🕇 Home

 $\langle \rangle$

PCAN-Basic Documentation



Welcome to the documentation of PCAN-Basic, the new small Version of the PCAN-API from PEAK-System.

Introduction - DLL API Reference - Additional Information

Last Update: 31.07.2017

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

🕇 Home

< ^ >

Introduction

Welcome to the documentation to PCAN-Basic.

The PCAN system of the company PEAK-System Technik GmbH consists of a collection of Windows Device Drivers, which allow the real-time connection of Windows applications to all CAN busses physically connected to a PC.

PCAN-Basic, successor of PCAN-Light, is a simple programming interface to the PCAN system. Via the PCAN-Basic DII it is possible to connect own applications to the Device drivers and the PCAN hardware, to communicate with the CAN busses.

In this Chapter

Topics	Description
<u>Understanding</u> <u>PCAN-Basic</u>	This section contains an introduction to PCAN-Basic API.
Using Events	Offers support about how to read CAN messages using notifications.
License Regulations	License regulations to this software.
Contact information	Contact information - PEAK-System Technik GmbH.

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

< ^ >

Understanding PCAN-Basic

PCAN-Basic is the new version of the <u>PCAN-Light API</u>. It consists of a collection of Windows Device Drivers which allow the real-time connection of Windows applications to all CAN busses physically connected to a PC.

Main differences between PCAN-Light and PCAN-Basic

- Information about the receive time of a CAN message.
- Easy switching between different PCAN-Channels (PCAN-PC hardware).
- The possibility to control some parameters in the hardware, eg. "Listen-Only" mode, automatic reset of the CAN controller on bus-off, etc.
- The use of event notifications, for faster processing of incomming CAN messages.
- An improved system for debugging's operations.
- The use of only one Dynamic Link Library (PCAN-Basic.DLL) for all supported hardware.
- The possibility to connect more than 2 channels per PCAN-Device. The following list shows the PCAN-Channels that can be connected per PCAN-Device:

	PCAN- ISA	PCAN- Dongle	PCAN- PCI	PCAN- USB	PCAN- PC- Card	PCAN- LAN
Number of Channels	8	1	16	16	2	16

Using the PCAN-Basic

The PCAN-basic offers the possibility to use several PCAN-Channels

within the same application in an easy way. The communication process is divided in 3 phases: initialization, interaction and finalization of a PCAN-Channel.

Initialization: In order to do CAN communication using a channel, it is necessary to first initialize it. This is done making a call to the function <u>CAN_Initialize</u> (class-method: <u>Initialize</u>), or <u>CAN_InitializeFD</u> (class-method: <u>InitializeFD</u>) in case FD communication is desired.

Interaction: After a successful initialization, a channel is ready to communicate with the connected CAN bus. Further configuration is not needed. The functions CAN_Read and CAN_Write (class-methods: Read and Write) can be then used to read and write CAN messages. If the channel being used is FD capable and it was initialized using CAN_InitializedFD, then the functions to use are CAN_ReadFD and CAN_WriteFD (class-methods: ReadFD and WriteFD). If desired, extra configuration can be made to improve a communication session, like changing the message filter to target specific messages.

Finalization: When the communication is finished, the function <u>CAN_Uninitialize</u> (class-method: <u>Uninitialize</u>) 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 FD	Yes	Pcan_usb.sys
PCAN-USB Pro	Yes	Pcan_usb.sys
PCAN-USB Pro FD	Yes	Pcan_usb.sys
PCAN-PC Card	Yes	Pcan_pcc.sys
PCAN-Ethernet Gateway DR	Yes	Pcan_lan.sys
PCAN-Wireless Gateway DR	Yes	Pcan_lan.sys
PCAN-Wireless Gateway	Yes	Pcan_lan.sys
PCAN-Wireless Automotive Gateway	Yes	Pcan_lan.sys

■ See Also

PCAN Fundamentals

PCAN-Light

PCAN-API

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

PCAN-Basic Documentation

Using Events

Event objects can be used to automatically notify a client on reception of a CAN 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 <u>CAN_SetValue</u> (**class-method**: <u>SetValue</u>) function to set the parameter PCAN_RECEIVE_EVENT. This parameter sets the handle for the event object. When receiving a message, the driver 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. ¬ <u>WaitForSingleObject</u>) without increasing the processor load. After the event is signaled, the receive buffer of the client can be read with the <u>CAN_Read</u> (**class method**: <u>Read</u>) function, and the CAN 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 nonsignaled. 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 a <u>ResetEvent</u> 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

CAN_SetValue (class-method: SetValue)

CAN_Read (class-method: Read)

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🕇 Home

License Regulations

Namings for products in this manual, that are registered trademarks, are not separately marked. Therefore the missing of the ® sign does not implicate, that the naming is a free trade name. Furthermore the used names do not indicate patent rights or anything similar. PEAK-System Technik GmbH makes no representation or warranties with respect to the use of enclosed software or the contents of this manual, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, PEAK-System Technik GmbH reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Copyright © 2000-2015 PEAK-System Technik GmbH

All rights reserved.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of PEAK-System Technik GmbH.

■ See Also

Contact Information

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

+ Home

< ^ >

Contact Information

This software is a product of:



PEAK-System Technik GmbH Otto-Röhm-Str. 69 64293 Darmstadt, Germany

Info:	<u>¬ info@peak-system.com</u>
Support:	<u>support@peak-</u> <u>system.com</u>
Web:	www.peak-system.com

Copyright $\textcircled{\mbox{\sc op}}$ 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\mbox{\rm feedback}}$ to this documentation

🕇 Home

 $\langle \rangle$

Reference

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

In this Chapter

Торіс	Description
<u>Namespaces</u>	Lists the defined namespaces for Microsoft's .NET Framework programming environment.
Modules	Lists the defined modules for Python 2.6 programming environment.
<u>Units</u>	Lists the defined units for Delphi's programming environment.
<u>Classes</u>	Lists the defined classes that implement the PCAN-Basic API.
Structures	Lists the defined structures.
<u>Types</u>	Lists the defined types.
<u>Methods</u>	Lists the defined class methods for using the PCAN-Basic API.
Functions	List the defined functions for using the PCAN- Basic API.
Definitions	Lists the defined values.

