dim result As cantp_status

' The Plug and Play channel USB1 is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K)
If CanTpApi.StatusIsOk_2016(result) Then
    MessageBox.Show("PCAN-USB (Ch-1) was initialized", "Success")
Else
    MessageBox.Show("Initialization failed", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Plug and Play channel USB1 is initialized.
  result := TCanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
  if TCanTpApi.StatusIsOk_2016(result) then
  begin
    MessageBox(0, 'PCAN-USB (Ch-1) was initialized', 'Success', MB_OK);
  end
  else
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_StatusIsOk_2016 on page 237

See also: cantp_status on page 60

3.6.27 StatusIsOk_2016(cantp_status, cantp_status)

Checks if a cantp_status matches the expected result in a non-strict mode, meaning that differences like bus or extra information flags are ignored.

Syntax

Pascal OO

```
class function StatusIsOk_2016(
    const status: cantp_status;
    const status_expected: cantp_status
): boolean; overload;
```

C#

```
public static bool StatusIsOk_2016(
    cantp_status status,
    cantp_status status_expected
);
```

C++ / CLR

```
static bool StatusIsOk_2016(
       cantp_status status,
```

```
        cantp_status status_expected
);
```

Visual Basic

```
Public Shared Function StatusIsOk_2016(
    ByVal status As cantp_status,
    ByVal status_expected As cantp_status
    ) As Boolean
End Function
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).
status_expected	The expected status (see cantp_status on page 60). The default value is PCANTP_STATUS_OK.

Returns

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

Remarks

When comparing a cantp_status, it is preferred to use StatusIsOk_2016 instead of comparing it with the "==" operator because StatusIsOk_2016 can remove information flag.

Example

The following example shows the use of the method StatusIsOk_2016 after initializing the channel PCANTP_HANDLE_USBBUS1.

C#

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
    MessageBox.Show("PCAN-USB (Ch-1) was initialized", "Success");
else
    MessageBox.Show("Initialization failed", "Error");
```

C++ / CLR

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CanTpApi_2016::Initialize_2016(PCANTP_HANDLE_USBBUS1, PCANTP_BAUDRATE_500K);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show("PCAN-USB (Ch-1) was initialized", "Success");
else
       MessageBox::Show("Initialization failed", "Error");
```

Visual Basic

```
Dim result As cantp_status

' The Plug and Play channel USB1 is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
```

```
    MessageBox.Show("PCAN-USB (Ch-1) was initialized", "Success")
Else
    MessageBox.Show("Initialization failed", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Plug and Play channel USB1 is initialized.
  result := TCanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, 'PCAN-USB (Ch-1) was initialized', 'Success', MB_OK);
  end
  else
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_StatusIsOk_2016 on page 237; **See also:** cantp_status on page 60

3.6.28 StatusIsOk_2016(cantp_status, cantp_status, boolean)

Checks if a cantp_status matches an expected result (default is PCANTP_STATUS_OK).

Syntax

Pascal OO

```
class function StatusIsOk_2016(
    const status: cantp_status;
    const status_expected: cantp_status;
    strictc: Boolean
): boolean; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_StatusIsOk_2016")]
[return: MarshalAs(UnmanagedType.I1)]
public static extern bool StatusIsOk_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_status status,
    [MarshalAs(UnmanagedType.U4)]
    cantp_status status_expected,
    [MarshalAs(UnmanagedType.I1)]
    bool strict);
```

C++ / CLR

```
static bool StatusIsOk_2016(
       cantp_status status,
       cantp_status status_expected,
       bool strict);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_StatusIsOk_2016")>
Public Shared Function StatusIsOk_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal status As cantp_status,
    <MarshalAs(UnmanagedType.U4)>
    ByVal status_expected As cantp_status,
    <MarshalAs(UnmanagedType.I1)>
    ByVal strict As Boolean) As Byte
End Function
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).
status_expected	The expected status (see cantp_status on page 60). The default value is PCANTP_STATUS_OK.
strict	Enable strict mode (default is false). Strict mode ensures that bus or extra information are the same.

Returns

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

Remarks

When comparing a cantp_status, it is preferred to use StatusIsOk_2016 instead of comparing it with the "==" operator because StatusIsOk_2016 can remove information flag.

Example

The following example shows the use of the method StatusIsOk_2016 after initializing the channel PCANTP_HANDLE_USBBUS1.

C#

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    MessageBox.Show("PCAN-USB (Ch-1) was initialized", "Success");
else
    MessageBox.Show("Initialization failed", "Error");
```

C++ / CLR

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CanTpApi_2016::Initialize_2016(PCANTP_HANDLE_USBBUS1, PCANTP_BAUDRATE_500K);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox::Show("PCAN-USB (Ch-1) was initialized", "Success");
else
       MessageBox::Show("Initialization failed", "Error");
```

Visual Basic

```
Dim result As cantp_status

' The Plug and Play channel USB1 is initialized.
```

```
result = CanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, False) Then
    MessageBox.Show("PCAN-USB (Ch-1) was initialized", "Success")
Else
    MessageBox.Show("Initialization failed", "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The Plug and Play channel USB1 is initialized.
  result := TCanTpApi.Initialize_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    cantp_baudrate.PCANTP_BAUDRATE_500K);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    MessageBox(0, 'PCAN-USB (Ch-1) was initialized', 'Success', MB_OK);
  end
  else
  begin
    MessageBox(0, 'Initialization failed', 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_StatusIsOk_2016 on page 237

See also: cantp_status on page 60

3.6.29 Read_2016

Reads a CANTP message from the receive queue of a PCANTP channel.

Overloads

	Method	Description
	Read_2016(cantp_handle, cantp_msg)	Reads a CANTP message from the receive queue of a PCANTP channel.
	Read_2016(cantp_handle, cantp_msg, cantp_timestamp)	Reads a CANTP message and its timestamp from the receive queue of a PCANTP channel.
	Read_2016(cantp_handle, cantp_msg, cantp_timestamp, cantp_msgtype)	Reads a CANTP message (filtered by type) from the receive queue of a PCANTP channel.

Plain function version: CANTP_Read_2016 on page 243

3.6.30 Read_2016(cantp_handle, cantp_msg)

Reads a CANTP message from the receive queue of a PCANTP channel.

Syntax

Pascal OO

```
class function Read_2016(
    channel: cantp_handle;
    var msg_buffer: cantp_msg
): cantp_status; overload;
```

C#

```
public static cantp_status Read_2016(
    cantp_handle channel,
    out cantp_msg msg_buffer);
```

C++ / CLR

```
static cantp_status Read_2016(
       cantp_handle channel,
       cantp_msg %msg_buffer);
```

Visual Basic

```
Public Shared Function Read_2016(
    ByVal channel As cantp_handle,
    ByRef msg_buffer As cantp_msg) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
msg_buffer	A cantp_msg buffer to store the CANTP message. (see cantp_msg on page 28)

Returns

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

PCANTP_STATUS_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.

Remarks

- In addition to checking cantp_status code, the cantp_netstatus field should be checked as it contains the network status of the message (see cantp_msg on page 28, cantp_msgdata on page 22 and cantp_netstatus on page 53).
- In case of ISOTP message, the message type contained in the message cantp_netaddrinfo should be checked too as it indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic, and a pending message flag) (see cantp_msg on page 28, cantp_msgdata_isotp on page 26 and cantp_isotp_msgtype on page 90).
- The message structure is automatically allocated and initialized in Read_2016 method. So once the message processed, the structure must be uninitialized (see MsgDataFree_2016 on page 196).

Example

The following example shows the use of the method Read_2016 on the channel PCANTP_HANDLE_USBBUS1. Depending on the result, a message will be shown to the user. This example is basic, the proper way to handle message reception is Using Events (see on page 259).

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

C#

```
cantp_status result;
cantp_msg msg = new cantp_msg();
```

```
bool bStop = false;

do
{
    // Reads the first message in the queue.
    result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out msg);
    if (CanTpApi.StatusIsOk_2016(result))
    {
        // Processes the received message.
        MessageBox.Show("A message was received", "Success");

        // ProcessMessage(msg);

        // Free allocated memory
        CanTpApi.MsgDataFree_2016(ref msg);
    }
    else
    {
        // An error occurred.
        MessageBox.Show("An error ocured", "Error");
        // Here can be decided if the loop has to be terminated.
        // bStop = HandleReadError(result);
    }
} while (!bStop);
```

C++ / CLR

```
cantp_status result;
cantp_msg msg = {};
bool bStop = false;

do
{
       // Reads the first message in the queue.
       result = CanTpApi_2016::Read_2016(PCANTP_HANDLE_USBBUS1, msg);
       if (CanTpApi_2016::StatusIsOk_2016(result))
       {
              // Processes the received message.
              MessageBox::Show("A message was received", "Success");

              // ProcessMessage(msg);

              // Free allocated memory
              CanTpApi_2016::MsgDataFree_2016(msg);
       }
       else
       {
              // An error occurred.
              MessageBox::Show("An error ocured", "Error");
              // Here can be decided if the loop has to be terminated.
              // bStop = HandleReadError(result);
       }
} while (!bStop);
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg
Dim bStop As Boolean = False

Do
    ' Reads the first message in the queue.
```

```
    result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg)
    If CanTpApi.StatusIsOk_2016(result) Then
        ' Processes the received message.
        MessageBox.Show("A message was received", "Success")

        ' ProcessMessage(msg)

        ' Free allocated memory
        CanTpApi.MsgDataFree_2016(msg)
    Else
        ' An error occurred.
        MessageBox.Show("An error ocured", "Error")
        ' Here can be decided if the loop has to be terminated.
        ' bStop = HandleReadError(result)
    End If
Loop While Not bStop
```

Pascal OO

```
var
  result: cantp_status;
  msg: cantp_msg;
  bStop: boolean;
begin
  bStop := false;

  repeat

    // Reads the first message in the queue.
    result := TCanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg);
    if TCanTpApi.StatusIsOk_2016(result) then
    begin
      // Processes the received message.
      MessageBox(0, 'A message was received', 'Success', MB_OK);

      // ProcessMessage(msg);

      // Free allocated memory
      TCanTpApi.MsgDataFree_2016(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;
end;
```

Plain function version: CANTP_Read_2016 on page 243

See also: cantp_msg on page 28, Write_2016 on page 170, getData_2016 on page 212

3.6.31 Read_2016(cantp_handle, cantp_msg, cantp_timestamp)

Reads a CANTP message and its timestamp from the receive queue of a PCANTP channel.

Syntax

Pascal OO

```
class function Read_2016(
    channel: cantp_handle;
    var msg_buffer: cantp_msg;
    timestamp_buffer: Pcantp_timestamp
): cantp_status; overload;
```

C#

```
public static cantp_status Read_2016(
    cantp_handle channel,
    out cantp_msg msg_buffer,
    out cantp_timestamp timestamp_buffer);
```

C++/CLR

```
static cantp_status Read_2016(
       cantp_handle channel,
       cantp_msg %msg_buffer,
       cantp_timestamp %timestamp_buffer);
```

Visual Basic

```
Public Shared Function Read_2016(
    ByVal channel As cantp_handle,
    ByRef msg_buffer As cantp_msg,
    ByRef timestamp_buffer As UInt64) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
msg_buffer	A cantp_msg buffer to store the CANTP message. (see cantp_msg on page 28)
timestamp_buffer	A cantp_timestamp structure buffer to get the reception time of the message. If this value is not desired, this parameter should be passed as NULL. (see cantp_timestamp on page 36)

Returns

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

PCANTP_STATUS_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.

Remarks

- In addition to checking cantp_status code, the cantp_netstatus field should be checked as it contains the network status of the message (see cantp_msg on page 28, cantp_msgdata on page 22 and cantp_netstatus on page 53).
- In case of ISOTP message, the message type contained in the message cantp_netaddrinfo should be checked too as it indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic, and a pending message flag) (see cantp_msg on page 28, cantp_msgdata_isotp on page 26 and cantp_isotp_msgtype on page 90).
- Specifying the value of “NULL” for the parameter timestamp_buffer causes reading a message without timestamp, when the reception time is not desired.
- The message structure is automatically allocated and initialized in Read_2016 method. So once the message processed, the structure must be uninitialized (see MsgDataFree_2016 on page 196).

Example

The following example shows the use of the method Read_2016 on the channel PCANTP_HANDLE_USBBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;
cantp_msg msg = new cantp_msg();
cantp_timestamp ts;
bool bStop = false;

do
{
    // Reads the first message in the queue.
    result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out msg, out ts);
    if (CanTpApi.StatusIsOk_2016(result))
    {
        // Processes the received message.
        MessageBox.Show("A message was received", "Success");

        // ProcessMessage(msg);

        // Free allocated memory
        CanTpApi.MsgDataFree_2016(ref msg);
    }
    else
    {
        // An error occurred.
        MessageBox.Show("An error ocured", "Error");
        // Here can be decided if the loop has to be terminated.
        // bStop = HandleReadError(result);
    }
} while (!bStop);
```

C++/CLR

```
cantp_status result;
cantp_msg msg = {};
cantp_timestamp ts;
bool bStop = false;

do
{
       // Reads the first message in the queue.
       result = CanTpApi_2016::Read_2016(PCANTP_HANDLE_USBBUS1, msg, ts);
       if (CanTpApi_2016::StatusIsOk_2016(result))
       {
              // Processes the received message.
              MessageBox::Show("A message was received", "Success");

              // ProcessMessage(msg);

              // Free allocated memory
              CanTpApi_2016::MsgDataFree_2016(msg);
       }
       else
       {
              // An error occurred.
              MessageBox::Show("An error ocured", "Error");
```

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

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg
Dim ts As cantp_timestamp
Dim bStop As Boolean = False

Do
    ' Reads the first message in the queue.
    result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg, ts)
    If CanTpApi.StatusIsOk_2016(result) Then
        ' Processes the received message.
        MessageBox.Show("A message was received", "Success")

        ' ProcessMessage(msg)

        ' Free allocated memory
        CanTpApi.MsgDataFree_2016(msg)
    Else
        ' An error occurred.
        MessageBox.Show("An error ocured", "Error")
        ' Here can be decided if the loop has to be terminated.
        ' bStop = HandleReadError(result)
    End If
Loop While Not bStop
```

Pascal OO

```
var
  result: cantp_status;
  msg: cantp_msg;
  bStop: boolean;
  ts: cantp_timestamp;
begin
  bStop := false;
  repeat
    // Reads the first message in the queue.
    result := TCanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg, @ts);
    if TCanTpApi.StatusIsOk_2016(result) then
    begin
      // Processes the received message.
      MessageBox(0, 'A message was received', 'Success', MB_OK);

      // ProcessMessage(msg);

      // Free allocated memory
      TCanTpApi.MsgDataFree_2016(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;
end;
```

Plain function version: CANTP_Read_2016 on page 243

See also: cantp_msg on page 28, Write_2016 on page 170, getData_2016 on page 212

3.6.32 Read_2016(cantp_handle, cantp_msg, cantp_timestamp, cantp_msgtype)

Reads a CANTP message of a specific type from the receive queue of a PCANTP channel.

Syntax

Pascal OO

```
class function Read_2016(
    channel: cantp_handle;
    var msg_buffer: cantp_msg;
    timestamp_buffer: Pcantp_timestamp;
    msg_type: cantp_msgtype
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Read_2016")]
private static extern cantp_status Read_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    out cantp_msg msg_buffer,
    IntPtr timestamp_buffer,
    [MarshalAs(UnmanagedType.U4)]
    cantp_msgtype msg_type);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Read_2016")]
       static cantp_status Read_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       cantp_msg %msg_buffer,
       cantp_timestamp %timestamp_buffer,
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgtype msg_type);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Read_2016")>
Public Shared Function Read_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByRef msg_buffer As cantp_msg,
    ByRef timestamp_buffer As cantp_timestamp,
    <MarshalAs(UnmanagedType.U4)>
    ByVal msg_type As cantp_msgtype) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
msg_buffer	A cantp_msg buffer to store the CANTP message. (see cantp_msg on page 28)
timestamp_buffer	A cantp_timestamp structure buffer to get the reception time of the message. If this value is not desired, this parameter should be passed as NULL. (see cantp_timestamp on page 36)
msg_type	A cantp_msgtype structure buffer to filter the message to read. By default, accept any message type. (see cantp_msgtype on page 85)

Returns

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

PCANTP_STATUS_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.

Remarks

- In addition to checking cantp_status code, the cantp_netstatus field should be checked as it contains the network status of the message (see cantp_msg on page 28, cantp_msgdata on page 22 and cantp_netstatus on page 53).
- In case of ISOTP message, the message type contained in the message cantp_netaddrinfo should be checked too as it indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic, and a pending message flag) (see cantp_msg on page 28, cantp_msgdata_isotp on page 26 and cantp_isotp_msgtype on page 90).
- Specifying the value of “NULL” for the parameter timestamp_buffer causes reading a message without timestamp, when the reception time is not desired.
- The message structure is automatically allocated and initialized in Read_2016 method. So once the message processed, the structure must be uninitialized (see MsgDataFree_2016 on page 196).

Example

The following example shows the use of the method Read_2016 on the channel PCANTP_HANDLE_USBBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;
cantp_msg msg = new cantp_msg();
cantp_timestamp ts;
bool bStop = false;

do
{
    // Reads the first message in the queue.
    result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out msg, out ts,
        cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
    if (CanTpApi.StatusIsOk_2016(result))
    {
        // Processes the received message.
        MessageBox.Show("A message was received", "Success");

        // ProcessMessage(msg);
```

```
        // Free allocated memory
        CanTpApi.MsgDataFree_2016(ref msg);
    }
    else
    {
        // An error occurred.
        MessageBox.Show("An error ocured", "Error");
        // Here can be decided if the loop has to be terminated.
        // bStop = HandleReadError(result);
    }
} while (!bStop);
```

C++ / CLR

```
cantp_status result;
cantp_msg msg = {};
cantp_timestamp ts;
bool bStop = false;

do
{
       // Reads the first message in the queue.
       result = CanTpApi_2016::Read_2016(PCANTP_HANDLE_USBBUS1, msg, ts, PCANTP_MSGTYPE_ISOTP);
       if (CanTpApi_2016::StatusIsOk_2016(result))
       {
              // Processes the received message.
              MessageBox::Show("A message was received", "Success");

              // ProcessMessage(msg);

              // Free allocated memory
              CanTpApi_2016::MsgDataFree_2016(msg);
       }
       else
       {
              // An error occurred.
              MessageBox::Show("An error ocured", "Error");
              // Here can be decided if the loop has to be terminated.
              // bStop = HandleReadError(result);
       }
} while (!bStop);
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg
Dim ts As cantp_timestamp
Dim bStop As Boolean = False

Do
    ' Reads the first message in the queue.
    result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg, ts,
        cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
    If CanTpApi.StatusIsOk_2016(result) Then
        ' Processes the received message.
        MessageBox.Show("A message was received", "Success")

        ' ProcessMessage(msg)

        ' Free allocated memory
        CanTpApi.MsgDataFree_2016(msg)
    Else
```

```
        ' An error occurred.
        MessageBox.Show("An error ocured", "Error")
        ' Here can be decided if the loop has to be terminated.
        ' bStop = HandleReadError(result)
    End If
Loop While Not bStop
```

Pascal OO

```
var
  result: cantp_status;
  msg: cantp_msg;
  ts: cantp_timestamp;
  bStop: bool;
begin
  bStop := false;

  repeat
    // Reads the first message in the queue.
    result := TCanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg, @ts,
      cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
    if TCanTpApi.StatusIsOk_2016(result) then
    begin
      // Processes the received message.
      MessageBox(0, 'A message was received', 'Success', MB_OK);

      // ProcessMessage(msg);

      // Free allocated memory
      TCanTpApi.MsgDataFree_2016(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;
end;
```

Plain function version: CANTP_Read_2016 on page 243

See also: cantp_msg on page 28, Write_2016 below, getData_2016 on page 212

3.6.33 Write_2016

Transmits a CANTP message.