Copyright $\ensuremath{\mathbb C}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🕇 Home



Namespaces

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

Namespaces

	Name	Description
8	Peak	Contains all namespaces that are part of the managed programming environment from PEAK-System.
8	Peak.Can	Contains types and classes for using the PCAN API from PEAK-System.
8	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.
8	Peak.Lin	Contains types and classes used to handle with LIN devices from PEAK-System.
8	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).

Copyright $\ensuremath{\mathbb C}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🕇 Home

< ^ >

Peak.Can.Basic

The Peak.Can.Basic namespace contain types and classes for using the PCAN-Basic API within the .NET Framework programming environment, in order to handle with PCAN devices from PEAK-System.

🗉 Remarks

Under the Delphi environment, these elements are enclosed in the <u>PCANBasic-Unit</u>. 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
\$	<u>TPCANHandle</u>	Represents a PCAN-hardware channel handle.
<i>•</i>	TPCANBitrateFD	Represents a bit rate with flexible data rate.
۵	TPCANTimestampFD	Represents the timestamp of a CAN message with flexible data rate.

Classes

	Class	Description
*:	PCANBasic	Defines a class which represents the PCAN-Basic API.

Structures

	Structure	Description
>>	<u>TPCANMsg</u>	Defines a CAN message.
>>	TPCANTimestamp	Defines a time-stamp of a CAN message.
	<u>TPCANMsgFD</u>	Defines a CAN message with flexible data rate.

Enumerations

	Enumeration	Description
3	TPCANStatus	Represents a PCAN status/error code.
7	TPCANDevice	Represents a PCAN device.
2	TPCANParameter	Represents a PCAN parameter to be read or set.
2	TPCANMessageType	Represents the type of a CAN message
.	<u>TPCANType</u>	Represents the type of a Not-Plug- And-Play PCAN hardware.
.	TPCANMode	Represents a PCAN filter mode.
2	TPCANBaudrate	Represents a PCAN bit rate register value.

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Home
 Hom

< ^ >

Modules

PEAK offers the implementation of some specific programming interfaces as modules for programming under Python (ver. 2.6). The following modules are available to be used:

Modules

	Name	Description
8	PCAN- Basic- Module	Python module for using the PCAN-Basic API from PEAK-System.

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🕇 Home

< ^ >

PCANBasic Module

The <u>Peak.Can.Basic</u> module contain types and classes for using the PCAN-Basic API within Python 2.6 programming environment, in order to handle with PCAN devices from PEAK-System.

Classes

	Class	Description
*\$	PCANBasic	Defines a class which represents the PCAN-Basic API.

■ Structures

	Structure	Description
*	<u>TPCANMsg</u>	Defines a CAN message. The members of this structure are sequentially byte aligned.
	TPCANTimestamp	Defines a time-stamp of a CAN message.
>	TPCANMsgFD	Defines a CAN message with flexible data rate.

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

🕇 Home



Units

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

□ Units

	Name	Description
8	<u>PCANBasic-</u> <u>Unit</u>	Delphi unit for using the PCAN-Basic API from PEAK-System.

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Home

PCANBasic Unit

The PCANBasic-Unit contain types and classes for using the PCAN-Basic API within the Delphi's programming environment, in order to handle with PCAN devices from PEAK-System.

🗉 Remarks

For the .NET Framework, these elements are enclosed in the <u>Peak.Can.Basic</u> 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
٢	TPCANHandle	Represents a PCAN-hardware channel handle.
۵	TPCANBitrateFD	Represents a bit rate with flexible data rate.
۵	TPCANTimestampFD	Represents the timestamp of a CAN message with flexible data rate.

Classes

	Class	Description
*:	TPCANBasic	Defines a class which represents the PCAN-Basic API.

Structures



	Structure	Description
>	<u>TPCANMsg</u>	Defines a CAN message.
>>	<u>TPCANTimestamp</u>	Defines a time-stamp of a CAN message.
	<u>TPCANMsgFD</u>	Defines a CAN message with flexible data rate.

Enumerations

	Enumeration	Description
.	TPCANStatus	Represents a PCAN status/error code.
7	TPCANDevice	Represents a PCAN device.
.	TPCANParameter	Represents a PCAN parameter to be read or set.
.	TPCANMessageType	Represents the type of a CAN message
.	<u>TPCANType</u>	Represents the type of a Not-Plug- And-Play PCAN hardware.
3	TPCANMode	Represents a PCAN filter mode.
7	TPCANBaudrate	Represents a PCAN bit rate register value.

Copyright $\ensuremath{\textcircled{O}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

< ^ >

Classes

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

Classes

	Class	Description
** 100	PCANBasic	Defines a class to use the PCAN- Basic API within the Microsoft's .NET Framework programming environment, and Python.
*** 100	TPCANBasic	Defines a class to use the PCAN- Basic API within the Delphi programming environment.

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

< ^ >

PCANBasic

Defines a class which represents the PCAN-Basic API for using within the Microsoft's .NET Framework and Python (ver. 2.6).

Syntax

∃ C#
public static class PCANBasic
∃ C++ / CLR
public ref class PCANBasic abstract sealed
■ Visual Basic
Public NotInheritable Class PCANBasic
class PCANBasic

Remarks

The PCANBasic class collects and implements the PCAN-Basic API functions. Each method is called just like the API function with the exception that the prefix "CAN_" is not used. The structure and functionality of the methods and API functions is the same.