Syntax

Pascal OO

```
class function Write_2016(
    channel: cantp_handle;
    var msg_buffer: cantp_msg
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Write_2016")]
public static extern cantp_status Write_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel,
    ref cantp_msg msg_buffer);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Write_2016")]
static cantp_status Write_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel,
       cantp_msg %msg_buffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Write_2016")>
Public Shared Function Write_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle,
    ByRef msg_buffer As cantp_msg) As cantp_status
End Function
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
msg_buffer	A cantp_msg buffer containing the CANTP message to be sent (see cantp_msg on page 28).

Returns

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

PCANTP_STATUS_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_STATUS_PARAM_INVALID_VALUE	The message is not a valid message.
PCANTP_STATUS_PARAM_INVALID_TYPE	The message type is not valid.
PCANTP_STATUS_MAPPING_NOT_INITIALIZED	The mapping is unknown.

Remarks

The Write_2016 method does 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 (see cantp_netstatus on page 53).

Example

The following example shows the use of the method Write_2016 on the channel PCANTP_HANDLE_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, request_mapping was configured (see AddMapping_2016 on page 116) and receive_event is set (see Using Events on page 259).

C#

```
cantp_status result;
cantp_msg request_msg = new cantp_msg();
cantp_msg loopback_msg = new cantp_msg();
bool wait_result;

// Allocate message structure
result = CanTpApi.MsgDataAlloc_2016(out request_msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    MessageBox.Show("Message allocation error: " + result, "Error");

// Prepare an ISO-TP message containing 3 bytes of raw data.
result = CanTpApi.MsgDataInit_2016(out request_msg, request_mapping.can_id,
    request_mapping.can_msgtype, 3, (String)null, ref request_mapping.netaddrinfo);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
    MessageBox.Show("Message initialization error: " + result, "Error");

// The message is sent using the PCAN-USB.
result = CanTpApi.Write_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, ref request_msg);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
{
    // Read the transmission confirmation.
    wait_result = receive_event.WaitOne(5000);
    if (wait_result)
    {
        result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out loopback_msg);
        if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK))
            MessageBox.Show("Read=" + result + ", type=" + loopback_msg.type + ", netstatus=" +
                loopback_msg.Msgdata_any_Copy.netstatus, "Read");
        else
            MessageBox.Show("Read error: " + result, "Error");
    }
}
else
{
    MessageBox.Show("Write error: " + result, "Error");
}
```

C++ / CLR

```
cantp_status result;
cantp_msg request_msg = {};
cantp_msg loopback_msg = {};
bool wait_result;

// Allocate message structure
result = CanTpApi_2016::MsgDataAlloc_2016(request_msg, PCANTP_MSGTYPE_ISOTP);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox::Show(String::Format("Message allocation error: ", (int)result), "Error");

// Prepare an ISO-TP message containing 3 bytes of raw data.
result = CanTpApi_2016::MsgDataInit_2016(request_msg, request_mapping.can_id,
       request_mapping.can_msgtype, 3, (String^)nullptr, request_mapping.netaddrinfo);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox::Show(String::Format("Message initialization error: {0}",
              (int)result), "Error");

// The message is sent using the PCAN-USB.
result = CanTpApi_2016::Write_2016(PCANTP_HANDLE_USBBUS1, request_msg);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
{
```

```
        // Read the transmission confirmation.
        wait_result = receive_event.WaitOne(5000);
        if (wait_result)
        {
                result = CanTpApi_2016::Read_2016(PCANTP_HANDLE_USBBUS1, loopback_msg);
                if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK))
                        MessageBox::Show(String::Format("Read={0}, type={1}, netstatus={2}",
                                (int)result, (int)loopback_msg.type,
                                (int)loopback_msg.msgdata.any->netstatus), "Read");
                else
                        MessageBox::Show(String::Format("Read error: {0}", (int)result), "Error");
        }
}
else
{
        MessageBox::Show(String::Format("Write error: ", (int)result), "Error");
}
```

Visual Basic

```
Dim result As cantp_status
Dim request_msg As cantp_msg
Dim loopback_msg As cantp_msg
Dim wait_result As Boolean

' Allocate message structure
result = CanTpApi.MsgDataAlloc_2016(request_msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Message allocation error: " + result.ToString(), "Error")
End If

' Prepare an ISO-TP message containing 3 bytes of raw data.
Dim empty_data As String = Nothing
result = CanTpApi.MsgDataInit_2016(request_msg, request_mapping.can_id,
    request_mapping.can_msgtype, 3, empty_data, request_mapping.netaddrinfo)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
End If

' The message is sent using the PCAN-USB.
result = CanTpApi.Write_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, request_msg)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Read the transmission confirmation.
    wait_result = receive_event.WaitOne(5000)
    If wait_result Then
        result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, loopback_msg)
        If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
            MessageBox.Show("Read=" + result.ToString() + ", type=" +
                loopback_msg.type.ToString() + ", netstatus=" +
                loopback_msg.Msgdata_any_Copy.netstatus.ToString(), "Read")
        Else
            MessageBox.Show("Read error: " + result.ToString(), "Error")
        End If
    End If
Else
    MessageBox.Show("Write error: " + result.ToString(), "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  request_msg: cantp_msg;
  loopback_msg: cantp_msg;
  wait_result: UInt32;
begin

  // Allocate message structure
  result := TCanTpApi.MsgDataAlloc_2016(request_msg,
    cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    MessageBox(0, PWideChar(format('Message allocation error: %d',
      [Integer(result)])), 'Error', MB_OK);
  end;

  // Prepare an ISO-TP message containing 3 bytes of raw data.
  result := TCanTpApi.MsgDataInit_2016(request_msg, request_mapping.can_id,
    request_mapping.can_msgtype, 3, PAnsiChar(Nil),
    @request_mapping.netaddrinfo);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin
    MessageBox(0, PWideChar(format('Message initialization error: %d',
      [Integer(result)])), 'Error', MB_OK);
  end;

  // The message is sent using the PCAN-USB.
  result := TCanTpApi.Write_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
    request_msg);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
  begin

    // Read the transmission confirmation.
    wait_result := WaitForSingleObject(receive_event, 5000);
    if wait_result = WAIT_OBJECT_0 then
    begin
      result := TCanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1,
        loopback_msg);
      if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) then
      begin
        MessageBox(0, PWideChar(format('Read=%d, type=%d, netstatus=%d',
          [Integer(result), Integer(loopback_msg.typem),
          Integer(loopback_msg.msgdata_any.netstatus)])), 'Read', MB_OK);
      end
      else
      begin
        MessageBox(0, PWideChar(format('Read error: %d', [Integer(result)])),
          'Error', MB_OK);
      end;
    end;
  end
  else
  begin
    MessageBox(0, PWideChar(format('Write error: %d', [Integer(result)])),
      'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_Write_2016 on page 245

See also: cantp_msg on page 28, Read_2016 on page 160

More examples: see API examples folder and iisotp_read_write example.

3.6.34 Reset_2016

Resets the receive and transmit queues of a PCANTP channel.

Syntax

Pascal OO

```
class function Reset_2016(
    channel: cantp_handle
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Reset_2016")]
public static extern cantp_status Reset_2016(
    [MarshalAs(UnmanagedType.U4)]
    cantp_handle channel);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_Reset_2016")]
static cantp_status Reset_2016(
       [MarshalAs(UnmanagedType::U4)]
       cantp_handle channel);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_Reset_2016")>
Public Shared Function Reset_2016(
    <MarshalAs(UnmanagedType.U4)>
    ByVal channel As cantp_handle) As cantp_status
End Function
```

Parameters

Parameter	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).

Returns

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

PCANTP_STATUS_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 method Reset_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;

// The PCI channel 1 is reset.
result = CanTpApi.Reset_2016(cantp_handle.PCANTP_HANDLE_PCIBUS1);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    MessageBox.Show("An error occured", "Error");
else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset", "Success");
```

C++ / CLR

```
cantp_status result;

// The PCI channel 1 is reset.
result = CanTpApi_2016::Reset_2016(PCANTP_HANDLE_PCIBUS1);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox::Show("An error occured", "Error");
else
       MessageBox::Show("PCAN-PCI (Ch-1) was reset", "Success");
```

Pascal OO

```
var
  result: cantp_status;
begin

  // The PCI channel 1 is reset.
  result := TCanTpApi.Reset_2016(cantp_handle.PCANTP_HANDLE_PCIBUS1);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    MessageBox(0, 'An error occured', 'Error', MB_OK);
  end
  else
  begin
    MessageBox(0, 'PCAN-PCI (Ch-1) was reset', 'Success', MB_OK);
  end;
end;
```

Visual Basic

```
Dim result As cantp_status

' The PCI channel 1 is reset.
result = CanTpApi.Reset_2016(cantp_handle.PCANTP_HANDLE_PCIBUS1)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("An error occured", "Error")
Else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset", "Success")
End If
```

Plain function version: CANTP_Reset_2016 on page 246

See also: Uninitialize_2016 on page 107

3.6.35 MsgEqual_2016

Checks if two CANTP messages are equal.

Syntax

Pascal OO

```
class function MsgEqual_2016(
    const msg_buffer1: Pcantp_msg;
    const msg_buffer2: Pcantp_msg;
    ignoreSelfReceiveFlag: Boolean
): boolean;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgEqual_2016")]
[return: MarshalAs(UnmanagedType.I1)]
public static extern bool MsgEqual_2016(
    [In]ref cantp_msg msg_buffer1,
    [In]ref cantp_msg msg_buffer2,
    [MarshalAs(UnmanagedType.I1)]
    bool ignoreSelfReceiveFlag);
```

C++ / CLR

```
static bool MsgEqual_2016(
       cantp_msg %msg_buffer1,
       cantp_msg %msg_buffer2,
       bool ignoreSelfReceiveFlag);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgEqual_2016")>
Public Shared Function MsgEqual_2016(
    ByRef msg_buffer1 As cantp_msg,
    ByRef msg_buffer2 As cantp_msg,
    <MarshalAs(UnmanagedType.I1)>
    ByVal ignoreSelfReceiveFlag As Boolean) As Byte
End Function
```

Parameters

Parameters	Description
msg_buffer1	A cantp_msg structure buffer (see cantp_msg on page 28).
msg_buffer2	Another cantp_msg structure buffer to compare with first parameter (see cantp_msg on page 28).
ignoreSelfReceiveFlag	States if comparison should ignore loopback flag (i.e if true the method will return true when comparing a request and its loopback confirmation).

Returns

The return value is a boolean. It is true if the messages are the same or false if they are not.

Remarks

If one message is the indication of an incoming/outgoing ISO-TP message, the actual data-content will not be compared. In that case the method checks if the messages' network address information matches.

Example

The following example shows the use of the method MsgEqual_2016. It allocates and initializes a first message structure, copies it in a second structure then checks that the two structures are the same.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg msg_1;
cantp_msg msg_2;

// Initialize the first message
result = CanTpApi.MsgDataAlloc_2016(out msg_1, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    result = CanTpApi.MsgDataInit_2016(out msg_1, request_mapping.can_id,
        request_mapping.can_msgtype, 4, "PEAK", ref request_mapping.netaddrinfo);
    if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {

        // Copy msg_1 in msg_2
        result = CanTpApi.MsgCopy_2016(out msg_2, ref msg_1);
        if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
        {

            // Compare msg_1 and msg_2, should be the same
            if (CanTpApi.MsgEqual_2016(ref msg_1, ref msg_2, false))
                MessageBox.Show("msg_1 and msg_2 are the same!", "Success");
            else
                MessageBox.Show("msg_1 and msg_2 are different!", "Error");
        }
    }
}
```

C++ / CLR

```
cantp_status result;
cantp_msg msg_1;
cantp_msg msg_2;

// Initialize the first message
result = CanTpApi_2016::MsgDataAlloc_2016(msg_1, PCANTP_MSGTYPE_ISOTP);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
{
       result = CanTpApi_2016::MsgDataInit_2016(msg_1, request_mapping.can_id,
              request_mapping.can_msgtype, 4, "PEAK", request_mapping.netaddrinfo);
       if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       {

              // Copy msg_1 in msg_2
              result = CanTpApi_2016::MsgCopy_2016(msg_2, msg_1);
              if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
              {

                     // Compare msg_1 and msg_2, should be the same
                     if (CanTpApi_2016::MsgEqual_2016(msg_1, msg_2, false))
                            MessageBox::Show("msg_1 and msg_2 are the same!", "Success");
                     else
                            MessageBox::Show("msg_1 and msg_2 are different!", "Error");
              }
       }
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg_1 As cantp_msg
Dim msg_2 As cantp_msg

' Initialize the first message
result = CanTpApi.MsgDataAlloc_2016(msg_1, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    result = CanTpApi.MsgDataInit_2016(msg_1, request_mapping.can_id,
        request_mapping.can_msgtype, 4, "PEAK", request_mapping.netaddrinfo)
    If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

        ' Copy msg_1 in msg_2
        result = CanTpApi.MsgCopy_2016(msg_2, msg_1)
        If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

            ' Compare msg_1 and msg_2, should be the same
            If CanTpApi.MsgEqual_2016(msg_1, msg_2, False) Then
                MessageBox.Show("msg_1 and msg_2 are the same!", "Success")
            Else
                MessageBox.Show("msg_1 and msg_2 are different!", "Error")
            End If
        End If
    End If
End If
```

Pascal OO

```
var
  result: cantp_status;
  msg_1: cantp_msg;
  msg_2: cantp_msg;
begin

  // Initialize the first message
  result := TCanTpApi.MsgDataAlloc_2016(msg_1,
    cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin

    result := TCanTpApi.MsgDataInit_2016(msg_1, request_mapping.can_id,
      request_mapping.can_msgtype, 4, 'PEAK', @request_mapping.netaddrinfo);
    if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
    then
    begin

      // Copy msg_1 in msg_2
      result := TCanTpApi.MsgCopy_2016(msg_2, @msg_1);
      if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
      then
      begin

        // Compare msg_1 and msg_2, should be the same
        if TCanTpApi.MsgEqual_2016(@msg_1, @msg_2, false) then
        begin
          MessageBox(0, 'msg_1 and msg_2 are the same!', 'Success', MB_OK);
        end
        else
        begin
          MessageBox(0, 'msg_1 and msg_2 are different!', 'Error', MB_OK);
```

```
        end;
      end;
    end;
  end;
end;
```

Plain function version: CANTP_MsgEqual_2016 on page 250

See also: cantp_msg on page 28, MsgCopy_2016 below

3.6.36 MsgCopy_2016

Copies a CANTP message to another buffer.

Syntax

Pascal OO

```
class function MsgCopy_2016(
    var msg_buffer_dst: cantp_msg;
    const msg_buffer_src: Pcantp_msg
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgCopy_2016")]
public static extern cantp_status MsgCopy_2016(
    out cantp_msg msg_buffer_dst,
    [In]ref cantp_msg msg_buffer_src);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgCopy_2016")]
static cantp_status MsgCopy_2016(
       cantp_msg %msg_buffer_dst,
       [In] cantp_msg %msg_buffer_src);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgCopy_2016")>
Public Shared Function MsgCopy_2016(
    ByRef msg_buffer_dst As cantp_msg,
    ByRef msg_buffer_src As cantp_msg) As cantp_status
End Function
```

Parameters

Parameters	Description
msg_buffer_dst	A cantp_msg structure buffer to store the copied message (see cantp_msg on page 28).
msg_buffer_src	The cantp_msg structure buffer used as the source (see cantp_msg on page 28).

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	One of the given messages is not valid.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory during the copy.

Example

The following example shows the use of the method MsgCopy_2016. It allocates and initializes a first message structure, copies it in a second structure then checks that the two structures are the same.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg msg_1;
cantp_msg msg_2;

// Initialize the first message
result = CanTpApi.MsgDataAlloc_2016(out msg_1, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    result = CanTpApi.MsgDataInit_2016(out msg_1, request_mapping.can_id,
        request_mapping.can_msgtype, 4, "PEAK", ref request_mapping.netaddrinfo);
    if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {

        // Copy msg_1 in msg_2
        result = CanTpApi.MsgCopy_2016(out msg_2, ref msg_1);
        if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
        {

            // Compare msg_1 and msg_2, should be the same
            if (CanTpApi.MsgEqual_2016(ref msg_1, ref msg_2, false))
                MessageBox.Show("msg_1 and msg_2 are the same!", "Success");
            else
                MessageBox.Show("msg_1 and msg_2 are different!", "Error");
        }
    }
}
```

C++ / CLR

```
cantp_status result;
cantp_msg msg_1;
cantp_msg msg_2;

// Initialize the first message
result = CanTpApi_2016::MsgDataAlloc_2016(msg_1, PCANTP_MSGTYPE_ISOTP);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
{
       result = CanTpApi_2016::MsgDataInit_2016(msg_1, request_mapping.can_id,
              request_mapping.can_msgtype, 4, "PEAK", request_mapping.netaddrinfo);
       if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       {

              // Copy msg_1 in msg_2
              result = CanTpApi_2016::MsgCopy_2016(msg_2, msg_1);
              if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
              {

                     // Compare msg_1 and msg_2, should be the same
                     if (CanTpApi_2016::MsgEqual_2016(msg_1, msg_2, false))
                            MessageBox::Show("msg_1 and msg_2 are the same!", "Success");
                     else
                            MessageBox::Show("msg_1 and msg_2 are different!", "Error");
              }
```

```
        }
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg_1 As cantp_msg
Dim msg_2 As cantp_msg

' Initialize the first message
result = CanTpApi.MsgDataAlloc_2016(msg_1, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)) Then
    result = CanTpApi.MsgDataInit_2016(msg_1, request_mapping.can_id,
        request_mapping.can_msgtype, 4, "PEAK", request_mapping.netaddrinfo)
    If (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)) Then
        ' Copy msg_1 in msg_2
        result = CanTpApi.MsgCopy_2016(msg_2, msg_1)
        If (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)) Then

            ' Compare msg_1 and msg_2, should be the same
            If (CanTpApi.MsgEqual_2016(msg_1, msg_2, False)) Then
                MessageBox.Show("msg_1 and msg_2 are the same!", "Success")
            Else
                MessageBox.Show("msg_1 and msg_2 are different!", "Error")
            End If
        End If
    End If
End If
```

Pascal OO

```
var
  result: cantp_status;
  msg_1: cantp_msg;
  msg_2: cantp_msg;
begin

  // Initialize the first message
  result := TCanTpApi.MsgDataAlloc_2016(msg_1,
    cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin

    result := TCanTpApi.MsgDataInit_2016(msg_1, request_mapping.can_id,
      request_mapping.can_msgtype, 4, 'PEAK', @request_mapping.netaddrinfo);
    if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
    then
    begin

      // Copy msg_1 in msg_2
      result := TCanTpApi.MsgCopy_2016(msg_2, @msg_1);
      if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
      then
      begin

        // Compare msg_1 and msg_2, should be the same
        if TCanTpApi.MsgEqual_2016(@msg_1, @msg_2, false) then
        begin
          MessageBox(0, 'msg_1 and msg_2 are the same!', 'Success', MB_OK);
        end
        else
```

```
        begin
          MessageBox(0, 'msg_1 and msg_2 are different!', 'Error', MB_OK);
        end;
      end;
    end;
  end;
end;
```

Plain function version: CANTP_MsgCopy_2016 on page 251

See also: cantp_msg on page 28, MsgEqual_2016 on page 177

3.6.37 MsgDlcToLength_2016

Converts a CAN DLC to its corresponding length.

Syntax

Pascal OO

```
class function MsgDlcToLength_2016(
    const dlc: Byte
): UInt32;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDlcToLength_2016")]
public static extern UInt32 MsgDlcToLength_2016(
    byte dlc);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDlcToLength_2016")]
       static UInt32 MsgDlcToLength_2016(
       Byte dlc);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgDlcToLength_2016")>
Public Shared Function MsgDlcToLength_2016(
    ByVal dlc As Byte) As UInt32
End Function
```

Parameters

Parameter	Description
dlc	The Data Length Code (DLC) to convert.

Returns

The corresponding length of the dlc parameter.

Example

The following example shows the use of the method MsgDlcToLength_2016.

C#