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

```
TPCANStatus res;
// Static use, without any instance
//
res = PCANBasic.Initialize(PCAN_USBBUS1,PCAN_BAUD_
```

Within Python, a variable must be instantiated with an object of type

PCANBasic, in order to use the API.

```
# Object instantiation
#
objPCAN = PCANBasic()
res = objPCAN.Initialize(PCAN_USBBUS1, PCAN_BAUD_5
```

Note that this class under Delphi is called <u>TPCANBasic</u>.

∃ See Also

Methods Definitions

Delphi: TPCANBasic

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send feedback to this documentation

🕇 Home

< ^ >

TPCANBasic

Defines a class which represents the PCAN-Basic API in the Delphi programming environment.

Syntax

■ Pascal OO TPCANBasic = class

Remarks

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

Note that this class under .NET Framework is called <u>PCANBasic</u>.

■ See Also

<u>Methods</u> <u>Definitions</u>

.NET Framework: PCANBasic

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🕇 Home

< ^ >

Structures

The PCAN-Basic API defines the following structures:

Name	Description
TPCANMsg	Defines a CAN message.
<u>TPCANTimestamp</u>	Defines the point of time at which a CAN message was received.
<u>TPCANMsgFD</u>	Defines a CAN message with flexible data rate.

■ See Also

Reference

Copyright $\ensuremath{\textcircled{O}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🕇 Home

< ^ >

TPCANMsg

Defines a CAN message.

Syntax

```
□ C++
typedef struct
{
    DWORD ID;
    TPCANMessageType MSGTYPE;
    BYTE LEN;
    BYTE DATA[8];
 TPCANMsg;
```

```
Pascal OO
```

```
TPCANMsg = record
    ID: Longword;
    MSGTYPE: <u>TPCANMessageType</u>;
    LEN: Byte;
    DATA: array[0..7] of Byte;
end;
```

□ C#

```
public struct TPCANMsg
{
    public uint ID;
    [MarshalAs(UnmanagedType.U1)]
    public TPCANMessageType MSGTYPE;
    public byte LEN;
    [MarshalAs(UnmanagedType.ByValArray, SizeCo
    public byte[] DATA;
```

}



Remarks

The members of this structure are sequentially byte aligned.

Fields

Name	Description
ID	11/29-bit message identifier.
MSGTYPE	Type of the message. Bit mask indicating the type of the message. Several message types can be combined.
LEN	Data Length Code of the message (08).
DATA	Data of the message (DATA[0]DATA[7]).

■ See Also

CAN_Read (class-method: Read)

CAN_Write (class-method: Write)

TPCANTimestamp

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Home

< ^ >

TPCANTimestamp

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

Syntax



end;

□ C#

```
public struct TPCANTimestamp
{
    public uint millis;
    public ushort millis_overflow;
    public ushort micros;
}
```

```
□ C++ / CLR
```

public value struct TPCANTimestamp
{

UInt32 millis; UInt16 millis_overflow; UInt16 micros;

};

Bisual Basic

```
Public Structure TPCANTimestamp

Public millis As UInt32

Public millis_overflow As UInt16

Public micros As UInt16

End Structure
```

B Python

Remarks

The members of this structure are sequentially byte aligned.

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

Fields

Name	Description
millis	Base-value: milliseconds: 0 2^32-1.
millis_overflow	Roll-arounds of millis.
micros	Microseconds: 0999.



CAN_Write (class-method: Write) TPCANMsg

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\text{feedback}}$ to this documentation

🕇 Home

< ^ >

TPCANMsgFD

Defines a CAN message with flexible data rate.

Syntax



```
Pascal OO
```

```
TPCANMsqFD = record
    ID: Longword;
    MSGTYPE: <u>TPCANMessageType</u>;
    DLC: Byte;
    DATA: array[0..63] of Byte;
end;
```

□ C#

```
public struct TPCANMsgFD
{
    public uint ID;
    [MarshalAs(UnmanagedType.U1)]
    public TPCANMessageType MSGTYPE;
    public byte DLC;
    [MarshalAs(UnmanagedType.ByValArray, SizeCo
    public byte[] DATA;
```

}

```
□ C++ / CLR
public value struct TPCANMsgFD
{
    UInt32 ID;
    [MarshalAs(UnmanagedType::U1)]
    TPCANMessageType MSGTYPE;
    Byte DLC;
    [MarshalAs(UnmanagedType::ByValArray, Size(
    array<Byte>^ DATA;
};
□ Visual Basic
Public Structure TPCANMsgFD
    Public ID As UInt32
    <MarshalAs(UnmanagedType.U1)>
    Public MSGTYPE As <u>TPCANMessageType</u>
    Public DLC As Byte
    <MarshalAs(UnmanagedType.ByValArray, SizeCo
    Public DATA As Byte()
End Structure
Python
from ctypes import *
class TPCANMsgFD (Structure):
    _fields_ = [ ("ID", c_uint),
                  ("MSGTYPE", <u>TPCANMessageType</u>),
                  ("DLC", c_ubyte),
("DATA", c_ubyte * 64)]
```

∃ Remarks

The members of this structure are sequentially byte aligned.

Fields

Name	Description
ID	11/29-bit message identifier.
MSGTYPE	Type of the message. Bit mask indicating the type of the message. Several message types can be combined.
DLC	Data Length Code of the message (015).
DATA	Data of the message (DATA[0]DATA[63]).

Remark

Longer Data field with CAN FD messages:

The length of data bytes contained in a CAN message is given by the DATA LENGTH CODE field (DLC). The coding of the DLC within FD messages is different. There are 7 additional codes (from 9 to 15) that allows a FD Messages to transport up to 64 bytes of data. The relationship between DLC and data bytes length is as follow:

DLC	Data Bytes
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

9	12
10	16
11	20
12	24
13	32
14	48
15	64

■ See Also

<u>CAN_ReadFD</u> (class-method: <u>ReadFD</u>) <u>CAN_WriteFD</u> (class-method: <u>WriteFD</u>)

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

< ^ >

Types

The PCAN-Basic API defines the following types:

Name	Description
<u>TPCANHandle</u>	Represents a PCAN hardware channel handle.
TPCANStatus	Represents a PCAN status/error code.
TPCANDevice	Represents a PCAN device.
<u>TPCANParameter</u>	Represents a PCAN parameter to be read or set.
<u>TPCANMessageType</u>	Represents the type of a PCAN message.
<u>TPCANType</u>	Represents the type of PCAN hardware to be initialized.
TPCANMode	Represents a PCAN filter mode.
TPCANBaudrate	Represents a PCAN bit rate register value.
<u>TPCANBitrateFD</u>	Represents a bit rate string with flexible data rate.
TPCANTimestampFD	Represents the timestamp of a CAN message with flexible data rate.

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

< ^ >

TPCANHandle

Represents a PCAN-hardware channel handle.

Syntax

□ C++

#define TPCANHandle WORD

Pascal OO

TPCANHandle = Word;

□ C#

using TPCANHandle = System.UInt16;

□ C++ / CLR

#define TPCANHandle System::UInt16

Visual Basic

Imports TPCANHandle = System.UInt16

Bereich Python

TPCANHandle = c_ushort

🗉 Remarks

FD capable Hardware:

Some hardware can transmit using a flexible data rate (FD capable). Although there are no special PCAN-Handles to identify these hardware, it is possible to ask if a hardware is able to communicate using the FD protocol. The PCAN-Basic parameter <u>PCAN_CHANNEL_FEATURES</u> allows to investigate whether a hardware is FD capable before being initialized.

.NET Framework programming languages:

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

Aliases are defined in the <u>Peak.Can.Basic</u> 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 <u>Peak.Can.Basic</u> Namespace. Otherwise, just use the native type, which in this case is a UInt16.

C#:

```
using System;
using <u>Peak.Can.Basic;</u>
using TPCANHandle = System.UInt16; // Alias's decl
```

Visual Basic:

```
Imports System
Imports <u>Peak.Can.Basic</u>
Imports TPCANHandle = System.UInt16 ' Alias declar
```

See Also

PCAN Handle Definitions

Copyright $\ensuremath{\textcircled{O}}$ 2017. PEAK-System Technik GmbH. All rights reserved.
PCAN-Basic Documentation