```
Byte dlc = 10;
UInt32 length;
```

```
length = CanTpApi.MsgDlcToLength_2016(dlc);
MessageBox.Show("For dlc=" + dlc + ", length=" + length, "Info");
```

C++ / CLR

```
Byte dlc = 10;
UInt32 length;
length = CanTpApi_2016::MsgDlcToLength_2016(dlc);
MessageBox::Show("For dlc=" + dlc + ", length=" + length, "Info");
```

Visual Basic

```
Dim dlc As Byte = 10
Dim length As UInt32
length = CanTpApi.MsgDlcToLength_2016(dlc)
MessageBox.Show("For dlc=" + dlc.ToString() + ", length=" + length.ToString(), "Info")
```

Pascal OO

```
var
  dlc: Byte;
  length: UInt32;
begin
  dlc := 10;
  length := TCanTpApi.MsgDlcToLength_2016(dlc);
  MessageBox(0, PWideChar(format('For dlc=%d, length=%d', [Integer(dlc),
    Integer(length)])), 'Info', MB_OK);
end;
```

Plain function version: CANTP_MsgDlcToLength_2016 on page 252

See also: MsgLengthToDlc_2016 below

3.6.38 MsgLengthToDlc_2016

Converts a data length to a corresponding CAN DLC.

Syntax

Pascal OO

```
class function MsgLengthToDlc_2016(
    const length: UInt32
): Byte;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgLengthToDlc_2016")]
public static extern byte MsgLengthToDlc_2016(
    UInt32 length);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgLengthToDlc_2016")]
static Byte MsgLengthToDlc_2016(
       UInt32 length);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgLengthToDlc_2016")>
Public Shared Function MsgLengthToDlc_2016(
```

```
    ByVal length As UInt32) As Byte
End Function
```

Parameters

Parameters	Description
length	The length to convert.

Returns

The smallest DLC that can hold the requested length (0x00-0x0F).

Remarks

The returned DLC can hold more data that the requested length.

Example

The following example shows the use of the method MsgLengthToDlc_2016.

C#

```
Byte dlc;
UInt32 length = 16;
dlc = CanTpApi.MsgLengthToDlc_2016(length);
MessageBox.Show("For length=" + length + ", dlc=" + dlc, "Info");
```

C++ / CLR

```
Byte dlc;
UInt32 length = 16;
dlc = CanTpApi_2016::MsgLengthToDlc_2016(length);
MessageBox::Show("For length=" + length + ", dlc=" + dlc, "Info");
```

Visual Basic

```
Dim dlc As Byte
Dim length As UInt32 = 16
dlc = CanTpApi.MsgLengthToDlc_2016(length)
MessageBox.Show("For length=" + length.ToString() + ", dlc=" + dlc.ToString(), "Info")
```

Pascal OO

```
var
  dlc: Byte;
  length: UInt32;
begin
  length := 16;
  dlc := TCanTpApi.MsgLengthToDlc_2016(length);
  MessageBox(0, PWideChar(format('For length=%d, dlc=%d', [Integer(length),
    Integer(dlc)])), 'Info', MB_OK);
end;
```

Plain function version: CANTP_MsgLengthToDlc_2016 on page 253

See also: MsgDlcToLength_2016 on page 183

3.6.39 MsgDataAlloc_2016

Allocates a CANTP message based on the given type.

Syntax

Pascal OO

```
class function MsgDataAlloc_2016(
    var msg_buffer: cantp_msg;
    msg_type: cantp_msgtype
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataAlloc_2016")]
public static extern cantp_status MsgDataAlloc_2016(
    out cantp_msg msg_buffer,
    [MarshalAs(UnmanagedType.U4)]
    cantp_msgtype type);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataAlloc_2016")]
static cantp_status MsgDataAlloc_2016(
       cantp_msg %msg_buffer,
       [MarshalAs(UnmanagedType::U4)]
       cantp_msgtype type);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgDataAlloc_2016")>
Public Shared Function MsgDataAlloc_2016(
    ByRef msg_buffer As cantp_msg,
    <MarshalAs(UnmanagedType.U4)>
    ByVal type As cantp_msgtype) As cantp_status
End Function
```

Parameters

Parameters	Description
msg_buffer	A cantp_msg structure buffer (see cantp_msg on page 28). It will be freed if required.
type	Type of the message to allocate (see cantp_msgtype on page 85).

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the given message is not valid.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory for the given message.

Remarks

In case of handling reception messages, it is not mandatory to allocate and initialize message. Indeed, the message allocation is automatically realized within the method Read_2016. Yet to prevent random memory artifacts, it is recommended to call the function with the type PCANTP_MSGTYPE_NONE: the function will make sure to zero-initialize the buffer.

Once allocated, a message should be initialized by calling the method MsgDataInit_2016. Then freed using the method MsgDataFree_2016.

Example

The following example shows the use of the method MsgDataAlloc_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;
cantp_msg msg;

// Allocate message structure
result = CanTpApi.MsgDataAlloc_2016(out msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    MessageBox.Show("Message allocation error: "+ result, "Error");
else
    MessageBox.Show("Message is allocated!", "Success");
```

C++ / CLR

```
cantp_status result;
cantp_msg msg;

// Allocate message structure
result = CanTpApi_2016::MsgDataAlloc_2016(msg, PCANTP_MSGTYPE_ISOTP);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox::Show(String::Format("Message allocation error: ", (int)result), "Error");
else
       MessageBox::Show("Message is allocated!", "Success");
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg

' Allocate message structure
result = CanTpApi.MsgDataAlloc_2016(msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Message allocation error: " + result.ToString(), "Error")
Else
    MessageBox.Show("Message is allocated!", "Success")
End If
```

Pascal OO

```
var
  result: cantp_status;
  msg: cantp_msg;
begin

  // Allocate message structure
  result := TCanTpApi.MsgDataAlloc_2016(msg,
    cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    MessageBox(0, PWideChar(format('Message allocation error: %d',
      [Integer(result)])), 'Error', MB_OK);
```

```
  end
  else
  begin
    MessageBox(0, 'Message is allocated!', 'Success', MB_OK);
  end;
end;
```

Plain function version: CANTP_MsgDataAlloc_2016 on page 247

See also: cantp_msg on page 28, MsgDataInit_2016 below, MsgDataFree_2016 on page 196

3.6.40 MsgDataInit_2016

Initializes an allocated CANTP message.

Overloads

	Method	Description
	MsgDataInit_2016(cantp_msg, UInt32, cantp_can_msgtype, UInt32, Byte[])	Initializes an allocated CANTP message from Byte array buffer without network address information (for non ISOTP messages).
	MsgDataInit_2016(cantp_msg, UInt32, cantp_can_msgtype, UInt32, Byte[], cantp_netaddrinfo)	Initializes an allocated CANTP message from Byte array buffer (only for ISOTP messages).

Plain function version: CANTP_MsgDataInit_2016 on page 248

3.6.41 MsgDataInit_2016(cantp_msg, UInt32, cantp_can_msgtype, UInt32, Byte[])

Initializes an allocated CANTP message from Byte array buffer without network address information (for non ISOTP messages).

Syntax

Pascal OO

```
class function MsgDataInit_2016(
    var msg_buffer: cantp_msg;
    can_id: UInt32;
    can_msgtype: cantp_can_msgtype;
    data_length: UInt32;
    const data: PByte
): cantp_status; overload;
```

C#

```
public static cantp_status MsgDataInit_2016(
    out cantp_msg msg_buffer,
    UInt32 can_id,
    cantp_can_msgtype can_msgtype,
    UInt32 data_length,
    Byte[] data);
```

C++ / CLR

```
static cantp_status MsgDataInit_2016(
       cantp_msg %msg_buffer,
       UInt32 can_id,
       cantp_can_msgtype can_msgtype,
```

```
        UInt32 data_length,
        array<Byte>^ data);
```

Visual Basic

```
Public Shared Function MsgDataInit_2016(
    ByRef msg_buffer As cantp_msg,
    ByVal can_id As UInt32,
    ByVal can_msgtype As cantp_can_msgtype,
    ByVal data_length As UInt32,
    ByVal data As Byte()) As cantp_status
End Function
```

Parameters

Parameters	Description
msg_buffer	An allocated cantp_msg structure buffer (see cantp_msg on page 28 and CANTP_MsgDataAlloc_2016 on page 247).
can_id	CAN identifier (ISO-TP message may ignore this parameter and use PCANTP_MAPPING_FLOW_CTRL_NONE (-1)).
can_msgtype	Combination of CAN message types (like "extended CAN ID", "FD", "RTR", etc. flags). See cantp_msgtype on page 85.
data_length	The length in bytes of the data.
data	A buffer to initialize the message's data with. If NULL, message's data is initialized with zeros.

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	One of the given parameters is not valid (null message, bad canid, incorrect length depending on the message type...).
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory during initialization.

Example

The following example shows the use of the method MsgDataInit_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg request_msg;

result = CanTpApi.MsgDataAlloc_2016(out request_msg, cantp_msgtype.PCANTP_MSGTYPE_CAN);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{

    // Initialize the allocated message with "PEAK" as data
    Byte[] data = new byte[4];
    data[0] = 0x50;
    data[1] = 0x45;
    data[2] = 0x41;
    data[3] = 0x4B;
    result = CanTpApi.MsgDataInit_2016(out request_msg, request_mapping.can_id,
        request_mapping.can_msgtype, 4, data);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
        MessageBox.Show("Message initialization error: " + result, "Error");
}
```

C++ / CLR

```
cantp_status result;
cantp_msg request_msg;

result = CanTpApi_2016::MsgDataAlloc_2016(request_msg, PCANTP_MSGTYPE_CAN);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
{

       // Initialize the allocated message with "PEAK" as data
       array<Byte>^ data = gcnew array<Byte>(4);
       data[0] = 0x50;
       data[1] = 0x45;
       data[2] = 0x41;
       data[3] = 0x4B;
       result = CanTpApi_2016::MsgDataInit_2016(request_msg, request_mapping.can_id,
              request_mapping.can_msgtype, 4, data);
       if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
              MessageBox::Show(String::Format("Message initialization error: ", (int)result),
                     "Error");
}
```

Visual Basic

```
Dim result As cantp_status
Dim request_msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(request_msg, cantp_msgtype.PCANTP_MSGTYPE_CAN)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "PEAK" as data
    Dim data(4) As Byte
    data(0) = &H50
    data(1) = &H45
    data(2) = &H41
    data(3) = &H4B
    result = CanTpApi.MsgDataInit_2016(request_msg, request_mapping.can_id,
        request_mapping.can_msgtype, 4, data)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    End If
End If
```

Pascal OO

```
var
  result: cantp_status;
  request_msg: cantp_msg;
  data: array [0 .. 4] of Byte;
begin
  result := TCanTpApi.MsgDataAlloc_2016(request_msg,
    cantp_msgtype.PCANTP_MSGTYPE_CAN);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin

    // Initialize the allocated message with 'PEAK' as data
    data[0] := $50;
    data[1] := $45;
    data[2] := $41;
    data[3] := $4B;
    result := TCanTpApi.MsgDataInit_2016(&request_msg, request_mapping.can_id,
```

```
      request_mapping.can_msgtype, 4, PByte(@data));
    if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK,
      false) then
    begin
      MessageBox(0, PWideChar(format('Message initialization error: %d',
        [Integer(result)])), 'Error', MB_OK);
    end;
  end;
end;
```

Plain function version: CANTP_MsgDataInit_2016 on page 248

See also: cantp_msg on page 28, MsgDataAlloc_2016 on page 186, MsgDataFree_2016 on page 196

3.6.42 MsgDataInit_2016(cantp_msg, UInt32, cantp_can_msgtype, UInt32, Byte[], cantp_netaddrinfo)

Initializes an allocated CANTP message from Byte array buffer (only for ISOTP messages).

Syntax

Pascal OO

```
class function MsgDataInit_2016(
    var msg_buffer: cantp_msg;
    can_id: UInt32;
    can_msgtype: cantp_can_msgtype;
    data_length: UInt32;
    const data: PByte;
    netaddrinfo: Pcantp_netaddrinfo
): cantp_status; overload;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataInit_2016")]
public static extern cantp_status MsgDataInit_2016(
    out cantp_msg msg_buffer,
    UInt32 can_id,
    [MarshalAs(UnmanagedType.U4)]
    cantp_can_msgtype can_msgtype,
    UInt32 data_length,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
    Byte[] data,
    ref cantp_netaddrinfo netaddrinfo);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataInit_2016")]
static cantp_status MsgDataInit_2016(
       cantp_msg %msg_buffer,
       UInt32 can_id,
       [MarshalAs(UnmanagedType::U4)]
       cantp_can_msgtype can_msgtype,
       UInt32 data_length,
       [MarshalAs(UnmanagedType::LPArray, SizeParamIndex = 3)]
       array<Byte>^ data,
       cantp_netaddrinfo %netaddrinfo);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgDataInit_2016")>
Public Shared Function MsgDataInit_2016(
    ByRef msg_buffer As cantp_msg,
    ByVal can_id As UInt32,
    <MarshalAs(UnmanagedType.U4)>
    ByVal can_msgtype As cantp_can_msgtype,
    ByVal data_length As UInt32,
    ByVal data As Byte(),
    ByRef netaddrinfo As cantp_netaddrinfo) As cantp_status
End Function
```

Parameters

Parameters	Description
msg_buffer	An allocated cantp_msg structure buffer (see cantp_msg on page 28 and CANTP_MsgDataAlloc_2016 on page 247).
can_id	CAN identifier (ISO-TP message may ignore this parameter and use PCANTP_MAPPING_FLOW_CTRL_NONE (-1)).
can_msgtype	Combination of CAN message types (like "extended CAN ID", "FD", "RTR", etc. flags). See cantp_msgtype on page 85.
data_length	The length in bytes of the data.
data	A buffer to initialize the message's data with. If NULL, message's data is initialized with zeros.
netaddrinfo	Network address information of the ISO-TP message (see cantp_netaddrinfo on page 19). Only valid with an ISO-TP message.

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	One of the given parameters is not valid (null message, bad canid, incorrect length depending on the message type...).
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory during initialization.

Example

The following example shows the use of the method MsgDataInit_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg request_msg;

result = CanTpApi.MsgDataAlloc_2016(out request_msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    // Initialize the allocated message with "PEAK" as data
    Byte[] data = new byte[4];
    data[0] = 0x50;
    data[1] = 0x45;
    data[2] = 0x41;
    data[3] = 0x4B;
    result = CanTpApi.MsgDataInit_2016(out request_msg, request_mapping.can_id,
        request_mapping.can_msgtype, 4, data, ref request_mapping.netaddrinfo);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
        MessageBox.Show("Message initialization error: " + result, "Error");
}
```

C++ / CLR

```
cantp_status result;
cantp_msg request_msg;

result = CanTpApi_2016::MsgDataAlloc_2016(request_msg, PCANTP_MSGTYPE_ISOTP);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
{
	// Initialize the allocated message with "PEAK" as data
	array<Byte>^ data = gcnew array<Byte>(4);
	data[0] = 0x50;
	data[1] = 0x45;
	data[2] = 0x41;
	data[3] = 0x4B;
	result = CanTpApi_2016::MsgDataInit_2016(request_msg, request_mapping.can_id,
		request_mapping.can_msgtype, 4, data, request_mapping.netaddrinfo);
	if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
		MessageBox::Show(String::Format("Message initialization error: ", (int)result),
			"Error");
}
```

Visual Basic

```
Dim result As cantp_status
Dim request_msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(request_msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "PEAK" as data
    Dim data(4) As Byte
    data(0) = &H50
    data(1) = &H45
    data(2) = &H41
    data(3) = &H4B
    result = CanTpApi.MsgDataInit_2016(request_msg, request_mapping.can_id,
        request_mapping.can_msgtype, 4, data, request_mapping.netaddrinfo)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    End If
End If
```

Pascal OO

```
var
  result: cantp_status;
  request_msg: cantp_msg;
  data: array [0 .. 4] of Byte;
begin

  result := TCanTpApi.MsgDataAlloc_2016(request_msg,
    cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin

    // Initialize the allocated message with 'PEAK' as data
    data[0] := $50;
    data[1] := $45;
    data[2] := $41;
    data[3] := $4B;
    result := TCanTpApi.MsgDataInit_2016(request_msg, request_mapping.can_id,
```

```
      request_mapping.can_msgtype, 4, PByte(@data),
      @request_mapping.netaddrinfo);
    if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK,
      false) then
    begin
      MessageBox(0, PWideChar(format('Message initialization error: %d',
        [Integer(result)])), 'Error', MB_OK);
    end;
  end;
end;
```

Plain function version: CANTP_MsgDataInit_2016 on page 248

See also: cantp_msg on page 28, MsgDataAlloc_2016 on page 186, MsgDataFree_2016 on page 196, setData_2016 on page 208

3.6.43 MsgDataInitOptions_2016

Initializes several options for the CANTP message that will override the channel's parameter(s).

Syntax

Pascal OO

```
class function MsgDataInitOptions_2016(
    var msg_buffer: cantp_msg;
    nb_options: UInt32
): cantp_status;
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataInitOptions_2016")]
static cantp_status MsgDataInitOptions_2016(
       cantp_msg %msg_buffer,
       UInt32 nb_options);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgDataInitOptions_2016")>
Public Shared Function MsgDataInitOptions_2016(
    ByRef msg_buffer As cantp_msg,
    ByVal nb_options As UInt32) As cantp_status
End Function
```

Parameters

Parameters	Description
msg_buffer	An allocated cantp_msg structure buffer (see cantp_msg on page 28).
nb_options	Number of options to initialize.

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	The given message is not a valid message.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory.

Example

The following example shows the use of the method MsgDataInitOptions_2016. It sets a new priority for an already initialized message.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
cantp_status result;

// Set priority
result = CanTpApi.MsgDataInitOptions_2016(out request_msg, 1);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    cantp_msgoption option;
    option.name = cantp_option.PCANTP_OPTION_J1939_PRIORITY;
    option.value = 0x02;
    if (CanTpApi.setOption_2016(ref request_msg, 0, ref option))
        MessageBox.Show("Set message priority.", "Info");
    else
        MessageBox.Show("Cannot set message priority.", "Error");
}
```

Note: The C# API provides the helper method setOption_2016 in order to set an option (see on page 218).

C++ / CLR

```
cantp_status result;

// Set priority
result = CanTpApi_2016::MsgDataInitOptions_2016(request_msg, 1);
if (CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
{
       request_msg.msgdata.any->options->buffer[0].name = PCANTP_OPTION_J1939_PRIORITY;
       request_msg.msgdata.any->options->buffer[0].value = 0x02;
       MessageBox::Show("Set message priority.", "Info");
}
```

Visual Basic

```
Dim result As cantp_status

' Set priority
result = CanTpApi.MsgDataInitOptions_2016(request_msg, 1)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    Dim new_option As cantp_msgoption
    new_option.name = cantp_option.PCANTP_OPTION_J1939_PRIORITY
    new_option.value = &H2
    If CanTpApi.setOption_2016(request_msg, 0, new_option) Then
        MessageBox.Show("Set message priority.", "Info")
    Else
        MessageBox.Show("Cannot set message priority.", "Error")
    End If
End If
```

Note: The VB API provides the helper method setOption_2016 in order to set an option (see on page 218).

Pascal OO

```
type
  Pcantp_msgoption = ^cantp_msgoption;
var
  result: cantp_status;
begin

  // Set priority
  result := TCanTpApi.MsgDataInitOptions_2016(request_msg, 1);
  if TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    optpointer := Pcantp_msgoption(request_msg.msgdata_any^.options^.buffer);
    optpointer^.name := cantp_option.PCANTP_OPTION_J1939_PRIORITY;
    optpointer^.value := $02;
    MessageBox(0, 'Set message priority.', 'Info', MB_OK);
  end;
end;
```

Plain function version: CANTP_MsgDataInitOptions_2016 on page 254

See also: cantp_msgoption on page 16, cantp_option on page 93

3.6.44 MsgDataFree_2016

Deallocates a CANTP message.

Syntax

Pascal OO

```
class function MsgDataFree_2016(
    var msg_buffer: cantp_msg
): cantp_status;
```

C#

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataFree_2016")]
public static extern cantp_status MsgDataFree_2016(
    ref cantp_msg msg_buffer);
```

C++ / CLR

```
[DllImport("PCAN-ISO-TP.dll", EntryPoint = "CANTP_MsgDataFree_2016")]
static cantp_status MsgDataFree_2016(
       cantp_msg %msg_buffer);
```

Visual Basic

```
<DllImport("PCAN-ISO-TP.dll", EntryPoint:="CANTP_MsgDataFree_2016")>
Public Shared Function MsgDataFree_2016(
    ByRef msg_buffer As cantp_msg) As cantp_status
End Function
```

Parameters

Parameter	Description
msg_buffer	An allocated cantp_msg structure (see cantp_msg on page 28 and MsgDataAlloc_2016 on page 186).