🕇 Home

< ^ >

TPCANStatus

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

Syntax

⊟ C++	
#define	TPCANStatus DWORD
#define	PCAN_ERROR_OK 0x00000
#define	PCAN_ERROR_XMTFULL 0x00001
#define	PCAN_ERROR_OVERRUN 0x00002
#define	PCAN_ERROR_BUSLIGHT 0x00004
#define	PCAN_ERROR_BUSHEAVY 0x00008
#define	PCAN_ERROR_BUSWARNING PCAN_ERROR_BUSH
#define	PCAN_ERROR_BUSPASSIVE 0x40000
#define	PCAN_ERROR_BUSOFF 0x00010
#define	PCAN_ERROR_ANYBUSERR (PCAN_ERROR_BUSWAF
#define	PCAN_ERROR_QRCVEMPTY 0x00020
#define	PCAN_ERROR_QOVERRUN 0x00040
#define	PCAN_ERROR_QXMTFULL 0x00080
#define	PCAN_ERROR_REGTEST 0x00100
#define	PCAN_ERROR_NODRIVER 0x00200
#define	PCAN_ERROR_HWINUSE 0x00400
#define	PCAN_ERROR_NETINUSE 0x00800
#define	PCAN_ERROR_ILLHW 0x01400
#define	PCAN_ERROR_ILLNET 0x01800
#define	PCAN_ERROR_ILLCLIENT 0x01C00
#define	<pre>PCAN_ERROR_ILLHANDLE (PCAN_ERROR_ILLHW</pre>
#define	PCAN_ERROR_RESOURCE 0x02000
#define	PCAN_ERROR_ILLPARAMTYPE 0x04000

```
#define PCAN ERROR ILLPARAMVAL 0x08000
#define PCAN ERROR UNKNOWN 0x10000
#define PCAN ERROR ILLDATA 0x20000
#define PCAN ERROR CAUTION 0x2000000
#define PCAN ERROR INITIALIZE 0x4000000
#define PCAN ERROR ILLOPERATION 0x8000000
Pascal OO
{$Z4}
TPCANStatus = (
    PCAN\_ERROR\_OK = \$00000,
    PCAN ERROR XMTFULL = $00001,
    PCAN ERROR OVERRUN = $00002,
    PCAN\_ERROR\_BUSLIGHT = \$00004,
    PCAN ERROR BUSHEAVY = 00008,
    PCAN_ERROR_BUSHWARNING = Byte(PCAN_ERROR_E
    PCAN_ERROR_BUSPASSIVE = $40000,
    PCAN_ERROR_BUSOFF = $00010,
    PCAN_ERROR_ANYBUSERR = Byte(PCAN_ERROR_BUS)
    PCAN_ERROR_QRCVEMPTY = $00020,
    PCAN\_ERROR\_QOVERRUN = \$00040,
    PCAN\_ERROR\_QXMTFULL = $00080,
    PCAN_ERROR_REGTEST = $00100,
    PCAN_ERROR_NODRIVER = $00200,
    PCAN_ERROR_HWINUSE = $00400,
    PCAN_ERROR_NETINUSE = $00800,
    PCAN ERROR ILLHW = $01400,
    PCAN_ERROR_ILLNET = $01800,
    PCAN_ERROR_ILLCLIENT = $01C00,
    PCAN_ERROR_ILLHANDLE = Byte(PCAN_ERROR_ILLH
    PCAN\_ERROR\_RESOURCE = \$02000,
    PCAN ERROR ILLPARAMTYPE = $04000,
    PCAN ERROR ILLPARAMVAL = $08000,
    PCAN_ERROR_UNKNOWN = $10000,
```

```
PCAN\_ERROR\_ILLDATA = $20000,
    PCAN_ERROR_CAUTION = $2000000,
    PCAN_ERROR_INITIALIZE = $4000000,
    PCAN_ERROR_ILLOPERATION = $8000000
);
□ C#
[Flags]
public enum TPCANStatus : uint
{
    PCAN\_ERROR\_OK = 0 \times 00000,
    PCAN ERROR XMTFULL = 0 \times 00001,
    PCAN\_ERROR\_OVERRUN = 0 \times 00002,
    PCAN\_ERROR\_BUSLIGHT = 0 \times 00004,
    PCAN ERROR BUSHEAVY = 0 \times 00008,
    PCAN_ERROR_BUSWARNING = PCAN_ERROR_BUSHE/
    PCAN\_ERROR\_BUSPASSIVE = 0x40000,
    PCAN\_ERROR\_BUSOFF = 0 \times 00010,
    PCAN_ERROR_ANYBUSERR = (PCAN_ERROR_BUSWARN]
    PCAN\_ERROR\_QRCVEMPTY = 0x00020,
    PCAN\_ERROR\_QOVERRUN = 0 \times 00040,
    PCAN\_ERROR\_QXMTFULL = 0 \times 00080,
    PCAN\_ERROR\_REGTEST = 0 \times 00100,
    PCAN\_ERROR\_NODRIVER = 0 \times 00200,
    PCAN\_ERROR\_HWINUSE = 0 \times 00400,
    PCAN\_ERROR\_NETINUSE = 0 \times 00800,
    PCAN\_ERROR\_ILLHW = 0x01400,
    PCAN\_ERROR\_ILLNET = 0 \times 01800,
    PCAN\_ERROR\_ILLCLIENT = 0 \times 01 C \times 000,
    PCAN_ERROR_ILLHANDLE = (PCAN_ERROR_ILLHW |
    PCAN\_ERROR\_RESOURCE = 0 \times 02000,
    PCAN ERROR ILLPARAMTYPE = 0 \times 04000,
    PCAN_ERROR_ILLPARAMVAL = 0x08000,
    PCAN\_ERROR\_UNKNOWN = 0 \times 10000,
```

```
PCAN\_ERROR\_ILLDATA = 0x20000,
    PCAN_ERROR_CAUTION = 0 \times 2000000,
    PCAN_ERROR_INITIALIZE = 0x4000000,
    PCAN_ERROR_ILLOPERATION = 0x8000000,
}
□ C++ / CLR
[Flags]
public enum class TPCANStatus : UInt32
{
    PCAN\_ERROR\_OK = 0 \times 00000,
    PCAN ERROR XMTFULL = 0 \times 00001,
    PCAN\_ERROR\_OVERRUN = 0 \times 00002,
     PCAN\_ERROR\_BUSLIGHT = 0 \times 00004,
     PCAN ERROR BUSHEAVY = 0 \times 00008,
    PCAN_ERROR_BUSWARNING = PCAN_ERROR_BUSHE/
    PCAN\_ERROR\_BUSPASSIVE = 0 \times 40000,
     PCAN\_ERROR\_BUSOFF = 0 \times 00010,
    PCAN_ERROR_ANYBUSERR = (PCAN_ERROR_BUSWARN]
    PCAN\_ERROR\_QRCVEMPTY = 0x00020,
    PCAN\_ERROR\_QOVERRUN = 0 \times 00040,
    PCAN\_ERROR\_QXMTFULL = 0 \times 00080,
    PCAN\_ERROR\_REGTEST = 0 \times 00100,
     PCAN\_ERROR\_NODRIVER = 0 \times 00200,
    PCAN\_ERROR\_HWINUSE = 0 \times 00400,
     PCAN\_ERROR\_NETINUSE = 0 \times 00800,
    PCAN\_ERROR\_ILLHW = 0x01400,
    PCAN\_ERROR\_ILLNET = 0 \times 01800,
    PCAN\_ERROR\_ILLCLIENT = 0 \times 01 C 00,
    PCAN_ERROR_ILLHANDLE = (PCAN_ERROR_ILLHW |
    PCAN\_ERROR\_RESOURCE = 0 \times 02000,
     PCAN ERROR ILLPARAMTYPE = 0 \times 04000,
    PCAN_ERROR_ILLPARAMVAL = 0 \times 08000,
    PCAN\_ERROR\_UNKNOWN = 0 \times 10000,
```

```
PCAN_ERROR_ILLDATA = 0x20000,
PCAN_ERROR_CAUTION = 0x2000000,
PCAN_ERROR_INITIALIZE = 0x4000000,
PCAN_ERROR_ILLOPERATION = 0x8000000,
```

```
Visual Basic
```

```
<Flags()>
Public Enum TPCANStatus As UInt32
    PCAN ERROR OK = \&HO
    PCAN ERROR XMTFULL = &H1
    PCAN ERROR OVERRUN = \&H2
    PCAN ERROR BUSLIGHT = \&H4
    PCAN ERROR BUSHEAVY = \&H8
    PCAN ERROR BUSWARNING = PCAN ERROR BUSHEAV
    PCAN ERROR BUSPASSIVE = &H40000
    PCAN\_ERROR\_BUSOFF = \&H10
    PCAN\_ERROR\_ANYBUSERR = (PCAN\_ERROR\_BUSWARN]
    PCAN\_ERROR\_QRCVEMPTY = \&H20
    PCAN_ERROR_QOVERRUN = &H40
    PCAN_ERROR_QXMTFULL = &H80
    PCAN\_ERROR\_REGTEST = \&H100
    PCAN ERROR NODRIVER = \&H200
    PCAN\_ERROR\_HWINUSE = \&H400
    PCAN ERROR NETINUSE = & H800
    PCAN ERROR ILLHW = \&H1400
    PCAN ERROR ILLNET = \&H1800
    PCAN ERROR ILLCLIENT = &H1C00
    PCAN_ERROR_ILLHANDLE = (PCAN_ERROR_ILLHW O)
    PCAN ERROR RESOURCE = &H2000
    PCAN ERROR ILLPARAMTYPE = &H4000
    PCAN ERROR ILLPARAMVAL = & H8000
    PCAN ERROR UNKNOWN = &H10000
    PCAN ERROR ILLDATA = &H20000
```

```
PCAN_ERROR_CAUTION = &H2000000
PCAN_ERROR_INITIALIZE = &H4000000
PCAN_ERROR_ILLOPERATION = &H8000000
End Enum
```

Bereich Python

TPCANStatus = int

```
PCAN_ERROR_OK = TPCANStatus(0x00000)
PCAN ERROR XMTFULL = TPCANStatus(0x00001)
PCAN_ERROR_OVERRUN = TPCANStatus(0x00002
PCAN ERROR BUSLIGHT = TPCANStatus(0x00004)
PCAN\_ERROR\_BUSHEAVY = TPCANStatus(0x00008)
PCAN_ERROR_BUSWARNING = TPCANStatus(PCAN_ERROR_
PCAN_ERROR_BUSPASSIVE = TPCANStatus(0x40000)
PCAN_ERROR_BUSOFF = TPCANStatus(0x00010)
PCAN_ERROR_ANYBUSERR = TPCANStatus(PCAN_ERROR_E
PCAN_ERROR_QRCVEMPTY = TPCANStatus(0x00020)
PCAN_ERROR_QOVERRUN = TPCANStatus(0x00040)
PCAN_ERROR_QXMTFULL = TPCANStatus(0x00080)
PCAN_ERROR_REGTEST = TPCANStatus(0x00100)
PCAN\_ERROR\_NODRIVER = TPCANStatus(0x00200)
PCAN_ERROR_HWINUSE = TPCANStatus(0x00400)
PCAN_ERROR_NETINUSE = TPCANStatus(0x00800)
PCAN\_ERROR\_ILLHW = TPCANStatus(0x01400)
PCAN_ERROR_ILLNET = TPCANStatus(0x01800)
PCAN_ERROR_ILLCLIENT = TPCANStatus(0x01C00)
PCAN_ERROR_ILLHANDLE = TPCANStatus(PCAN_ERROR_]
PCAN_ERROR_RESOURCE = TPCANStatus(0x02000)
PCAN_ERROR_ILLPARAMTYPE = TPCANStatus(0x04000)
PCAN_ERROR_ILLPARAMVAL = TPCANStatus(0x08000)
PCAN_ERROR_UNKNOWN = TPCANStatus(0x10000)
PCAN_ERROR_ILLDATA = TPCANStatus(0x20000)
PCAN_ERROR_CAUTION = TPCANStatus(0x2000000)
```

PCAN_ERROR_INITIALIZE = TPCANStatus(0x4000000) PCAN_ERROR_ILLOPERATION = TPCANStatus(0x8000000)

🗉 Remarks

Note that the values of the different PCAN-Status definitions are able to be bitwise combined. In some cases it is possible to get more than one error code as result of calling a function.

*Note that the values of PCAN_ERROR_INITIALIZE and PCAN_ERROR_ILLOPERATION were changed!

PCAN_ERROR_INITIALIZE changed from 0x40000 to 0x4000000

PCAN_ERROR_ILLOPERATION changed from 0x80000 to 0x8000000

Values

	1	Π
Name	Value	Description
PCAN_ERROR_OK	0x00000 (000000)	No error. Success.
PCAN_ERROR_XMTFULL	0x00001 (000001)	Transmit buffer in CAN controller is full.
PCAN_ERROR_OVERRUN	0x00002 (000002)	CAN controller was read too late.
PCAN_ERROR_BUSLIGHT	0x00004 (000004)	Bus error: an error counter reached the 'light' limit.
PCAN_ERROR_BUSHEAVY	0x00008	Bus error: an

	(000008)	error counter reached the 'heavy' limit.
PCAN_ERROR_BUSWARNING	0x00008 (000008)	Bus error: an error counter reached the 'warning' limit.
PCAN_ERROR_BUSPASSIVE	0x40000 (262144)	Bus error: the CAN controller is error passive.
PCAN_ERROR_BUSOFF	0x00010 (000016)	Bus error: the CAN controller is in bus-off state.
PCAN_ERROR_ANYBUSERR	0x4001C (262172)	Mask for all bus errors.
PCAN_ERROR_QRCVEMPTY	0x00020 (000032)	Receive queue is empty.
PCAN_ERROR_QOVERRUN	0x00040 (000064)	Receive queue was read too late.
PCAN_ERROR_QXMTFULL	0x00080 (000128)	Transmit queue is full.
PCAN_ERROR_REGTEST	0x00100 (000256)	Test of the CAN controller hardware registers failed (no