Returns

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

PCANTP_STATUS_CAUTION_BUFFER_IN_USE	The message structure is currently in use. It cannot be deleted.
PCANTP_STATUS_PARAM_INVALID_VALUE	The message is not valid.

Example

The following example shows the use of the method MsgDataFree_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C#

```
cantp_status result;
cantp_msg msg;

// Allocate message structure
result = CanTpApi.MsgDataAlloc_2016(out msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    MessageBox.Show("Message allocation error: " + result, "Error");

// Free message structure
result = CanTpApi.MsgDataFree_2016(ref msg);
if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    MessageBox.Show("Message free error: " + result, "Error");
```

C++ / CLR

```
cantp_status result;
cantp_msg msg;

// Allocate message structure
result = CanTpApi_2016::MsgDataAlloc_2016(msg, PCANTP_MSGTYPE_ISOTP);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox::Show(String::Format("Message allocation error: {0}", (int)result), "Error");

// Free message structure
result = CanTpApi_2016::MsgDataFree_2016(msg);
if (!CanTpApi_2016::StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox::Show(String::Format("Message free error: ", (int)result), "Error");
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg

' Allocate message structure
result = CanTpApi.MsgDataAlloc_2016(msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Message allocation error: " + result.ToString(), "Error")
End If

' Free message structure
result = CanTpApi.MsgDataFree_2016(msg)
If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
    MessageBox.Show("Message free error: " + result.ToString(), "Error")
End If
```

Pascal OO

```
var
  result: cantp_status;
  msg: cantp_msg;
begin

  // Allocate message structure
  result := TCanTpApi.MsgDataAlloc_2016(msg,
    cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    MessageBox(0, PWideChar(format('Message allocation error: %d',
      [Integer(result)])), 'Error', MB_OK);
  end;

  // Free message structure
  result := TCanTpApi.MsgDataFree_2016(msg);
  if NOT TCanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false)
  then
  begin
    MessageBox(0, PWideChar(format('Message free error: %d', [Integer(result)])
      ), 'Error', MB_OK);
  end;
end;
```

Plain function version: CANTP_MsgDataFree_2016 on page 249

See also: cantp_msg on page 28, MsgDataAlloc_2016 on page 186, MsgDataInit_2016 on page 188

3.6.45 allocProgressBuffer_2016

C# and VB specific helper method. In a progress structure (see cantp_msgprogress on page 31), allocates a buffer to receive a copy of the pending message. When finished, must be released (see freeProgressBuffer_2016 on page 200).

Syntax

C#

```
public static cantp_status allocProgressBuffer_2016(
    ref cantp_msgprogress prog,
    cantp_msgtype type);
```

Visual Basic

```
Public Shared Function allocProgressBuffer_2016(
    ByRef prog As cantp_msgprogress,
    ByVal type As cantp_msgtype) As cantp_status
End Function
```

Parameters

Parameters	Description
Prog	Progress structure containing the buffer to initialize (see cantp_msgprogress on page 31).
Type	Type of the pending message (see cantp_msgtype on page 85).

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the given message is not valid.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory for the given message.

Example

The following example shows the use of the method allocProgressBuffer_2016 on the channel PCANTP_HANDLE_USBBUS1. It prints the pending message status for each progress step.

Note: It is assumed that the channel was already initialized, and a message is being received.

C#

```
cantp_status result;
cantp_msg msg = new cantp_msg();

// Read message.
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out msg);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)
    && (cantp_msgtype.PCANTP_MSGTYPE_ISOTP & msg.type) == cantp_msgtype.PCANTP_MSGTYPE_ISOTP
    && ((msg.Msgdata_isotp_Copy.netaddrinfo.msgtype
    & cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX)
    == cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX))
{
    // An ISOTP message is being received, wait and show progress.
    cantp_msgprogress progress;
    do
    {
        progress = new cantp_msgprogress();
        CanTpApi.allocProgressBuffer_2016(ref progress, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, ref msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_RX, out progress);
        Console.Write("RX " + progress.percentage+ "%, data = ");

        // Not recommended for example only: get and print current pending message data
        cantp_msg pending_msg = new cantp_msg();
        CanTpApi.getProgressBuffer_2016(ref progress, ref pending_msg);
        byte data;
        for (int i=0;i< pending_msg.Msgdata_any_Copy.length;i++)
        {
            CanTpApi.getData_2016(ref pending_msg, i, out data);
            Console.Write("0x"+data.ToString("X2")+" ");
        }
        Console.Write("\n");
        CanTpApi.freeProgressBuffer_2016(ref progress);
    } while (progress.state == cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING);
}
else
{
    Console.WriteLine("Read error: " + result.ToString());
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg = New cantp_msg()

' Read message.
```

```
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) And
(cantp_msgtype.PCANTP_MSGTYPE_ISOTP And msg.type) = cantp_msgtype.PCANTP_MSGTYPE_ISOTP And
((msg.Msgdata_isotp_Copy.netaddrinfo.msgtype And
cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX) =
cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX) Then

    ' An ISOTP message is being received, wait and show progress
    Dim progress As cantp_msgprogress
    Do
        progress = New cantp_msgprogress()
        CanTpApi.allocProgressBuffer_2016(progress, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_RX, progress)
        Console.Write("RX " + progress.percentage.ToString() + "%, data = ")

        ' Not recommended for example only: get and print current pending message data
        Dim pending_msg As cantp_msg
        CanTpApi.getProgressBuffer_2016(progress, pending_msg)
        Dim data As Byte
        For i As UInt32 = 0 To pending_msg.Msgdata_any_Copy.length - 1
            CanTpApi.getData_2016(pending_msg, i, data)
            Console.Write("0x" + data.ToString("X2") + " ")
        Next
        Console.WriteLine("")
        CanTpApi.freeProgressBuffer_2016(progress)
    Loop While progress.state = cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING
Else
    Console.WriteLine("Read error: " + result.ToString())
End If
```

See also: cantp_msgprogress on page 31, freeProgressBuffer_2016 below, getProgressBuffer_2016 on page 202

3.6.46 freeProgressBuffer_2016

Free the buffer receiving the pending message in a progress structure, if allocated with allocProgressBuffer_2016 (see on page 198).

Syntax

C#

```
public static void freeProgressBuffer_2016(
    ref cantp_msgprogress prog);
```

Visual Basic

```
Public Shared Sub freeProgressBuffer_2016(
    ByRef prog As cantp_msgprogress)
End Sub
```

Parameters

Parameter	Description
prog	Progress structure containing the buffer (see cantp_msgprogress on page 31).

Example

The following example shows the use of the method freeProgressBuffer_2016 on the channel PCANTP_HANDLE_USBBUS1. It prints the pending message status for each progress step.

Note: It is assumed that the channel was already initialized, and a message is being received.

C#

```
cantp_status result;
cantp_msg msg = new cantp_msg();

// Read message.
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out msg);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)
    && (cantp_msgtype.PCANTP_MSGTYPE_ISOTP & msg.type) == cantp_msgtype.PCANTP_MSGTYPE_ISOTP
    && ((msg.Msgdata_isotp_Copy.netaddrinfo.msgtype
    & cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX)
    == cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX))
{
    // An ISOTP message is being received, wait and show progress
    cantp_msgprogress progress;
    do
    {
        progress = new cantp_msgprogress();
        CanTpApi.allocProgressBuffer_2016(ref progress, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, ref msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_RX, out progress);
        Console.Write("RX " + progress.percentage+ "%, data = ");

        // Not recommended for example only: get and print current pending message data.
        cantp_msg pending_msg = new cantp_msg();
        CanTpApi.getProgressBuffer_2016(ref progress, ref pending_msg);
        byte data;
        for (int i=0;i< pending_msg.Msgdata_any_Copy.length;i++)
        {
            CanTpApi.getData_2016(ref pending_msg, i, out data);
            Console.Write("0x"+data.ToString("X2")+" ");
        }
        Console.Write("\n");
        CanTpApi.freeProgressBuffer_2016(ref progress);
    } while (progress.state == cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING);
}
else
{
    Console.WriteLine("Read error: " + result.ToString());
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg = New cantp_msg()

' Read message.
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) And
(cantp_msgtype.PCANTP_MSGTYPE_ISOTP And msg.type) = cantp_msgtype.PCANTP_MSGTYPE_ISOTP And
((msg.Msgdata_isotp_Copy.netaddrinfo.msgtype And
cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX) =
cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX) Then

    ' An ISOTP message is being received, wait and show progress
    Dim progress As cantp_msgprogress
    Do
        progress = New cantp_msgprogress()
        CanTpApi.allocProgressBuffer_2016(progress, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg,
```

```
            cantp_msgdirection.PCANTP_MSGDIRECTION_RX, progress)
        Console.Write("RX " + progress.percentage.ToString() + "%, data = ")

        ' Not recommended for example only: Get and print current pending message data
        Dim pending_msg As cantp_msg
        CanTpApi.getProgressBuffer_2016(progress, pending_msg)
        Dim data As Byte
        For i As UInt32 = 0 To pending_msg.Msgdata_any_Copy.length - 1
            CanTpApi.getData_2016(pending_msg, i, data)
            Console.Write("0x" + data.ToString("X2") + " ")
        Next
        Console.WriteLine("")
        CanTpApi.freeProgressBuffer_2016(progress)
    Loop While progress.state = cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING
Else
    Console.WriteLine("Read error: " + result.ToString())
End If
```

See also: cantp_msgprogress on page 31, allocProgressBuffer_2016 on page 198, getProgressBuffer_2016 below

3.6.47 getProgressBuffer_2016

C# and VB specific helper method. Get the current pending message of a progress structure.

Syntax

C#

```
public static bool getProgressBuffer_2016(
    ref cantp_msgprogress prog,
    ref cantp_msg pendmsg);
```

Visual Basic

```
Public Shared Function getProgressBuffer_2016(
    ByRef prog As cantp_msgprogress,
    ByRef pendmsg As cantp_msg) As Boolean
End Function
```

Parameters

Parameters	Description
prog	The cantp_msgprogress (see cantp_msgprogress on page 31).
pendmsg	Output, copy of the pending message (see cantp_msg on page 28)

Returns

True if ok, false if not ok.

Remarks

The progress structure buffer must be initialized (see allocProgressBuffer_2016 on page 198) and free (see freeProgressBuffer_2016 on page 200).

Keep in mind that requesting the content of a pending message will slowdown pending communications. Consequently, depending on the configured communication timeouts, it may disrupt ISO-TP communications.

Example

The following example shows the use of the method getProgressBuffer_2016 on the channel PCANTP_HANDLE_USBBUS1. It prints the pending message status for each progress step.

Note: It is assumed that the channel was already initialized, and a message is being received.

C#

```
cantp_status result;
cantp_msg msg = new cantp_msg();

// Read message.
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, out msg);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK)
    && (cantp_msgtype.PCANTP_MSGTYPE_ISOTP & msg.type) == cantp_msgtype.PCANTP_MSGTYPE_ISOTP
    && ((msg.Msgdata_isotp_Copy.netaddrinfo.msgtype
    & cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX)
    == cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX))
{
    // An ISOTP message is being received, wait and show progress
    cantp_msgprogress progress;
    do
    {
        progress = new cantp_msgprogress();
        CanTpApi.allocProgressBuffer_2016(ref progress, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, ref msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_RX, out progress);
        Console.Write("RX " + progress.percentage+ "%, data = ");

        // not recommended for example only: Get and print current pending message data
        cantp_msg pending_msg = new cantp_msg();
        CanTpApi.getProgressBuffer_2016(ref progress, ref pending_msg);
        byte data;
        for (int i=0;i< pending_msg.Msgdata_any_Copy.length;i++)
        {
            CanTpApi.getData_2016(ref pending_msg, i, out data);
            Console.Write("0x"+data.ToString("X2")+" ");
        }
        Console.Write("\n");
        CanTpApi.freeProgressBuffer_2016(ref progress);
    } while (progress.state == cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING);
}
else
{
    Console.WriteLine("Read error: " + result.ToString());
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg = New cantp_msg()

' Read message.
result = CanTpApi.Read_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) And
(cantp_msgtype.PCANTP_MSGTYPE_ISOTP And msg.type) = cantp_msgtype.PCANTP_MSGTYPE_ISOTP And
((msg.Msgdata_isotp_Copy.netaddrinfo.msgtype And
cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX) =
cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_RX) Then

    ' An ISOTP message is being received, wait and show progress
```

```
    Dim progress As cantp_msgprogress
    Do
        progress = New cantp_msgprogress()
        CanTpApi.allocProgressBuffer_2016(progress, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
        result = CanTpApi.GetMsgProgress_2016(cantp_handle.PCANTP_HANDLE_USBBUS1, msg,
            cantp_msgdirection.PCANTP_MSGDIRECTION_RX, progress)
        Console.Write("RX " + progress.percentage.ToString() + "%, data = ")

        ' Not recommended for example only: Get and print current pending message data
        Dim pending_msg As cantp_msg
        CanTpApi.getProgressBuffer_2016(progress, pending_msg)
        Dim data As Byte
        For i As UInt32 = 0 To pending_msg.Msgdata_any_Copy.length - 1
            CanTpApi.getData_2016(pending_msg, i, data)
            Console.Write("0x" + data.ToString("X2") + " ")
        Next
        Console.WriteLine("")
        CanTpApi.freeProgressBuffer_2016(progress)
    Loop While progress.state = cantp_msgprogress_state.PCANTP_MSGPROGRESS_STATE_PROCESSING
Else
    Console.WriteLine("Read error: " + result.ToString())
End If
```

See also: cantp_msgprogress on page 31, allocProgressBuffer_2016 on page 198, freeProgressBuffer_2016 on page 200

3.6.48 getFlags_2016

C# and VB specific helper method. Get the flags of a message in a safe way.

Syntax

C#

```
public static bool getFlags_2016(
    ref cantp_msg msg,
    out cantp_msgflag flags);
```

Visual Basic

```
Public Shared Function getFlags_2016(
    ByRef msg As cantp_msg,
    ByRef flags As cantp_msgflag) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containg the flags (see cantp_msg on page 28).
flags	Output, value of the flags (see cantp_msgflag on page 87)

Returns

True if ok, false if not ok.

Remarks

This method is safer than directly getting the value via the cantp_msg structure because it checks that message data is not null.

Example

The following example shows the use of the method getFlags_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel and the message were already initialized.

C#

```
// Get and print message flags value
cantp_msgflag flags;
if (CanTpApi.getFlags_2016(ref msg, out flags))
    MessageBox.Show("Message flags value = " + flags, "Info");
else
    MessageBox.Show("Get flags error!", "Error");
```

Visual Basic

```
' Get and print message flags value
Dim flags As cantp_msgflag
If CanTpApi.getFlags_2016(msg, flags) Then
    MessageBox.Show("Message flags value = " + flags.ToString(), "Info")
Else
    MessageBox.Show("Get flags error!", "Error")
End If
```

See also: cantp_msgflag on page 87, cantp_msg on page 28, cantp_msgdata on page 22

3.6.49 setLength_2016

C# and VB specific helper method. Set the length of a message. It should be use carefully: the proper way to set message size is using MsgDataInit_2016 (see on page 188).

Syntax

C#

```
public static bool setLength_2016(
    ref cantp_msg msg,
    UInt32 len);
```

Visual Basic

```
Public Shared Function setLength_2016(
    ByRef msg As cantp_msg,
    ByVal len As UInt32) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message structure (see cantp_msg on page 28).
len	New length value.

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method setLength_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel and the message were already initialized.

C#

```
// Set new message length to 7
if (CanTpApi.setLength_2016(ref request_msg, 7))
{
    uint length;
    if (CanTpApi.getLength_2016(ref request_msg, out length))
        MessageBox.Show("New message length = " + length, "Info");
    else
        MessageBox.Show("Get length error!", "Error");
}
else
{
    MessageBox.Show("Set length error!", "Error");
}
```

Visual Basic

```
' Set new message length to 7
If CanTpApi.setLength_2016(request_msg, 7) Then
    Dim length As UInt32
    If CanTpApi.getLength_2016(request_msg, length) Then
        MessageBox.Show("New message length = " + length.ToString(), "Info")
    Else
        MessageBox.Show("Get length error!", "Error")
    End If
Else
    MessageBox.Show("Set length error!", "Error")
End If
```

See also: getLength_2016 below, cantp_msg on page 28, cantp_msgdata on page 22

3.6.50 getLength_2016

C# and VB specific helper method. Get the length of a message in a safe way.

Syntax

C#

```
public static bool getLength_2016(
    ref cantp_msg msg,
    out UInt32 len);
```

Visual Basic

```
Public Shared Function getLength_2016(
    ByRef msg As cantp_msg,
    ByRef len As UInt32) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message to get data length (see cantp_msg on page 28).
len	Output, value of the length.

Returns

True if ok, false if not ok.

Remarks

This method is safer than directly getting the value via the cantp_msg structure because it checks that message data is not null.

Example

The following example shows the use of the method getLength_2016 on the channel PCANTP_HANDLE_PCIBUS1. It allocates, initializes a message then it prints the message length.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg request_msg;

result = CanTpApi.MsgDataAlloc_2016(out request_msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{

    // Initialize the allocated message with "PEAK" as data
    result = CanTpApi.MsgDataInit_2016(out request_msg, request_mapping.can_id,
        request_mapping.can_msgtype, 4, "PEAK", ref request_mapping.netaddrinfo);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {
        MessageBox.Show("Message initialization error: " + result, "Error");
    }
    else
    {
        uint length;
        if (CanTpApi.getLength_2016(ref request_msg, out length))
            MessageBox.Show("Message length = " + length, "Info");
        else
            MessageBox.Show("Get length error!", "Error");
    }
}
```

Visual Basic

```
Dim result As cantp_status
Dim request_msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(request_msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "PEAK" as data
    result = CanTpApi.MsgDataInit_2016(request_msg, request_mapping.can_id,
        request_mapping.can_msgtype, 4, "PEAK", request_mapping.netaddrinfo)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    Else
```

```
        Dim length As UInt32
        If CanTpApi.getLength_2016(request_msg, length) Then
            MessageBox.Show("Message length = " + length.ToString(), "Info")
        Else
            MessageBox.Show("Get length error!", "Error")
        End If
    End If
End If
```

See also: setLength_2016 on page 205, cantp_msg on page 28, cantp_msgdata on page 22

3.6.51 setData_2016

C# and VB specific helper method. Set the data of a message.

Overloads

	Method	Description
	setData_2016(cantp_msg, Int32, byte)	Set a byte of the data of a message.
	setData_2016(cantp_msg, Int32, byte[], Int32)	Set bytes of the data of a message from a contiguous byte array.

3.6.52 setData_2016(cantp_msg, Int32, byte)

C# and VB specific helper method. Set a byte of the data of a message.