		hardware found).
PCAN_ERROR_NODRIVER	0x00200 (000512)	Driver not loaded.
PCAN_ERROR_HWINUSE	0x00400 (001024)	PCAN- Hardware already in use by a PCAN- Net.
PCAN_ERROR_NETINUSE	0x00800 (002048)	A PCAN- Client is already connected to the PCAN- Net.
PCAN_ERROR_ILLHW	0x01400 (005120)	PCAN- Hardware handle is invalid.
PCAN_ERROR_ILLNET	0x01800 (006144)	PCAN-Net handle is invalid.
PCAN_ERROR_ILLCLIENT	0x01C00 (007168)	PCAN-Client handle is invalid.
PCAN_ERROR_ILLHANDLE	0x01C00 (007168)	Mask for all handle errors.
PCAN_ERROR_RESOURCE	0x02000 (008192)	Resource (FIFO, Client, timeout) cannot be

		created.
PCAN_ERROR_ILLPARAMTYPE	0x04000 (016384)	Invalid parameter.
PCAN_ERROR_ILLPARAMVAL	0x08000 (032768)	Invalid parameter value.
PCAN_ERROR_UNKNOWN	0x10000 (065536)	Unknown error
PCAN_ERROR_ILLDATA	0x20000 (131072)	Invalid data, function, or action.
PCAN_ERROR_CAUTION	0x2000000 (33554432)	Operation succeeded but with irregularities.
PCAN_ERROR_INITIALIZE*	0x4000000 (67108864)	Channel is not initialized.
PCAN_ERROR_ILLOPERATION*	0x8000000 (134217728)	Invalid operation.

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

PCAN-Basic Documentation

🔒 Home

<^>

TPCANDevice

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

Syntax

□ C++	
#define T	PCANDevice BYTE
<pre>#define P #define P</pre>	CAN_NONE 0x00 CAN_PEAKCAN 0x01 CAN_ISA 0x02 CAN_DNG 0x03 CAN_PCI 0x04 CAN_USB 0x05 CAN_PCC 0x06
#define P #define P	CAN_VIRTUAL 0x07 CAN_LAN 0x08
Bernal OO	
<pre>{\$Z1} TPCANDevi PCAN_ PCAN_</pre>	ce = (NONE = 0, PEAKCAN = 1, ISA = 2, DNG = 3, PCI = 4, USB = 5, PCC = 6, VIRTUAL = 7, LAN = 8

```
);
□ C#
public enum TPCANDevice : byte
{
    PCAN_NONE = 0,
    PCAN_PEAKCAN = 1,
    PCAN_{ISA} = 2,
    PCAN_DNG = 3,
    PCAN_PCI = 4,
    PCAN_USB = 5,
    PCAN PCC = 6,
    PCAN_VIRTUAL = 7,
    PCAN_LAN = 8
}
□ C++ / CLR
public enum class TPCANDevice : Byte
{
    PCAN_NONE = 0,
    PCAN_PEAKCAN = 1,
    PCAN_{ISA} = 2,
    PCAN_DNG = 3,
    PCAN_PCI = 4,
    PCAN_USB = 5,
    PCAN_PCC = 6,
    PCAN_VIRTUAL = 7,
    PCAN LAN = 8
};
Visual Basic
Public Enum TPCANDevice As Byte
    PCAN NONE = 0
```

```
PCAN_NONE = 0

PCAN_PEAKCAN = 1
```

```
PCAN_ISA = 2

PCAN_DNG = 3

PCAN_PCI = 4

PCAN_USB = 5

PCAN_PCC = 6

PCAN_VIRTUAL = 7

PCAN_LAN = 8

End Enum
```

Python

```
TPCANDevice = c_ubyte
```

```
PCAN_NONE = TPCANDevice(0x00)
PCAN_PEAKCAN = TPCANDevice(0x01)
PCAN_ISA = TPCANDevice(0x02)
PCAN_DNG = TPCANDevice(0x03)
PCAN_PCI = TPCANDevice(0x04)
PCAN_USB = TPCANDevice(0x05)
PCAN_PCC = TPCANDevice(0x06)
PCAN_VIRTUAL = TPCANDevice(0x07)
PCAN_LAN = TPCANDevice(0x08)
```

Remarks

The PCAN-Devices PCAN_PEAKCAN and PCAN_VIRTUAL are not used within the PCAN-Basic API.

Values

Name	Value	Description
PCAN_NONE	0	Undefined, unknown or not selected PCAN device value.
PCAN_PEAKCAN	1	PCAN Non-Plug And Play devices. NOT USED WITHIN PCAN-Basic

		API.
PCAN_ISA	2	PCAN-ISA, PCAN-PC/104.
PCAN_DNG	3	PCAN-Dongle.
PCAN_PCI	4	PCAN-PCI, PCAN-cPCI, PCAN- miniPCI, PCAN-PC/104-Plus, and PCAN-PCI Express.
PCAN_USB	5	PCAN-USB and PCAN-USB Pro.
PCAN_PCC	6	PCAN-PC Card.
PCAN_VIRTUAL	7	PCAN Virtual hardware. NOT USED WITHIN PCAN-Basic API.
PCAN_LAN	8	PCAN Gateway devices.