Syntax

C#

```
public static bool setData_2016(
    ref cantp_msg msg,
    Int32 i,
    byte val);
```

Visual Basic

```
Public Shared Function setData_2016(
    ByRef msg As cantp_msg,
    ByVal i As Integer,
    ByVal val As Byte) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the data (see cantp_msg on page 28).
i	Offset of the byte, cannot be more than 2147483647.
val	New byte value

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method setData_2016 on the channel PCANTP_HANDLE_PCIBUS1. It allocates, initializes and set the data of a message then it prints the message data using getData_2016 method.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg msg;

result = CanTpApi.MsgDataAlloc_2016(out msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    // Initialize the allocated message with "OK" as data
    result = CanTpApi.MsgDataInit_2016(out msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, (String)null, ref request_mapping.netaddrinfo);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {
        MessageBox.Show("Message initialization error: " + result, "Error");
    }
    else
    {
        // Set new data
        Byte new_data_0 = Convert.ToByte('O');
        Byte new_data_1 = Convert.ToByte('K');
        if (CanTpApi.setData_2016(ref msg, 0, new_data_0)
            && CanTpApi.setData_2016(ref msg, 1, new_data_1))
        {
            // Get and print message data
            Byte[] data = new Byte[2];
            if (CanTpApi.getData_2016(ref msg, 0, data, 2))
            {
                MessageBox.Show("Get data: \"" + Convert.ToChar(data[0]) + ""
                    + Convert.ToChar(data[1]) + "\"", "Info");
            }
            else
            {
                MessageBox.Show("Get data error: " + result, "Error");
            }
        }
        else
        {
            MessageBox.Show("Set data error: " + result, "Error");
        }
    }
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "OK" as data
    Dim str_data As String = Nothing
    result = CanTpApi.MsgDataInit_2016(msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, str_data, request_mapping.netaddrinfo)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    Else

        ' Set new data: "OK"
        Dim new_data_0 As Byte = &H4F
```

```
        Dim new_data_1 As Byte = &H4B
        If CanTpApi.setData_2016(msg, 0, new_data_0) And CanTpApi.setData_2016(msg, 1,
                new_data_1) Then

            ' Get and print message data
            Dim data(2) As Byte
            If CanTpApi.getData_2016(msg, 0, data, 2) Then
                MessageBox.Show("Get data: " + Convert.ToChar(data(0)) + "" +
                    Convert.ToChar(data(1)), "Info")
            Else
                MessageBox.Show("Get data error: " + result.ToString(), "Error")
            End If
        Else
            MessageBox.Show("Set data error: " + result.ToString(), "Error")
        End If
    End If
End If
```

See also: getData_2016 on page 212, cantp_msg on page 28, cantp_msgdata on page 22

3.6.53 setData_2016(cantp_msg, Int32, byte[], Int32)

C# and VB specific helper method. Set bytes of the data of a message from a contiguous byte array.

Syntax

C#

```
public static bool setData_2016(
    ref cantp_msg msg,
    Int32 i,
    byte[] vals,
    Int32 nb);
```

Visual Basic

```
Public Shared Function setData_2016(
    ByRef msg As cantp_msg,
    ByVal i As Integer,
    ByVal vals As Byte(),
    ByVal nb As Integer) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the data (see cantp_msg on page 28).
i	Offset of the first byte to set in the message, cannot be more than 2147483647.
vals	Values to set.
nb	Number of bytes to set.

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method setData_2016 on the channel PCANTP_HANDLE_PCIBUS1. It allocates, initializes and set the data of a message then it prints the message data using getData_2016 method.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg msg;

result = CanTpApi.MsgDataAlloc_2016(out msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{

    // Initialize the allocated message with "OK" as data
    result = CanTpApi.MsgDataInit_2016(out msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, (String)null, ref request_mapping.netaddrinfo);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {
        MessageBox.Show("Message initialization error: " + result, "Error");
    }
    else
    {
        // Set new data
        Byte[] new_data = new Byte[2];
        new_data[0]=Convert.ToByte('O');
        new_data[1]=Convert.ToByte('K');
        if (CanTpApi.setData_2016(ref msg, 0, new_data,2))
        {

            // Get and print message data
            Byte[] data = new Byte[2];
            if (CanTpApi.getData_2016(ref msg, 0, data, 2))
            {
                MessageBox.Show("Get data: \"" + Convert.ToChar(data[0]) + ""
                    + Convert.ToChar(data[1]) + "\"", "Info");
            }
            else
            {
                MessageBox.Show("Get data error: " + result, "Error");
            }
        }
        else
        {
            MessageBox.Show("Set data error: " + result, "Error");
        }
    }
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "OK" as data
    Dim str_data As String = Nothing
    result = CanTpApi.MsgDataInit_2016(msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, str_data, request_mapping.netaddrinfo)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    Else
```

```
        ' Set new data "OK"
        Dim new_data(2) As Byte
        new_data(0) = &H4F
        new_data(1) = &H4B
        If CanTpApi.setData_2016(msg, 0, new_data, 2) Then

            ' Get and print message data
            Dim data(2) As Byte
            If CanTpApi.getData_2016(msg, 0, data, 2) Then
                MessageBox.Show("Get data: " + Convert.ToChar(data(0)) + "" +
                    Convert.ToChar(data(1)), "Info")
            Else
                MessageBox.Show("Get data error: " + result.ToString(), "Error")
            End If
        Else
            MessageBox.Show("Set data error: " + result.ToString(), "Error")
        End If
    End If
End If
```

See also: getData_2016 below, cantp_msg on page 28, cantp_msgdata on page 22

3.6.54 getData_2016

C# and VB specific helper method. Get the data of a message.

Overloads

	Method	Description
	getData_2016(cantp_msg, Int32, byte)	Get a byte of the data of a message.
	getData_2016(cantp_msg, Int32, byte[], Int32)	Get bytes of the data of a message.

3.6.55 getData_2016(cantp_msg, Int32, byte)

C# and VB specific helper method. Get a byte of the data of a message.

Syntax

C#

```
public static bool getData_2016(
    ref cantp_msg msg,
    Int32 i,
    out byte val);
```

Visual Basic

```
Public Shared Function getData_2016(
    ByRef msg As cantp_msg,
    ByVal i As Integer,
    ByRef val As Byte) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the data to retrieve (see cantp_msg on page 28).
i	Offset of the bytes in the message, cannot be more than 2147483647.
val	Output, value of the byte.

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method getData_2016 on the channel PCANTP_HANDLE_PCIBUS1. It allocates, initializes and print the data of a message.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg msg;

result = CanTpApi.MsgDataAlloc_2016(out msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{

    // Initialize the allocated message with "OK" as data
    result = CanTpApi.MsgDataInit_2016(out msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, "OK", ref request_mapping.netaddrinfo);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {
        MessageBox.Show("Message initialization error: " + result, "Error");
    }
    else
    {
        // Get and print message data
        Byte data_0, data_1;
        if (CanTpApi.getData_2016(ref msg, 0, out data_0)
            && CanTpApi.getData_2016(ref msg, 1, out data_1))
        {
            MessageBox.Show("Get data: \"" + Convert.ToChar(data_0) + ""
                + Convert.ToChar(data_1) + "\"", "Info");
        }
        else
        {
            MessageBox.Show("Get data error: " + result, "Error");
        }
    }
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "OK" as data
    result = CanTpApi.MsgDataInit_2016(msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, "OK", request_mapping.netaddrinfo)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    Else

        ' Get and print message data
        Dim data_0 As Byte
        Dim data_1 As Byte
```

```
        If CanTpApi.getData_2016(msg, 0, data_0) And CanTpApi.getData_2016(msg, 1, data_1) Then
            MessageBox.Show("Get data: " + Convert.ToChar(data_0) + "" +
                Convert.ToChar(data_1), "Info")
        Else
            MessageBox.Show("Get data error: " + result.ToString(), "Error")
        End If
    End If
End If
```

See also: setData_2016 on page 208, cantp_msg on page 28, cantp_msgdata on page 22

3.6.56 getData_2016(cantp_msg, Int32, byte[], Int32)

C# and VB specific helper method. Get bytes of the data of a message.

Syntax

C#

```
public static bool getData_2016(
    ref cantp_msg msg,
    Int32 i,
    byte[] vals,
    Int32 nb);
```

Visual Basic

```
Public Shared Function getData_2016(
    ByRef msg As cantp_msg,
    ByVal i As Integer,
    ByRef vals As Byte(),
    ByVal nb As Integer) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the data to get (see cantp_msg on page 28).
i	Offset of the first byte to get in the message, cannot be more than 2147483647
vals	Output, data values.
nb	Number of bytes to get

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method getData_2016 on the channel PCANTP_HANDLE_PCIBUS1. It allocates, initializes and print the data of a message.

Note: It is assumed that the channel and the mapping were already initialized.

C#

```
cantp_status result;
cantp_msg msg;

result = CanTpApi.MsgDataAlloc_2016(out msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP);
```

```
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{

    // Initialize the allocated message with "OK" as data
    result = CanTpApi.MsgDataInit_2016(out msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, "OK", ref request_mapping.netaddrinfo);
    if (!CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
    {
        MessageBox.Show("Message initialization error: " + result, "Error");
    }
    else
    {
        // Get and print message data
        Byte[] data = new Byte[2];
        if (CanTpApi.getData_2016(ref msg, 0, data, 2))
        {
            MessageBox.Show("Get data: \"" + Convert.ToChar(data[0]) + ""
                + Convert.ToChar(data[1]) + "\"", "Info");
        }
        else
        {
            MessageBox.Show("Get data error: " + result, "Error");
        }
    }
}
```

Visual Basic

```
Dim result As cantp_status
Dim msg As cantp_msg

result = CanTpApi.MsgDataAlloc_2016(msg, cantp_msgtype.PCANTP_MSGTYPE_ISOTP)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    ' Initialize the allocated message with "OK" as data
    result = CanTpApi.MsgDataInit_2016(msg, request_mapping.can_id,
        request_mapping.can_msgtype, 2, "OK", request_mapping.netaddrinfo)
    If Not CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then
        MessageBox.Show("Message initialization error: " + result.ToString(), "Error")
    Else

        ' Get and print message data
        Dim data(2) As Byte
        If CanTpApi.getData_2016(msg, 0, data, 2) Then
            MessageBox.Show("Get data: " + Convert.ToChar(data(0)) + "" +
                Convert.ToChar(data(1)), "Info")
        Else
            MessageBox.Show("Get data error: " + result.ToString(), "Error")
        End If
    End If
End If
```

See also: setData_2016 on page 208, cantp_msg on page 28, cantp_msgdata on page 22

3.6.57 getNetStatus_2016

C# and VB specific helper method. Get the netstatus of a message in a safe way.

Syntax

C#

```
public static bool getNetStatus_2016(
    ref cantp_msg msg,
    out cantp_netstatus status);
```

Visual Basic

```
Public Shared Function getNetStatus_2016(
    ByRef msg As cantp_msg,
    ByRef status As cantp_netstatus) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the netstatus (see cantp_msg on page 28).
status	Output parameter, will contain the netstatus value (see cantp_netstatus on page 53)

Returns

True if ok, false if not ok.

Remarks

This method is safer than directly getting the value via the cantp_msg structure because it checks that message data is not null.

Example

The following example shows the use of the method getOptionsNumber_2016 on the channel PCANTP_HANDLE_PCIBUS1. It prints the network status of an already initialized message.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
cantp_netstatus netstatus_buffer;
if (CanTpApi.getNetStatus_2016(ref msg, out netstatus_buffer))
    MessageBox.Show("Netstatus: " + netstatus_buffer, "Info");
else
    MessageBox.Show("Cannot get netstatus of the given message!", "Error");
```

Visual Basic

```
Dim netstatus_buffer As cantp_netstatus
If CanTpApi.getNetStatus_2016(msg, netstatus_buffer) Then
    MessageBox.Show("Netstatus: " + netstatus_buffer.ToString(), "Info")
Else
    MessageBox.Show("Cannot get netstatus of the given message!", "Error")
End If
```

See also: cantp_msg on page 28, cantp_netstatus on page 53

3.6.58 getOption_2016

C# and VB specific helper method. Get an option of a message.

Syntax

C#

```
public static bool getOption_2016(
    ref cantp_msg msg,
    int number,
    out cantp_msgoption option);
```

Visual Basic

```
Public Shared Function getOption_2016(
    ByRef msg As cantp_msg,
    ByVal number As Integer,
    ByRef opt As cantp_msgoption) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the option (see cantp_msg on page 28).
number	Number of the option (index).
option	Where to store a copy of the option (see cantp_msgoption on page 16).

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method getOption_2016. It sets a new priority for an already initialized message then check its value by using getOption_2016 method.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
cantp_status result;

result = CanTpApi.MsgDataInitOptions_2016(out request_msg, 1);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    cantp_msgoption option;
    option.name = cantp_option.PCANTP_OPTION_J1939_PRIORITY;
    option.value = 0x02;
    if (CanTpApi.setOption_2016(ref request_msg, 0, ref option))
        MessageBox.Show("Set message priority.", "Info");
    else
        MessageBox.Show("Cannot set message priority.", "Error");

    // Get message option (value should be 0x02)
    cantp_msgoption option_buffer;
    if (CanTpApi.getOption_2016(ref request_msg, 0, out option_buffer))
        MessageBox.Show("Get message priority value: " + option_buffer.value, "Info");
    else
        MessageBox.Show("Cannot get message priority.", "Error");
}
```

Visual Basic

```
Dim result As cantp_status
```

```
result = CanTpApi.MsgDataInitOptions_2016(request_msg, 1)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    Dim new_option As cantp_msgoption
    new_option.name = cantp_option.PCANTP_OPTION_J1939_PRIORITY
    new_option.value = &H2
    If CanTpApi.setOption_2016(request_msg, 0, new_option) Then
        MessageBox.Show("Set message priority.", "Info")
    Else
        MessageBox.Show("Cannot set message priority.", "Error")
    End If

    ' Get message option (value should be 0x02)
    Dim option_buffer As cantp_msgoption
    If CanTpApi.getOption_2016(request_msg, 0, option_buffer) Then
        MessageBox.Show("Get message priority value: " + option_buffer.value.ToString(),
            "Info")
    Else
        MessageBox.Show("Cannot get message priority.", "Error")
    End If
End If
```

See also: setOption_2016 below, cantp_msgoption on page 16, cantp_msg on page 28

3.6.59 setOption_2016

C# and VB specific helper method. Modifies an option of a message.

Syntax

C#

```
public static bool setOption_2016(
    ref cantp_msg msg,
    int number,
    ref cantp_msgoption option);
```

Visual Basic

```
Public Shared Function setOption_2016(
    ByRef msg As cantp_msg,
    ByVal number As Integer,
    ByRef opt As cantp_msgoption) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the option (see cantp_msg on page 28).
number	Number of the option (index).
option	Value to set to the option (see cantp_msgoption on page 16).

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method setOption_2016. It sets a new priority for an already initialized message.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
cantp_status result;

// Set priority
result = CanTpApi.MsgDataInitOptions_2016(out request_msg, 1);
if (CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK, false))
{
    cantp_msgoption option;
    option.name = cantp_option.PCANTP_OPTION_J1939_PRIORITY;
    option.value = 0x02;
    if (CanTpApi.setOption_2016(ref request_msg, 0, ref option))
        MessageBox.Show("Set message priority.", "Info");
    else
        MessageBox.Show("Cannot set message priority.", "Error");
}
```

Visual Basic

```
Dim result As cantp_status

result = CanTpApi.MsgDataInitOptions_2016(request_msg, 1)
If CanTpApi.StatusIsOk_2016(result, cantp_status.PCANTP_STATUS_OK) Then

    Dim new_option As cantp_msgoption
    new_option.name = cantp_option.PCANTP_OPTION_J1939_PRIORITY
    new_option.value = &H2
    If CanTpApi.setOption_2016(request_msg, 0, new_option) Then
        MessageBox.Show("Set message priority.", "Info")
    Else
        MessageBox.Show("Cannot set message priority.", "Error")
    End If
End If
```

See also: getOption_2016 on page 216, cantp_msgoption on page 16, cantp_msg on page 28, MsgDataInitOptions_2016 on page 194

3.6.60 getOptionsNumber_2016

C# and VB specific helper method. Get the number of options of a message.

Syntax

C#

```
public static bool getOptionsNumber_2016(
    ref cantp_msg msg,
    out UInt32 number);
```

Visual Basic

```
Public Shared Function getOptionsNumber_2016(
    ByRef msg As cantp_msg,
    ByRef number As Integer) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the options (see cantp_msg on page 28).
number	Will contain the number of option (output parameter).

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method getOptionsNumber_2016 on the channel PCANTP_HANDLE_PCIBUS1. It prints the number of options of an already initialized message.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
uint number_of_options;
CanTpApi.getOptionsNumber_2016(ref msg, out number_of_options);
MessageBox.Show("Number of message options: " + number_of_options, "Info");
```

Visual Basic

```
Dim number_of_options As UInt32
CanTpApi.getOptionsNumber_2016(msg, number_of_options)
MessageBox.Show("Number of message options: " + number_of_options.ToString(), "Info")
```

See also: getOption_2016 on page 216, setOption_2016 on page 218, cantp_msg on page 28

3.6.61 setNetaddrinfo_2016

C# and VB specific helper method. Set the network address information of an ISO-TP message.

Syntax

C#

```
public static bool setNetaddrinfo_2016(
    ref cantp_msg msg,
    ref cantp_netaddrinfo adr);
```

Visual Basic

```
Public Shared Function setNetaddrinfo_2016(
    ByRef msg As cantp_msg,
    ByRef adr As cantp_netaddrinfo) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the network address information structure (see cantp_msg on page 28).
adr	The new network address information of the message (see cantp_netaddrinfo on page 19).

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method setNetaddrinfo_2016 on the channel PCANTP_HANDLE_PCIBUS1. It set a new network address information in an already initialized message then get and prints this information.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
cantp_netaddrinfo nai;
nai.extension_addr = 0x00;
nai.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_NORMAL;
nai.msgtype = cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC;
nai.source_addr = 0xF1;
nai.target_addr = 0x01;
nai.target_type = cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Set new network address information
if (CanTpApi.setNetaddrinfo_2016(ref msg, ref nai))
{
    // Get and print new network address information
    cantp_netaddrinfo nai_buffer;
    if (CanTpApi.getNetaddrinfo_2016(ref msg, out nai_buffer))
    {
        Console.WriteLine("Extension address: " + nai_buffer.extension_addr);
        Console.WriteLine("Addressing format: " + nai_buffer.format);
        Console.WriteLine("Isotp message type: " + nai_buffer.msgtype);
        Console.WriteLine("Source address: " + nai_buffer.source_addr);
        Console.WriteLine("Target address: " + nai_buffer.target_addr);
        Console.WriteLine("Target type: " + nai_buffer.target_type);
    }
    else
    {
        Console.WriteLine("Get network information error!");
    }
}
else
{
    Console.WriteLine("Set network information error!");
}
```

Visual Basic

```
Dim nai As cantp_netaddrinfo
nai.extension_addr = &H0
nai.format = cantp_isotp_format.PCANTP_ISOTP_FORMAT_NORMAL
nai.msgtype = cantp_isotp_msgtype.PCANTP_ISOTP_MSGTYPE_DIAGNOSTIC
nai.source_addr = &HF1
nai.target_addr = &H1
nai.target_type = cantp_isotp_addressing.PCANTP_ISOTP_ADDRESSING_PHYSICAL

' Set new network address information
If CanTpApi.setNetaddrinfo_2016(msg, nai) Then

    ' Get and print new network address information
    Dim nai_buffer As cantp_netaddrinfo
    If CanTpApi.getNetaddrinfo_2016(msg, nai_buffer) Then
        Console.WriteLine("Extension address: " + nai_buffer.extension_addr.ToString())
        Console.WriteLine("Addressing format: " + nai_buffer.format.ToString())
        Console.WriteLine("Isotp message type: " + nai_buffer.msgtype.ToString())
        Console.WriteLine("Source address: " + nai_buffer.source_addr.ToString())
```

```
        Console.WriteLine("Target address: " + nai_buffer.target_addr.ToString())
        Console.WriteLine("Target type: " + nai_buffer.target_type.ToString())
    Else
        Console.WriteLine("Get network information error!")
    End If
Else
    Console.WriteLine("Set network information error!")
End If
```

See also: getNetaddrinfo_2016 below, cantp_msg on page 28, cantp_netaddrinfo on page 19

3.6.62 getNetaddrinfo_2016

C# and VB specific helper method. Get the network address information of an ISO-TP message.

Syntax

C#

```
public static bool getNetaddrinfo_2016(
    ref cantp_msg msg,
    out cantp_netaddrinfo adr);
```

Visual Basic

```
Public Shared Function getNetaddrinfo_2016(
    ByRef msg As cantp_msg,
    ByRef adr As cantp_netaddrinfo) As Boolean
End Function
```

Parameters

Parameters	Description
msg	The message containing the network address information structure (see cantp_msg on page 28).
adr	Store the network address information of the message (see cantp_netaddrinfo on page 19).

Returns

True if ok, false if not ok.

Example

The following example shows the use of the method getNetaddrinfo_2016 on the channel PCANTP_HANDLE_PCIBUS1. It prints the network address information of an already initialized message.

Note: It is assumed that the channel and the message structure were already initialized.

C#

```
cantp_netaddrinfo nai_buffer;
if (CanTpApi.getNetaddrinfo_2016(ref msg, out nai_buffer))
{
    Console.WriteLine("Extension address: " + nai_buffer.extension_addr);
    Console.WriteLine("Addressing format: " + nai_buffer.format);
    Console.WriteLine("Isotp message type: " + nai_buffer.msgtype);
    Console.WriteLine("Source address: " + nai_buffer.source_addr);
    Console.WriteLine("Target address: " + nai_buffer.target_addr);
    Console.WriteLine("Target type: " + nai_buffer.target_type);
}
else
{
```

```
    Console.WriteLine("Get network information error!");
}
```

Visual Basic

```
Dim nai_buffer As cantp_netaddrinfo
If CanTpApi.getNetaddrinfo_2016(msg, nai_buffer) Then
    Console.WriteLine("Extension address: " + nai_buffer.extension_addr.ToString())
    Console.WriteLine("Addressing format: " + nai_buffer.format.ToString())
    Console.WriteLine("Isotp message type: " + nai_buffer.msgtype.ToString())
    Console.WriteLine("Source address: " + nai_buffer.source_addr.ToString())
    Console.WriteLine("Target address: " + nai_buffer.target_addr.ToString())
    Console.WriteLine("Target type: " + nai_buffer.target_type.ToString())
Else
    Console.WriteLine("Get network information error!")
End If
```

See also: setNetaddrinfo_2016 on page 220, cantp_msg on page 28, cantp_netaddrinfo on page 19

3.7 Functions

The functions of the PCAN ISO-TP 2016 API are divided in 5 groups of functionalities.

Connection

	Function	Description
	CANTP_Initialize_2016	Initializes a PCANTP channel based on a CANTP handle (without CAN-FD support).
	CANTP_InitializeFD_2016	Initializes a PCANTP channel based on a CANTP handle (including CAN-FD support)
	CANTP_Uninitialize_2016	Uninitializes a PCANTP channel.

Configuration

	Function	Description
	CANTP_SetValue_2016	Sets a configuration or information value within a PCANTP channel.
	CANTP_AddMapping_2016	Configures the ISO-TP mapping between a CAN ID and an ISO-TP network addressing information.
	CANTP_RemoveMapping_2016	Removes a user-defined PCANTP mapping between a CAN ID and Network Address Information.
	CANTP_RemoveMappings_2016	Removes all user-defined PCANTP mappings corresponding to a CAN ID.
	CANTP_AddFiltering_2016	Adds an entry to the CAN-ID white-list filtering.
	CANTP_RemoveFiltering_2016	Removes an entry from the CAN-ID white-list filtering.

Information

	Function	Description
	CANTP_GetValue_2016	Retrieves information from a PCANTP channel.
	CANTP_StatusGet_2016	Retrieves the value of a cantp_status subtype.
	CANTP_StatusIsOk_2016	Checks if a status matches an expected result (default is PCANTP_STATUS_OK).
	CANTP_GetMsgProgress_2016	Gets progress information on a specific message.
	CANTP_GetErrorText_2016	Gets a descriptive text for an error code.
	CANTP_GetCanBusStatus_2016	Gets information about the internal BUS status of a PCANTP channel.
	CANTP_GetMappings_2016	Retrieves all the mappings defined for a PCANTP channel.
	CANTP_StatusListTypes_2016	Lists the subtypes contained in the PCANTP status.

Communication

	Function	Description
	CANTP_Read_2016	Reads a CAN message from the receive queue of a PCANTP channel.
	CANTP_Write_2016	Transmits a CAN message using a connected PCANTP channel.
	CANTP_Reset_2016	Resets the receive and transmit queues of a PCANTP channel.

Messages handling

	Function	Description
	CANTP_MsgDataAlloc_2016	Allocates a CANTP message based on the given type.
	CANTP_MsgDataInit_2016	Initializes an allocated CANTP message.
	CANTP_MsgDataFree_2016	Deallocates a CANTP message.
	CANTP_MsgEqual_2016	Checks if two CANTP messages are equal.

	Function	Description
	CANTP_MsgCopy_2016	Copies a CANTP message to another buffer.
	CANTP_MsgDlcToLength_2016	Converts a CAN DLC to its corresponding length.
	CANTP_MsgLengthToDlc_2016	Converts a data length to a corresponding CAN DLC.
	CANTP_MsgDataInitOptions_2016	Initializes several options for the CANTP message that will override the channel's parameter(s).

See also: class-method version on page 98

3.7.1 CANTP_Initialize_2016

Initializes a PCANTP channel based on a PCANTP handle (without CAN FD support).

Syntax

C++

```
cantp_status __stdcall CANTP_Initialize_2016(
        cantp_handle channel,
        cantp_baudrate baudrate,
        cantp_hwtype hw_type = 0,
        uint32_t io_port = 0,
        uint16_t interrupt = 0);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
baudrate	The speed for the communication (see cantp_baudrate on page 46)
hw_type	Non plug-n-play: the type of hardware (see cantp_hwtype on page 48)
io_port	Non plug-n-play: the I/O address for the parallel port.
interrupt	Non plug-n-play: interrupt number of the parallel port.

Returns

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

PCANTP_STATUS_ALREADY_INITIALIZED	Indicates that the desired PCANTP channel is already in use.
PCANTP_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error.

Remarks

The CANTP_Initialize_2016 function initiates a PCANTP channel, preparing it for communicate within the CAN bus connected to it. Calls to the other functions will fail, if they are used with a channel handle, different than PCANTP_HANDLE_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a 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_2016 on page 228).

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 2016 API uses the same function for initializations of both, Plug and Play, and non-Plug and Play hardware. The CANTP_Initialize_2016 function has three additional parameters that are only for the connection of Non-Plug and Play hardware. With Plug and Play hardware, however, only two parameters are to be supplied. The remaining three are not evaluated.

Example

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

C++

```
cantp_status result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CANTP_Initialize_2016(PCANTP_HANDLE_PCIBUS2, PCANTP_BAUDRATE_500K);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_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_2016(PCANTP_HANDLE_NONEBUS);
```

See also: CANTP_Uninitialize_2016 on page 227, Understanding PCAN-ISO-TP on page 7

Class method version: Initialize_2016 on page 99

3.7.2 CANTP_InitializeFD_2016

Initializes a PCANTP channel based on a CANTP handle (including CAN-FD support).

Syntax

C++

```
cantp_status __stdcall CANTP_InitializeFD_2016(
	cantp_handle channel,
	const cantp_bitrate bitrate_fd);
```

Parameters

Parameters	Description
channel	The handle of a FD capable PCAN Channel (see cantp_handle on page 37)
bitrate_fd	The speed for the communication (see FD Bit Rate Parameter Definitions on page 35, cantp_bitrate on page 34)

Returns

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

PCANTP_STATUS_ALREADY_INITIALIZED	Indicates that the desired PCANTP channel is already in use.
PCANTP_STATUS_FLAG_PCAN_STATUS	This error flag states that the error is composed of a more precise PCAN-Basic error.

Remarks

The CANTP_InitializeFD_2016 function initiates a FD capable PCANTP channel, preparing it for communicate within the CAN bus connected to it. Calls to the other functions will fail, if they are used with a channel handle, different than PCANTP_HANDLE_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a 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_2016 on page 228).

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-USB hardware).

C++

```
cantp_status result;

// The Plug and Play channel (PCAN-USB) is initialized @500kbps/2Mbps.
result = CANTP_InitializeFD_2016(PCANTP_HANDLE_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 (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_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_2016(PCANTP_HANDLE_NONEBUS);
```

See also: CANTP_Uninitialize_2016 below, Understanding PCAN-ISO-TP on page 7, FD Bit Rate Parameter Definitions on page 35

Class method version: InitializeFD_2016 on page 105

3.7.3 CANTP_Uninitialize_2016

Uninitializes an already initialized PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_Uninitialize_2016(
       cantp_handle channel);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)

Returns

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

PCANTP_STATUS_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 initializing function. If the given channel can not be found, then an error is returned.
- **Multiple-Release**: Giving the handle value PCANTP_HANDLE_NONEBUS which instructs the API to search for all channels initialized by the calling application and release them all. This option causes 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++

```
cantp_status result;

// The Plug and Play channel (PCAN-PCI) is initialized.
result = CANTP_Initialize_2016(PCANTP_HANDLE_PCIBUS2, PCANTP_BAUDRATE_500K);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
        MessageBox(NULL, "Initialization failed", "Error", MB_OK);
else
        MessageBox(NULL, "PCAN-PCI (Ch-2) was initialized", "Success", MB_OK);

// Uninitialized PCI channel 2
CANTP_Uninitialize_2016(PCANTP_HANDLE_PCIBUS2);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
        MessageBox(NULL, "Uninitialization failed", "Error", MB_OK);
else
        MessageBox(NULL, "PCAN-PCI (Ch-2) was uninitialized", "Success", MB_OK);
```

See also: CANTP_Initialize_2016 on page 225

Class method version: Uninitialize_2016 on page 107

3.7.4 CANTP_SetValue_2016

Sets a configuration or information value within a PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_GetValue_2016(
       cantp_handle channel,
       cantp_parameter parameter,
```

```
        void* buffer,
        uint32_t buffer_length);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
parameter	The code of the value to be set (see cantp_parameter on page 70)
buffer	The buffer containing the numeric value to be set.
buffer_length	The length in bytes of the given buffer.

Returns

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

PCANTP_STAUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the function are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length.

Remarks

Use the function CANTP_SetValue_2016 to set configuration information or environment values of a PCANTP channel.

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

More information about the parameters and values can be found in Detailed parameters values on page 75.

Example

The following example shows the use of the function CANTP_SetValue_2016 on the channel PCANTP_HANDLE_PCIBUS2 to enable debug mode.

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

C++

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

// Enable CAN DEBUG mode.
iBuffer = PCANTP_DEBUG_CAN;
result = CANTP_SetValue_2016(PCANTP_HANDLE_PCIBUS2, PCANTP_PARAMETER_DEBUG, &iBuffer,
        sizeof(unsigned int));
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
        MessageBox(NULL, "Failed to set value", "Error", MB_OK);
else
        MessageBox(NULL, "Value changed successfully ", "Success", MB_OK);
```

See also: CANTP_GetValue_2016 on page 235, cantp_parameter on page 70

Class method version: SetValue_2016 on page 110

3.7.5 CANTP_AddMapping_2016

Adds a user-defined PCANTP mapping between CAN ID and ISOTP Network Address Information within a PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_AddMapping_2016(
       cantp_handle channel,
       cantp_mapping* mapping);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
mapping	Mapping to be added. (see cantp_mapping on page 20)

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_ALREADY_INITIALIZED	A mapping with the same CAN ID already exists.
PCANTP_STATUS_PARAM_INVALID_VALUE	Mapping is not valid regarding ISO-TP standard.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory to define mapping.

Remarks

The cantp_mapping structure is described on page 20.

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++

```
cantp_handle CanChannel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = {};
cantp_mapping response_mapping = {};

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype = PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type = PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Defines a second mapping to allow communication from Destination 0x13 to Source 0xF1.
response_mapping = request_mapping;
response_mapping.can_id = request_mapping.can_id_flow_ctrl;
response_mapping.can_id_flow_ctrl = request_mapping.can_id;
response_mapping.netaddrinfo.source_addr = request_mapping.netaddrinfo.target_addr;
response_mapping.netaddrinfo.target_addr = request_mapping.netaddrinfo.source_addr;
```

```
// Add request mapping
result = CANTP_AddMapping_2016(CanChannel, &request_mapping);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Failed to add request mapping.", "Error", MB_OK);
// Add response mapping
result = CANTP_AddMapping_2016(CanChannel, &response_mapping);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Failed to add response mapping.", "Error", MB_OK);
```

See also: CANTP_RemoveMapping_2016 below and CANTP_RemoveMappings_2016 on page 232

Class method version: AddMapping_2016 on page 116

3.7.6 CANTP_RemoveMapping_2016

Removes a user-defined PCANTP mapping on a channel using a unique mapping identifier.

Syntax

C++

```
cantp_status __stdcall CANTP_RemoveMapping_2016(
       cantp_handle channel,
       uintptr_t uid);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
uid	Unique identifier of the mapping to remove.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_MAPPING_NOT_INITIALIZED	The PCANTP mapping to remove is not initialized.

Example

The following example shows the use of the function CANTP_RemoveMapping_2016 on the PCANTP channel USB 1. It adds a mapping and removes it using unique identifier.

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

C++

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = {};

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = PCANTP_CAN_MSGTYPE_STANDARD;
```

```
request_mapping.netaddrinfo.format = PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype = PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type = PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Add request mapping
result = CANTP_AddMapping_2016(channel, &request_mapping);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Failed to add request mapping.", "Error", MB_OK);

// Remove request mapping
result = CANTP_RemoveMapping_2016(channel, request_mapping.uid);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Failed to remove mapping.", "Error", MB_OK);
```

Class-method: RemoveMapping_2016 on page 120

See also: cantp_mapping on page 20, CANTP_RemoveMappings_2016 below, CANTP_AddMapping_2016 on page 229

3.7.7 CANTP_RemoveMappings_2016

Removes all user-defined PCANTP mappings corresponding to a CAN ID.

Syntax

C++

```
cantp_status __stdcall CANTP_RemoveMappings_2016(
       cantp_handle channel,
       uint32_t can_id);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
can_id	The mapped CAN Identifier to search for that identifies the mapping to remove.

Returns

The return value is a cantp_status code. PCANTP_ STATUS_OK is returned on success. The typical error in case of failure is:

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_MAPPING_NOT_INITIALIZED	The PCANTP CANID to remove is not specified in a mapping.

Example

The following example shows the definition and removal of a mapping using 0xD1 CANID.

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

C++

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
cantp_mapping request_mapping = {};

// Defines a first mapping to allow communication from Source 0xF1 to Destination 0x13.
request_mapping.can_id = 0xD1;
```

```
request_mapping.can_id_flow_ctrl = 0xD2;
request_mapping.netaddrinfo.source_addr = 0xF1;
request_mapping.netaddrinfo.target_addr = 0x13;
request_mapping.netaddrinfo.extension_addr = 0x52;
request_mapping.can_msgtype = PCANTP_CAN_MSGTYPE_STANDARD;
request_mapping.netaddrinfo.format = PCANTP_ISOTP_FORMAT_MIXED;
request_mapping.netaddrinfo.msgtype = PCANTP_ISOTP_MSGTYPE_REMOTE_DIAGNOSTIC;
request_mapping.netaddrinfo.target_type = PCANTP_ISOTP_ADDRESSING_PHYSICAL;

// Add request mapping
result = CANTP_AddMapping_2016(channel, &request_mapping);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Failed to add request mapping.", "Error", MB_OK);

// Remove request mapping using CANID
result = CANTP_RemoveMappings_2016(channel, request_mapping.can_id);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Failed to remove mapping.", "Error", MB_OK);
```

Class method version: RemoveMappings_2016 on page 123

See also: cantp_mapping on page 20, CANTP_RemoveMapping_2016 on page 231, CANTP_AddMapping_2016 on page 229

3.7.8 CANTP_AddFiltering_2016

Adds an entry to the CAN-ID white-list filtering.

Syntax

C++

```
cantp_status __stdcall CANTP_AddFiltering_2016(
       cantp_handle channel,
       uint32_t can_id_from,
       uint32_t can_id_to,
       bool ignore_can_msgtype,
       cantp_can_msgtype can_msgtype);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
can_id_from	The lowest CAN ID wanted to be received.
can_id_to	The highest CAN ID wanted to be received.
ignore_can_msgtype	States if filter should check the CAN message type.
can_msgtype	If ignore_can_msgtype is false, the value states which types of CAN frame should be allowed (see cantp_can_msgtype on page 88).

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_NO_MEMORY	Memory allocation error when add element in the white list.

Example

The following example shows the use of the function CANTP_AddFiltering_2016 the channel PCANTP_HANDLE_USBBUS1. It adds a filter from 0xD1 can identifier to 0xD2 can identifier for standard messages.

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

C++

```
cantp_status result;
result = CANTP_AddFiltering_2016(PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       PCANTP_CAN_MSGTYPE_STANDARD);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Error adding CAN ID filter.", "Error", MB_OK);
```

Class-method version: AddFiltering_2016 on page 126

See also: CANTP_RemoveFiltering_2016 below, RemoveFiltering_2016 on page 128

3.7.9 CANTP_RemoveFiltering_2016

Removes an entry from the CAN-ID white-list filtering.

Syntax

C++

```
cantp_status __stdcall CANTP_RemoveFiltering_2016(
       cantp_handle channel,
       uint32_t can_id_from,
       uint32_t can_id_to,
       bool ignore_can_msgtype,
       cantp_can_msgtype can_msgtype);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
can_id_from	The lowest CAN ID wanted to be removed. (see CANTP_AddFiltering_2016 on page 233)
can_id_to	The highest CAN ID wanted to be removed. (see CANTP_AddFiltering_2016 on page 233)
ignore_can_msgtype	ignore_can_msgtype boolean of the filter to remove. (see CANTP_AddFiltering_2016 on page 233)
can_msgtype	can_msgtype of the filter to remove (see CANTP_AddFiltering_2016 on page 233)

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.
PCANTP_STATUS_NOT_INITIALIZED	The filter to remove is not in the filtering list.

Example

The following example shows the use of the function CANTP_RemoveFiltering_2016 on the channel PCANTP_HANDLE_USBBUS1. This example adds a filter from 0xD1 can identifier to 0xD2 can identifier then removes it.

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

C++

```
cantp_status result;
result = CANTP_AddFiltering_2016(PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       PCANTP_CAN_MSGTYPE_STANDARD);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Error adding filter.", "Error", MB_OK);

// Remove the previously added filter
result = CANTP_RemoveFiltering_2016(PCANTP_HANDLE_USBBUS1, 0xD1, 0xD2, false,
       PCANTP_CAN_MSGTYPE_STANDARD);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
       MessageBox(NULL, "Error removing filter.", "Error", MB_OK);
```

Class-method version: RemoveFiltering_2016 on page 128

See also: CANTP_AddFiltering_2016 on page 233, class-method AddFiltering_2016 on page 126

3.7.10 CANTP_GetValue_2016

Retrieves information from a PCAN channel.

Syntax

C++

```
cantp_status __stdcall CANTP_GetValue_2016(
       cantp_handle channel,
       cantp_parameter parameter,
       void* buffer,
       uint32_t buffer_length);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
parameter	The code of the value to retrieve (see cantp_parameter on page 70).
buffer	The buffer to return the value of the requested parameter.
buffer_length	The length in bytes of the given buffer.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the parameters passed to the function are invalid. Check the value of 'parameter' and assert it is compatible with the buffer length (see cantp_parameter on page 70).

Example

The following example shows the use of the function CANTP_GetValue_2016 on the channel PCANTP_HANDLE_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++

```
cantp_handle channel = PCANTP_HANDLE_USBBUS1;
cantp_status result;
unsigned int iBuffer = 0;
char strMsg[256];

// Gets the value of the ISO-TP Separation Time (STmin) parameter.
result = CANTP_GetValue_2016(channel, PCANTP_PARAMETER_SEPARATION_TIME, &iBuffer,
       sizeof(unsigned int));
if (result != PCANTP_STATUS_OK)
       MessageBox(NULL, "Failed to get value", "Error", MB_OK);
else
{
       sprintf(strMsg, "%d", iBuffer);
       MessageBox(NULL, strMsg, "Success", MB_OK);
}
```

Class method version: GetValue_2016 on page 131

See also: CANTP_SetValue_2016 on page 228, cantp_parameter on page 70, Detailed parameters values on page 75

3.7.11 CANTP_StatusGet_2016

Retrieves the value of a cantp_status subtype (like cantp_errstatus, cantp_busstatus, etc.).

Syntax

C++

```
uint32_t __stdcall CANTP_StatusGet_2016(
       const cantp_status status,
       const cantp_statustype type);
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).
type	The type of status to filter (see cantp_statustype on page 51).

Returns

The return is the value of the enumeration matching the requested type.

Example

The following example shows the use of the function CANTP_StatusGet_2016 on a status from CANTP_Uninitialize_2016 on an uninitialized channel. The goal is to generate a PCANTP_STATUS_NOT_INITIALIZED status).

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

C++

```
char str_msg[256];
cantp_status result = CANTP_Uninitialize_2016(PCANTP_HANDLE_USBBUS1);

// Check general error status: should throw PCANTP_STATUS_NOT_INITIALIZED (=1)
uint32_t general_error = CANTP_StatusGet_2016(result, PCANTP_STATUSTYPE_ERR);
if (general_error != PCANTP_ERRSTATUS_OK) {
```

```
        sprintf(str_msg, "General error code on uninitialized: %d", general_error);
        MessageBox(NULL, str_msg, "Success", MB_OK);
}

// Check network error status: should be PCANTP_STATUS_OK
uint32_t network_error = CANTP_StatusGet_2016(result, PCANTP_STATUSTYPE_NET);
if (network_error != PCANTP_NETSTATUS_OK) {
        MessageBox(NULL, "Network error!", "Error", MB_OK);
}
```

Class-method version: StatusGet_2016 on page 150

See also: cantp_status on page 60, cantp_statustype on page 51

3.7.12 CANTP_StatusIsOk_2016

Checks if a cantp_status matches an expected result (default is PCANTP_STATUS_OK).

Syntax

C++

```
bool __stdcall CANTP_StatusIsOk_2016(
        const cantp_status status,
        const cantp_status status_expected = PCANTP_STATUS_OK,
        bool strict = false);
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).
status_expected	The expected status (see cantp_status on page 60). The default value is PCANTP_STATUS_OK.
strict	Enable strict mode (default is false). Strict mode ensures that bus or extra information are the same.

Returns

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

Remarks

When comparing a cantp_status, it is preferred to use CANTP_StatusIsOk_2016 instead of comparing it with the "==" operator because CANTP_StatusIsOk_2016 can remove information flag.

Example

The following example shows the use of the function CANTP_StatusIsOk_2016 after initializing the channel PCANTP_HANDLE_USBBUS1.

C++

```
cantp_status result;

// The Plug and Play channel USB1 is initialized.
result = CANTP_Initialize_2016(PCANTP_HANDLE_USBBUS1, PCANTP_BAUDRATE_500K);
if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
        MessageBox(NULL, "PCAN-USB (Ch-1) was initialized", "Success", MB_OK);
else
        MessageBox(NULL, "Initialization failed", "Error", MB_OK);
```

Class-method version: StatusIsOk_2016 on page 154

See also: cantp_status on page 60

3.7.13 CANTP_GetMsgProgress_2016

Gets progress information on a specific message.

Syntax

C++

```
cantp_status __stdcall CANTP_GetMsgProgress_2016(
        cantp_handle channel,
        cantp_msg* msg_buffer,
        cantp_msgdirection direction,
        cantp_msgprogress* msgprogress_buffer);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37).
msg_buffer	A cantp_msg structure buffer matching the message to look for (see cantp_msg on page 28).
direction	The expected direction (incoming/outgoing) of the message (see cantp_msgdirection on page 96).
msgprogress_buffer	A cantp_msgprogress structure buffer to store the progress information (see cantp_msgprogress on page 31).

Returns

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

PCANTP_STATUS_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_STATUS_PARAM_INVALID_VALUE	The cantp_msg message or the cantp_msgprogress buffer is invalid.
PCANTP_STATUS_NO_MESSAGE	The message is unknown.
PCANTP_STATUS_LOCK_TIMEOUT	Internal lock timeout while searching the message within internal queues.

Example

The following example shows the use of the function CANTP_GetMsgProgress_2016 when receiving a loopback message on the PCANTP channel USB 1. Depending on the result, progress will be shown to the user.

Note: It is assumed that the channel was already initialized and a heavy ISOTP message has been sent.

C++

```
char str_msg[256];
cantp_msg loopback_msg = {};
cantp_msgprogress progress = {};
cantp_status result;

// Read transmission confirmation.
result = CANTP_Read_2016(PCANTP_HANDLE_USBBUS1, &loopback_msg);
if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK)
        && (PCANTP_MSGTYPE_ISOTP & loopback_msg.type) == PCANTP_MSGTYPE_ISOTP
        && ((loopback_msg.msgdata.isotp->netaddrinfo.msgtype
                & PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX)
                == PCANTP_ISOTP_MSGTYPE_FLAG_INDICATION_TX)) {

        // The message is being received, wait and show progress
        do {
                result = CANTP_GetMsgProgress_2016(PCANTP_HANDLE_USBBUS1, &loopback_msg,
                        PCANTP_MSGDIRECTION_TX, &progress);
```

```
                sprintf(str_msg, "TX Progress on loopback message: %d%%", progress.percentage);
                MessageBox(NULL, str_msg, "Success", MB_OK);
        } while (progress.state == PCANTP_MSGPROGRESS_STATE_PROCESSING);
}
else {
        sprintf(str_msg, "Read error: %d", result);
        MessageBox(NULL, str_msg, "Error", MB_OK);
}
```

Class-method version: GetMsgProgress_2016 on page 143

See also: cantp_msgprogress on page 31, cantp_msgprogress_state on page 95

3.7.14 CANTP_GetErrorText_2016

Gets a descriptive text for a given cantp_status error code.

Syntax

C++

```
cantp_status __stdcall CANTP_GetErrorText_2016(
        cantp_status error,
        uint16_t language,
        char* buffer,
        uint32_t bufferSize);
```

Parameters

Parameters	Description
error	A cantp_status error code (see cantp_status on page 60).
language	The current languages available for translation are: Neutral (0x00), German (0x07), English (0x09), Spanish (0x0A), Italian (0x10) and French (0x0C).
buffer	A buffer for a null-terminated char array.
bufferSize	Buffer length in bytes.

Returns

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

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

Remarks

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

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

Note: If the buffer is too small for the resulting text, the error 0x80008000 (PCANTP_STATUS_MASK_PCAN|PCAN_ERROR_ILLPARAMVAL) is returned. Even when only short texts are being currently returned, a text

within this function can have a maximum of 255 characters. For this reason, it is recommended to use a buffer with a length of at least 256 bytes.

Example

The following example shows the use of the function CANTP_GetErrorText_2016 to get the description of an error. The language of the description's text will be the same used by the operating system (if its language is supported; otherwise English is used).

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

C++

```
char str_msg[256];
cantp_status result;
cantp_status error_result;
error_result = CANTP_Uninitialize_2016(PCANTP_HANDLE_USBBUS1);
result = CANTP_GetErrorText_2016(error_result, 0x0, str_msg, 256);
if(CANTP_StatusIsOk_2016(result))
       MessageBox(NULL, str_msg, "Error on uninitialized", MB_OK);
```

Class-method version: GetErrorText_2016 on page 138

See also: cantp_status on page 60

3.7.15 CANTP_GetCanBusStatus_2016

Gets information about the internal BUS status of a PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_GetCanBusStatus_2016(
       cantp_handle channel);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)

Returns

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

PCANTP_STATUS_OK	Indicates that the status of the given PCANTP channel is OK.
PCANTP_STATUS_FLAG_BUS_LIGHT	Indicates a bus error within the given PCANTP channel. The hardware is in bus-light status.
PCANTP_STATUS_FLAG_BUS_HEAVY	Indicates a bus error within the given PCANTP channel. The hardware is in bus-heavy status.
PCANTP_STATUS_FLAG_BUS_OFF	Indicates a bus error within the given PCANTP channel. The hardware is in bus-off status.
PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.

Remarks

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

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

Example

The following example shows the use of the function CANTP_GetCanBusStatus_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
cantp_status result;
result = CANTP_GetCanBusStatus_2016(PCANTP_HANDLE_PCIBUS1);

// Checks the status of the PCI channel.
switch (result)
{
case PCANTP_STATUS_FLAG_BUS_LIGHT:
       MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-LIGHT status...", "Success", MB_OK);
       break;
case PCANTP_STATUS_FLAG_BUS_HEAVY:
       MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-HEAVY status...", "Success", MB_OK);
       break;
case PCANTP_STATUS_FLAG_BUS_OFF:
       MessageBox(NULL, "PCAN-PCI (Ch-1): Handling a BUS-OFF status...", "Success", MB_OK);
       break;
case PCANTP_STATUS_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;
}
```

Class-method version: GetCanBusStatus_2016 on page 140

See also: cantp_status on page 60

3.7.16 CANTP_GetMappings_2016

Retrieves all the mappings defined for a given PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_GetMappings_2016(
       cantp_handle channel,
       cantp_mapping* buffer,
       uint32_t* buffer_length);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
Buffer	A buffer to store an array of cantp_mapping (see cantp_mapping on page 20).
buffer_length	(In) The number of cantp_mapping element the buffer can store. (Out) The actual number of elements copied in the buffer.

Returns

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

PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of initialized channels of the calling application.
PCANTP_STATUS_PARAM_INVALID_VALUE	The buffer or the size is invalid.
PCANTP_STATUS_PARAM_BUFFER_TOO_SMALL	The given buffer is to small to store all mappings.

Example

The following example shows the use of the function CANTP_GetMappings_2016 on PCANTP_HANDLE_USBBUS1. It displays all mappings added on the channel.

Note: It is assumed that the channel and some mappings were already initialized.

C++

```
cantp_status result;
uint32_t count = 256;
cantp_mapping mappings[256];
result = CANTP_GetMappings_2016(PCANTP_HANDLE_USBBUS1, mappings, &count);
if (CANTP_StatusIsOk_2016(result)) {
    for (int i = 0; i < count; i++) {
        std::cout << "mappings[" << i << "]:";
        std::cout << "\n\t- can id: " << mappings[i].can_id;
        std::cout << "\n\t- can id flow control: " << mappings[i].can_id_flow_ctrl;
        std::cout << "\n\t- can message type: " << mappings[i].can_msgtype;
        std::cout << "\n\t- extension address: " << mappings[i].netaddrinfo.extension_addr;
        std::cout << "\n\t- addressing format: " << mappings[i].netaddrinfo.format;
        std::cout << "\n\t- isotp message type: " << mappings[i].netaddrinfo.msgtype;
        std::cout << "\n\t- source address: " << mappings[i].netaddrinfo.source_addr;
        std::cout << "\n\t- target address: " << mappings[i].netaddrinfo.target_addr;
        std::cout << "\n\t- target type: " << mappings[i].netaddrinfo.target_type << "\n";
    }
}
else {
    std::cout << "Failed to get mappings: " << result << "\n";
}
```

Class-method version: GetMappings_2016 on page 147

See also: cantp_mapping on page 20, CANTP_AddMapping_2016 on page 229

3.7.17 CANTP_StatusListTypes_2016

Lists the subtypes contained in the PCANTP status.

Syntax

C++

```
cantp_statustype __stdcall CANTP_StatusListTypes_2016(
       const cantp_status status);
```

Parameters

Parameters	Description
status	The status to analyze (see cantp_status on page 60).

Returns

An aggregation of cantp_statustype values.

Example

The following example shows the use of the function CANTP_StatusListTypes_2016 on the channel PCANTP_HANDLE_USBBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
char str_msg[256];
cantp_status result;
cantp_statustype statustype;
result = CANTP_Uninitialize_2016(PCANTP_HANDLE_USBBUS1);
if (!CANTP_StatusIsOk_2016(result,PCANTP_STATUS_OK,false)) {
       statustype = CANTP_StatusListTypes_2016(result);

       // Expected type: general error (=1)
       sprintf(str_msg, "Uninitialize error type: %d", statustype);
       MessageBox(NULL, str_msg, "Error", MB_OK);
}
```

Class-method version: StatusListTypes_2016 on page 152

See also: cantp_statustype on page 51, cantp_status on page 60

3.7.18 CANTP_Read_2016

Reads a CANTP message from the receive queue of a PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_Read_2016(
       cantp_handle channel,
       cantp_msg* msg_buffer,
       cantp_timestamp* timestamp_buffer = 0,
       cantp_msgtype msg_type = PCANTP_MSGTYPE_ANY);
```

Parameters

Parameters	Description
Channel	The handle of a PCANTP channel (see cantp_handle on page 37).
msg_buffer	A cantp_msg buffer to store the CANTP message. (see cantp_msg on page 28).
timestamp_buffer	A cantp_timestamp structure buffer to get the reception time of the message. If this value is not desired, this parameter should be passed as NULL (see cantp_timestamp on page 36).
msg_type	A cantp_msgtype structure buffer to filter the message to read. By default, accept any message type (see cantp_msgtype on page 85).

Returns

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

PCANTP_STATUS_NO_MESSAGE	Indicates that the receive queue of the channel is empty.
PCANTP_STATUS_NOT_INITIALIZED	Indicates that the given PCANTP channel was not found in the list of reserved channels of the calling application.

Remarks

- In addition to checking cantp_status code, the cantp_netstatus field should be checked as it contains the network status of the message (see cantp_msg on page 28, cantp_msgdata on page 22 and cantp_netstatus on page 53).
- In case of ISOTP message, the message type contained in the message cantp_netaddrinfo should be checked too as it indicates if the message is a complete ISO-TP message (diagnostic, remote diagnostic, and a pending message flag) (see cantp_msg on page 28, cantp_msgdata_isotp on page 26 and cantp_isotp_msgtype on page 90).
- Specifying the value of "NULL" for the parameter timestamp_buffer causes reading a message without timestamp, when the reception time is not desired.
- The message structure is automatically allocated and initialized in CANTP_Read_2016 function. So once the message processed, the structure must be uninitialized (see CANTP_MsgDataFree_2016 on page 249).

Example

The following example shows the use of the function CANTP_Read_2016 on the channel PCANTP_HANDLE_USBBUS1. Depending on the result, a message will be shown to the user. This example is basic, the proper way to handle message reception is Using Events (see on page 259).

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

C++

```
cantp_status result;
cantp_msg msg = {};
bool bStop = false;

do
{
	// Safely zero-initialize message buffer.
	CANTP_MsgDataAlloc(&msg, PCANTP_MSGTYPE_NONE);
	// Reads the first message in the queue.
	result = CANTP_Read_2016(PCANTP_HANDLE_USBBUS1, &msg);
	if (result == PCANTP_STATUS_OK)
	{
		// Processes the received message.
		MessageBox(NULL, "A message was received", "Success", MB_OK);

		//ProcessMessage(msg);

		// Free allocated memory
		CANTP_MsgDataFree_2016(&msg);
	}
	else
	{
		// An error occurred.
		MessageBox(NULL, "An error ocured", "Error", MB_OK);
		// Here can be decided if the loop has to be terminated.
		//bStop = HandleReadError(result);
	}
} while (!bStop);
```

Class method version: Read_2016 on page 160

See also: cantp_msg on page 28, CANTP_Write_2016 on page 245

More examples: see API "examples" folder and iisotp_read_write example.

3.7.19 CANTP_Write_2016

Transmits a CANTP message.

Syntax

C++

```
cantp_status __stdcall CANTP_Write_2016(
        cantp_handle channel,
        cantp_msg* msg_buffer);
```

Parameters

Parameters	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)
msg_buffer	A cantp_msg buffer containing the CANTP message to be sent. (see cantp_msg on page 28)

Returns

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

PCANTP_STATUS_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_STATUS_PARAM_INVALID_VALUE	The message is not a valid message.
PCANTP_STATUS_PARAM_INVALID_TYPE	The message type is not valid.
PCANTP_STATUS_MAPPING_NOT_INITIALIZED	The mapping is unknown.

Remarks

The CANTP_Write_2016 function does 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 (see cantp_netstatus on page 53).

Example

The following example shows the use of the function CANTP_Write_2016 on the channel PCANTP_HANDLE_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, request_mapping was configured (see CANTP_AddMapping_2016 on page 229) and receive_event is set (see Using Events on page 259).

C++

```
cantp_status result;
cantp_msg request_msg = {};
cantp_msg loopback_msg = {};
char str_msg[256];
int wait_result;

// Allocate message structure
result = CANTP_MsgDataAlloc_2016(&request_msg, PCANTP_MSGTYPE_ISOTP);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
        sprintf(str_msg, "Message allocation error: %d", result);
        MessageBox(NULL, str_msg, "Error", MB_OK);
}

// Prepare an ISO-TP message containing 3 bytes of raw data.
result = CANTP_MsgDataInit_2016(&request_msg, request_mapping.can_id,
```

```
        request_mapping.can_msgtype, 3, NULL, &request_mapping.netaddrinfo);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK)) {
        sprintf(str_msg, "Message initialization error: %d", result);
        MessageBox(NULL, str_msg, "Error", MB_OK);
}

// The message is sent using the PCAN-USB.
result = CANTP_Write_2016(PCANTP_HANDLE_USBBUS1, &request_msg);
if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK))
{
        // Read the transmission confirmation.
        wait_result = WaitForSingleObject(receive_event, 5000);
        if (wait_result == WAIT_OBJECT_0) {
                result = CANTP_Read_2016(PCANTP_HANDLE_USBBUS1, &loopback_msg);
                if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK)) {
                        sprintf(str_msg, "Read = %d, type=%d, netstatus=%d", result,
                                loopback_msg.type, loopback_msg.msgdata.any->netstatus);
                        MessageBox(NULL, str_msg, "Read", MB_OK);
                }
                else {
                        sprintf(str_msg, "Read error: %d", result);
                        MessageBox(NULL, str_msg, "Error", MB_OK);
                }
        }
}
else
{
        sprintf(str_msg, "Write error: %d", result);
        MessageBox(NULL, str_msg, "Error", MB_OK);
}
```

Class method version: Write_2016 on page 170

See also: cantp_msg on page 28, CANTP_Read_2016 on page 243

More examples: see API “examples” folder and iisotp_read_write example.

3.7.20 CANTP_Reset_2016

Resets the receive and transmit queues of a PCANTP channel.

Syntax

C++

```
cantp_status __stdcall CANTP_Reset_2016(
        cantp_handle channel);
```

Parameters

Parameter	Description
channel	The handle of a PCANTP channel (see cantp_handle on page 37)

Returns

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

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

Remarks

This function 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_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
cantp_status result;

// The PCI channel 1 is reset.
result = CANTP_Reset_2016(PCANTP_HANDLE_PCIBUS1);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false))
       MessageBox(NULL, "An error occured", "Error", MB_OK);
else
       MessageBox(NULL, "PCAN-PCI (Ch-1) was reset", "Success", MB_OK);
```

Class-method version: Reset_2016 on page 175

See also: CANTP_Uninitialize_2016 on page 227

3.7.21 CANTP_MsgDataAlloc_2016

Allocates a CANTP message based on the given type.

Syntax

C++

```
cantp_status __stdcall CANTP_MsgDataAlloc_2016(
       cantp_msg* msg_buffer,
       cantp_msgtype type);
```

Parameters

Parameters	Description
msg_buffer	A cantp_msg structure buffer (see cantp_msg on page 28). It will be freed if required.
type	Type of the message to allocate (see cantp_msgtype on page 85).

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	Indicates that the given message is not valid.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory for the given message.

Remarks

In case of handling reception messages, it is not mandatory to allocate and initialize message. Indeed, the message allocation is automatically realized within the function CANTP_Read_2016. Yet to prevent random memory artifacts, it is recommended to call the function with the type PCANTP_MSGTYPE_NONE: the function will make sure to zero-initialize the buffer.

Once allocated, a message should be initialized by calling the function CANTP_MsgDataInit_2016. Then freed using the function CANTP_MsgDataFree_2016.

Example

The following example shows the use of the function CANTP_MsgDataAlloc_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
cantp_status result;
cantp_msg msg = {};
char str_msg[256];

// Allocate message structure
result = CANTP_MsgDataAlloc_2016(&msg, PCANTP_MSGTYPE_ISOTP);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
	sprintf(str_msg, "Message allocation error: %d", result);
	MessageBox(NULL, str_msg, "Error", MB_OK);
}
else {
	MessageBox(NULL, "Message is allocated!", "Success", MB_OK);
}
```

Class-method version: MsgDataAlloc_2016 on page 186

See also: cantp_msg on page 28, CANTP_MsgDataInit_2016 below, CANTP_MsgDataFree_2016 on page 249

3.7.22 CANTP_MsgDataInit_2016

Initializes an allocated CANTP message.

Syntax

C++

```
cantp_status __stdcall CANTP_MsgDataInit_2016(
	cantp_msg* msg_buffer,
	uint32_t can_id,
	cantp_can_msgtype can_msgtype,
	uint32_t data_length,
	const void* data,
	cantp_netaddrinfo* netaddrinfo = 0);
```

Parameters

Parameters	Description
msg_buffer	An allocated cantp_msg structure buffer (see cantp_msg on page 28 and CANTP_MsgDataAlloc_2016 on page 247).
can_id	CAN identifier (ISO-TP message may ignore this parameter and use PCANTP_MAPPING_FLOW_CTRL_NONE (-1)).
can_msgtype	Combination of CAN message types (like "extended CAN ID", "FD", "RTR", etc. flags). See cantp_msgtype on page 85.
data_length	The length in bytes of the data.
data	A buffer to initialize the message's data with. If NULL, message's data is initialized with zeros.
netaddrinfo	Network address information of the ISO-TP message (see cantp_netaddrinfo on page 19). Only valid with an ISO-TP message.

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	One of the given parameters is not valid (null message, bad canid, incorrect length depending on the message type...).
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory during initialization.

Example

The following example shows the use of the function CANTP_MsgDataInit_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

Note: It is assumed that the channel and the mapping were already initialized.

C++

```
cantp_status result;
cantp_msg request_msg = {};
char str_msg[256];

result = CANTP_MsgDataAlloc_2016(&request_msg, PCANTP_MSGTYPE_ISOTP);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {

	// Initialize the allocated message with "PEAK" as data
	result = CANTP_MsgDataInit_2016(&request_msg, request_mapping.can_id,
		request_mapping.can_msgtype, 4, "PEAK", &request_mapping.netaddrinfo);
	if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
		sprintf(str_msg, "Message initialization error: %d", result);
		MessageBox(NULL, str_msg, "Error", MB_OK);
	}
}
```

Class-method version: MsgDataInit_2016 on page 188

See also: cantp_msg on page 28, CANTP_MsgDataAlloc_2016 on page 247, CANTP_MsgDataFree_2016 below.

3.7.23 CANTP_MsgDataFree_2016

Deallocates a CANTP message.

Syntax

C++

```
cantp_status __stdcall CANTP_MsgDataFree_2016(
	cantp_msg* msg_buffer);
```

Parameters

Parameter	Description
msg_buffer	An allocated cantp_msg structure (see cantp_msg on page 28 and CANTP_MsgDataAlloc_2016 on page 247).

Returns

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

PCANTP_STATUS_CAUTION_BUFFER_IN_USE	The message structure is currently in use. It cannot be deleted.
PCANTP_STATUS_PARAM_INVALID_VALUE	The message is not valid.

Example

The following example shows the use of the function CANTP_MsgDataFree_2016 on the channel PCANTP_HANDLE_PCIBUS1. Depending on the result, a message will be shown to the user.

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

C++

```
cantp_status result;
cantp_msg msg = {};
char str_msg[256];

// Allocate message structure
result = CANTP_MsgDataAlloc_2016(&msg, PCANTP_MSGTYPE_ISOTP);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
       sprintf(str_msg, "Message allocation error: %d", result);
       MessageBox(NULL, str_msg, "Error", MB_OK);
}

// Process message …

// Free message structure
result = CANTP_MsgDataFree_2016(&msg);
if (!CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
       sprintf(str_msg, "Message free error: %d", result);
       MessageBox(NULL, str_msg, "Error", MB_OK);
}
```

Class-method version: MsgDataFree_2016 on page 196

See also: cantp_msg on page 28, CANTP_MsgDataAlloc_2016 on page 247, CANTP_MsgDataInit_2016 on page 248

3.7.24 CANTP_MsgEqual_2016

Checks if two CANTP messages are equal.

Syntax

C++

```
bool __stdcall CANTP_MsgEqual_2016(
       const cantp_msg* msg_buffer1,
       const cantp_msg* msg_buffer2,
       bool ignoreSelfReceiveFlag);
```

Parameters

Parameters	Description
msg_buffer1	A cantp_msg structure buffer (see cantp_msg on page 28).
msg_buffer2	Another cantp_msg structure buffer to compare with first parameter (see cantp_msg on page 28).
ignoreSelfReceiveFlag	States if comparison should ignore loopback flag (i.e if true the function will return true when comparing a request and its loopback confirmation).

Returns

The return value is a boolean. It is true If the messages are the same or false if they are not.

Remarks

If one message is the indication of an incoming/outgoing ISO-TP message, the actual data-content will not be compared. In that case the function checks if the messages' network address information matches.

Example

The following example shows the use of the function CANTP_MsgEqual_2016. It allocates and initializes a first message structure, copies it in a second structure then checks that the two structures are the same.

Note: It is assumed that the channel and the mapping were already initialized.

C++

```
cantp_status result;
cantp_msg msg_1;
cantp_msg msg_2 = {};

// Initialize the first message
result = CANTP_MsgDataAlloc_2016(&msg_1, PCANTP_MSGTYPE_ISOTP);
if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
	result = CANTP_MsgDataInit_2016(&msg_1, request_mapping.can_id,
		request_mapping.can_msgtype, 4, "PEAK", &request_mapping.netaddrinfo);
	if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {

		// Copy msg_1 in msg_2
		result = CANTP_MsgCopy_2016(&msg_2, &msg_1);
		if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {

			// Compare msg_1 and msg_2, should be the same
			if (CANTP_MsgEqual_2016(&msg_1, &msg_2, false))
				MessageBox(NULL, "msg_1 and msg_2 are the same!", "Success", MB_OK);
			else
				MessageBox(NULL, "msg_1 and msg_2 are different!", "Error", MB_OK);
		}
	}
}
```

Class-method version: MsgEqual_2016 on page 177

See also: cantp_msg on page 28, CANTP_MsgCopy_2016 below

3.7.25 CANTP_MsgCopy_2016

Copies a CANTP message to another buffer.

Syntax

C++

```
cantp_status __stdcall CANTP_MsgCopy_2016(
	cantp_msg* msg_buffer_dst,
	const cantp_msg* msg_buffer_src);
```

Parameters

Parameters	Description
msg_buffer_dst	A cantp_msg structure buffer to store the copied message (cantp_msg on page 28).
msg_buffer_src	The cantp_msg structure buffer used as the source (see cantp_msg on page 28).

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	One of the given messages is not valid.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory during the copy.

Example

The following example shows the use of the function CANTP_MsgCopy_2016. It allocates and initializes a first message structure, copies it in a second structure then checks that the two structures are the same.

Note: It is assumed that the channel and the mapping were already initialized.

C++

```
cantp_status result;
cantp_msg msg_1;
cantp_msg msg_2 = {};

// Initialize the first message
result = CANTP_MsgDataAlloc_2016(&msg_1, PCANTP_MSGTYPE_ISOTP);
if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
	result = CANTP_MsgDataInit_2016(&msg_1, request_mapping.can_id,
		request_mapping.can_msgtype, 4, "PEAK", &request_mapping.netaddrinfo);
	if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {

		// Copy msg_1 in msg_2
		result = CANTP_MsgCopy_2016(&msg_2, &msg_1);
		if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {

			// Compare msg_1 and msg_2, should be the same
			if (CANTP_MsgEqual_2016(&msg_1, &msg_2, false))
				MessageBox(NULL, "msg_1 and msg_2 are the same!", "Success", MB_OK);
			else
				MessageBox(NULL, "msg_1 and msg_2 are different!", "Error", MB_OK);
		}
	}
}
```

Class-method version: MsgCopy_2016 on page 180

See also: cantp_msg on page 28, CANTP_MsgEqual_2016 on page 250

3.7.26 CANTP_MsgDlcToLength_2016

Converts a CAN DLC to its corresponding length.

Syntax

C++

```
uint32_t __stdcall CANTP_MsgDlcToLength_2016(
	const uint8_t dlc);
```

Parameters

Parameter	Description
dlc	The Data Length Code (DLC) to convert.

Returns

The corresponding length of the dlc parameter.

Example

The following example shows the use of the function CANTP_MsgDlcToLength_2016.

C++

```
char str_msg[256];
uint8_t dlc = 10;
uint32_t length;
length = CANTP_MsgDlcToLength_2016(dlc);
sprintf(str_msg, "For dlc=%d, length=%d", dlc, length);
MessageBox(NULL, str_msg, "Info", MB_OK);
```

Class-method version: MsgDlcToLength_2016 on page 183

See also: CANTP_MsgLengthToDlc_2016 below

3.7.27 CANTP_MsgLengthToDlc_2016

Converts a data length to a corresponding CAN DLC.

Syntax

C++

```
uint8_t __stdcall CANTP_MsgLengthToDlc_2016(
       const uint32_t length);
```

Parameters

Parameters	Description
length	The length to convert.

Returns

The smallest DLC that can hold the requested length (0x00-0x0F).

Remarks

The returned DLC can hold more data that the requested length.

Example

The following example shows the use of the function CANTP_MsgLengthToDlc_2016.

C++

```
char str_msg[256];
uint8_t dlc;
uint32_t length = 16;
dlc = CANTP_MsgLengthToDlc_2016(length);
sprintf(str_msg, "For length=%d, dlc=%d", length, dlc);
MessageBox(NULL, str_msg, "Info", MB_OK);
```

Class-method version: MsgLengthToDlc_2016 on page 184

See also: CANTP_MsgDlcToLength_2016 on page 252

3.7.28 CANTP_MsgDataInitOptions_2016

Initializes several options for the CANTP message that will override the channel's parameter(s).

Syntax

C++

```
cantp_status __stdcall CANTP_MsgDataInitOptions_2016(
        cantp_msg* msg_buffer,
        uint32_t nb_options);
```

Parameters

Parameters	Description
msg_buffer	An allocated cantp_msg structure buffer (see cantp_msg on page 28).
nb_options	Number of options to initialize.

Returns

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

PCANTP_STATUS_PARAM_INVALID_VALUE	The given message is not a valid message.
PCANTP_STATUS_NO_MEMORY	Failed to allocate memory.

Example

The following example shows the use of the function CANTP_MsgDataInitOptions_2016. It sets a new priority for an already initialized message.

Note: It is assumed that the channel and the message structure were already initialized.

C++

```
cantp_status result;

// Set priority
result = CANTP_MsgDataInitOptions_2016(&request_msg, 1);
if (CANTP_StatusIsOk_2016(result, PCANTP_STATUS_OK, false)) {
        request_msg.msgdata.any->options->buffer[0].name = PCANTP_OPTION_J1939_PRIORITY;
        request_msg.msgdata.any->options->buffer[0].value = 0x02;
        MessageBox(NULL, "Set message priority.", "Info", MB_OK);
}
```

Class-method version: MsgDataInitOptions_2016 on page 194

See also: cantp_msgoption on page 16, cantp_option on page 93

4 Additional Information

PCAN is the platform for PCAN-OBD-2, PCAN-UDS, and PCAN-Basic. In the following topics there is an overview of PCAN and the fundamental practice with the interface DLL CanApi2 (PCAN-API).

Topics	Description
PCAN Fundamentals	This section contains an introduction to PCAN.
PCAN-Basic	This section contains general information about the PCAN-Basic API.
PCAN-API	This section contains general information about the PCAN-API.
ISO-TP Network Addressing	This section contains general information about the ISO-TP network addressing format.

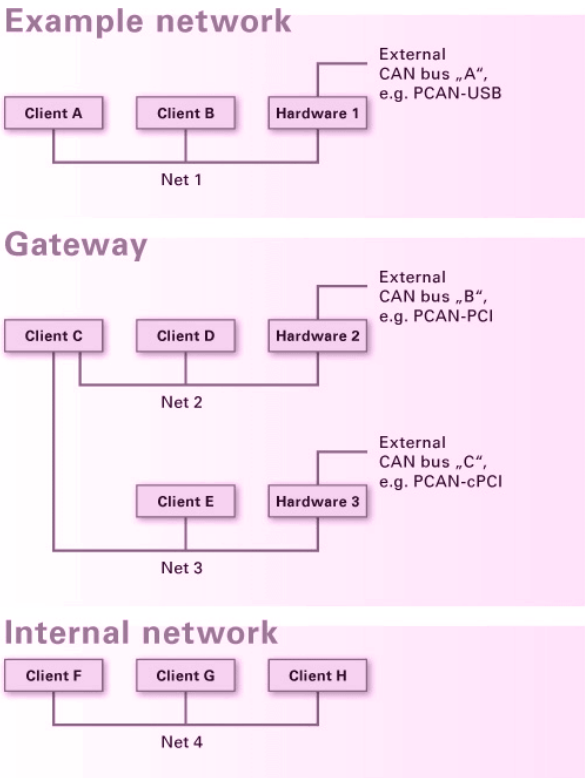
4.1 PCAN Fundamentals

PCAN is a synonym for PEAK CAN APPLICATIONS and is a flexible system for planning, developing, and using a CAN bus system. Developers as well as end users are getting a helpful and powerful product.

Basis for the communication between PCs and external hardware via CAN is a series of Windows kernel-mode drivers (virtual device drivers) e.g. `PCAN_USB.SYS`, `PCAN_PCI.SYS`, and `PCAN_xxx.SYS`. These drivers are the core of a complete CAN environment on a PC running Windows and work as interfaces between CAN software and PC-based CAN hardware. The drivers manage the entire data flow of every CAN device connected to the PC.

A user or administrator of a CAN installation gets access via the PCAN clients (short: clients). Several parameters of processes can be visualized and changed with their help. The drivers allow the connection of several clients at the same time.

Furthermore, several hardware components based on the SJA1000 CAN controller are supported by a PCAN driver. So-called nets provide the logical structure for CAN busses, which are virtually extended into the PC. On the hardware side, several Clients can be connected, too. The following figures demonstrate different possibilities of net configurations (also realizable at the same time).



Following rules apply to PCAN clients, nets, and hardware:

- One client can be connected to several nets.
- One net provides several clients.
- One piece of hardware belongs to one net.
- One net can include none or one piece of hardware.
- A message from a transmitting client is carried on to every other connected client and to the external bus via the connected CAN hardware.
- A message received by the CAN hardware is received by every connected client. However, clients react only on those messages that pass their acceptance filter

Users of PCAN-View 3 do not have to define and manage nets. If PCAN-View is instructed to connect directly to a PCAN hardware, the application automatically creates a net for the selected hardware and automatically establishes a connection with this net.

See also: PCAN-Basic below, ISO-TP Network Addressing Format on page 259

4.2 PCAN-Basic

PCAN-Basic is an Application Programming Interface for the use of a collection of Windows Device Drivers from PEAK-System, which allow the real-time connection of Windows applications to all CAN busses physically connected to a PC.

PCAN-Basic principal characteristics are:

- Retrieves information about the receive time of a CAN message.
- Easy switching between different PCAN channels (PCAN-PC hardware)
- The possibility to control some parameters in the hardware, e.g. "Listen-Only" mode, automatic reset of the CAN controller, etc.
- The use of event notifications for faster processing of incoming CAN messages
- An improved system for debugging operations
- The use of only one Dynamic Link Library (`PCANBasic.DLL`) for all supported hardware
- The possibility to connect more than two channels per PCAN-Device. The following list shows the PCAN channels that can be connected per PCAN-Device:

	PCAN-ISA	PCAN-Dongle	PCAN-PCI	PCAN-USB	PCAN-PC-Card	PCAN-LAN
Number of channels	8	1	16	16	2	16

Using the PCAN-Basic

The PCAN-basic offers the possibility to use several PCAN channels within the same application in an easy way. The communication process is divided in three phases: initialization, interaction, and finalization of a PCAN-channel.

Initialization: In order to do CAN communication using a channel, it is necessary to first initialize it. This is done making a call to the function CAN_Initialize (**class method version**: Initialize) or CAN_InitializeFD (**Class method version**: InitializeFD) in case FD communication is desired.

Interaction: After a successful initialization, a channel is ready to communicate with the connected CAN bus. Further configuration is not needed. The functions CAN_Read and CAN_Write (**class method versions**: Read and Write) can be then used to read and write CAN messages. If the channel being used is FD capable and it was initialized using CAN_InitializedFD, then the functions to use are CAN_ReadFD and CAN_WriteFD (**Class method versions**: ReadFD and WriteFD). If desired, extra configuration can be made to improve a communication session, like changing the message filter to target specific messages.

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

Hardware and Drivers

Overview of the current PCAN hardware and device drivers:

Hardware	Plug and Play hardware	Driver
PCAN-Dongle	No	Pcan_dng.sys
PCAN-ISA	No	Pcan_isa.sys
PCAN-PC/104	No	Pcan_isa.sys
PCAN-PCI	Yes	Pcan_pci.sys
PCAN-PCI Express	Yes	Pcan_pci.sys
PCAN-cPCI	Yes	Pcan_pci.sys
PCAN-miniPCI	Yes	Pcan_pci.sys
PCAN-PC/104-Plus	Yes	Pcan_pci.sys
PCAN-USB	Yes	Pcan_usb.sys
PCAN-USB Pro	Yes	Pcan_usb.sys
PCAN-PC Card	Yes	Pcan_pcc.sys

See also: PCAN Fundamentals on page 255, PCAN-API below, ISO-TP Network Addressing Format on page 259

4.3 PCAN-API

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

Important difference to PCAN-Basic:

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

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

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

Fields configuaration, configuration functions for nets and hardware:

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

Fields client, functions for the management of the clients:

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

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

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

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

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

See also: PCAN Fundamentals on page 255, PCAN-Basic on page 256, ISO-TP Network Addressing Format ISO-TP Network Addressing on page 259

4.4 ISO-TP Network Addressing Format

ISO-TP specifies three addressing formats to exchange data: normal, extended, and mixed addressing. Each addressing requires a different number of CAN frame data bytes to encapsulate the addressing information associated with the data to be exchanged.

The following table sums up the mandatory configuration to the ISO-TP API for each addressing format:

Addressing format	CAN ID length	Mandatory configuration steps
Normal addressing PCANTP_ISOTP_FORMAT_NORMAL	11 bits	Define mappings with CANTP_AddMapping.
	29 bits	Define mappings with CANTP_AddMapping.
Normal fixed addressing PCANTP_ISOTP_FORMAT_FIXED_NORMAL	11 bits	Addressing is invalid.
	29 bits	-
Extended addressing PCANTP_ISOTP_FORMAT_EXTENDED	11 bits	Define mappings with CANTP_AddMapping.
	29 bits	Define mappings with CANTP_AddMapping.
Mixed addressing PCANTP_ISOTP_FORMAT_MIXED	11 bits	Define mappings with CANTP_AddMapping.
	29 bits	-
Enhanced addressing PCANTP_ISOTP_FORMAT_ENHANCED	11 bits	Addressing is invalid.
	29 bits	- Note with ISO-15765:2016, this addressing is considered deprecated and disabled by default. See 3.5.14 cantp_parameter PCANTP_PARAMETER_SUPPORT_29B_ENHANCED)

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_2016 function (class method version: CANTP_SetValue_2016) to set the parameter PCANTP_PARAMETER_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_2016 function (class method version: Read_2016), 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_2016 on page 228, **class method version:** SetValue_2016 on page 110

CANTP_Read_2016 on page 243, **class method version:** Read_2016 on page 160