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

TPCANParameter

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

Syntax

⊟ C++	
#define	TPCANParameter BYTE
#define	PCAN_DEVICE_NUMBER 0x01
#define	PCAN_5VOLTS_POWER 0x02
#define	PCAN_RECEIVE_EVENT 0x03
#define	PCAN_MESSAGE_FILTER 0x04
#define	PCAN_API_VERSION 0x05
#define	PCAN_CHANNEL_VERSION 0x06
#define	PCAN_BUSOFF_AUTORESET 0x07
#define	PCAN_LISTEN_ONLY 0x08
#define	PCAN_LOG_LOCATION 0x09
#define	PCAN_LOG_STATUS 0x0A
#define	PCAN_LOG_CONFIGURE 0x0B
#define	PCAN_LOG_TEXT 0x0C
#define	PCAN_CHANNEL_CONDITION 0×0D
#define	PCAN_HARDWARE_NAME 0x0E
#define	PCAN_RECEIVE_STATUS 0x0F
#define	PCAN_CONTROLLER_NUMBER 0x10
#define	PCAN_TRACE_LOCATION 0x11
#define	PCAN_TRACE_STATUS 0x12
#define	PCAN_TRACE_SIZE 0x13
#define	PCAN_TRACE_CONFIGURE 0x14

```
#define PCAN_CHANNEL_IDENTIFYING 0x15
#define PCAN_CHANNEL_FEATURES 0x16
#define PCAN_BITRATE_ADAPTING 0x17
#define PCAN_BITRATE_INFO = 0x18
#define PCAN_BITRATE_INFO_FD = 0x19
#define PCAN_BUSSPEED_NOMINAL = 0x1A
#define PCAN_BUSSPEED_DATA = 0x1B
#define PCAN_IP_ADDRESS = 0x1C
#define PCAN_LAN_SERVICE_STATUS = 0x1D
#define PCAN_ALLOW_STATUS_FRAMES = 0x1E
#define PCAN_ALLOW_RTR_FRAMES = 0x20
#define PCAN_INTERFRAME_DELAY = 0x22
#define PCAN_ACCEPTANCE_FILTER_11BIT = 0x22
#define PCAN_ACCEPTANCE FILTER_29BIT = 0x23
```

Bernard Base Pascal OO

```
{$Z1}
TPCANParameter = (
    PCAN DEVICE NUMBER = 1,
    PCAN_5VOLTS_POWER = 2,
    PCAN_RECEIVE_EVENT = 3,
    PCAN\_MESSAGE\_FILTER = 4,
    PCAN\_API\_VERSION = 5,
    PCAN CHANNEL VERSION = 6,
    PCAN_BUSOFF_AUTORESET = 7,
    PCAN LISTEN ONLY = 8,
    PCAN_LOG_LOCATION = 9,
    PCAN\_LOG\_STATUS = 10,
    PCAN LOG CONFIGURE = 11,
    PCAN LOG TEXT = 12,
    PCAN CHANNEL CONDITION = 13,
    PCAN HARDWARE NAME = 14,
    PCAN RECEIVE STATUS = 15,
```

```
PCAN_CONTROLLER_NUMBER = 16,
PCAN_TRACE_LOCATION = 17,
PCAN_TRACE_STATUS = 18,
PCAN_TRACE_SIZE = 19,
PCAN_TRACE_CONFIGURE = 20,
PCAN_CHANNEL_IDENTIFYING = 21,
PCAN_CHANNEL_FEATURES = 22,
PCAN_BITRATE_ADAPTING = 23,
PCAN_BITRATE_INFO = 24,
PCAN_BITRATE_INFO_FD = 25,
PCAN_BUSSPEED_NOMINAL = 26,
PCAN_BUSSPEED_DATA = 27,
PCAN_IP_ADDRESS = 28,
PCAN\_LAN\_SERVICE\_STATUS = 29,
PCAN_ALLOW_STATUS_FRAMES = 30,
PCAN_ALLOW_RTR_FRAMES = 31,
PCAN_ALLOW_ERROR_FRAMES = 32,
PCAN_INTERFRAME_DELAY = 33,
PCAN_ACCEPTANCE_FILTER_11BIT = 34,
PCAN ACCEPTANCE FILTER 29BIT = 35
```

);

```
□ C#
```

```
public enum TPCANParameter : byte
{
    PCAN_DEVICE_NUMBER = 1,
    PCAN_5VOLTS_POWER = 2,
    PCAN_RECEIVE_EVENT = 3,
    PCAN_MESSAGE_FILTER = 4,
    PCAN_API_VERSION = 5,
    PCAN_CHANNEL_VERSION = 5,
    PCAN_CHANNEL_VERSION = 6,
    PCAN_BUSOFF_AUTORESET = 7,
    PCAN_LISTEN_ONLY = 8,
    PCAN_LOG_LOCATION = 9,
```

```
PCAN\_LOG\_STATUS = 10,
PCAN_LOG_CONFIGURE = 11,
PCAN_LOG_TEXT = 12,
PCAN_CHANNEL_CONDITION = 13,
PCAN_HARDWARE_NAME = 14,
PCAN_RECEIVE\_STATUS = 15,
PCAN_CONTROLLER_NUMBER = 16,
PCAN_TRACE_LOCATION = 17,
PCAN_TRACE_STATUS = 18,
PCAN_TRACE_SIZE = 19,
PCAN_TRACE_CONFIGURE = 20,
PCAN_CHANNEL_IDENTIFYING = 21,
PCAN_CHANNEL_FEATURES = 22,
PCAN_BITRATE_ADAPTING = 23,
PCAN_BITRATE_INFO = 24,
PCAN_BITRATE_INFO_FD = 25,
PCAN_BUSSPEED_NOMINAL = 26,
PCAN_BUSSPEED_DATA = 27,
PCAN_IP_ADDRESS = 28,
PCAN_LAN_SERVICE_STATUS = 29,
PCAN_ALLOW_STATUS_FRAMES = 30,
PCAN_ALLOW_RTR_FRAMES = 31,
PCAN_ALLOW_ERROR_FRAMES = 32,
PCAN_INTERFRAME_DELAY = 33,
PCAN_ACCEPTANCE_FILTER_11BIT = 34,
PCAN_ACCEPTANCE_FILTER_29BIT = 35,
```

□ C++ / CLR

}

```
public enum class TPCANParameter : Byte
{
    PCAN_DEVICE_NUMBER = 1,
    PCAN_5VOLTS_POWER = 2,
    PCAN_RECEIVE_EVENT = 3,
```



🖃 Visual Basic

```
Public Enum TPCANParameter As Byte
    PCAN DEVICE NUMBER = 1
   PCAN 5VOLTS POWER = 2
   PCAN RECEIVE EVENT = 3
   PCAN MESSAGE FILTER = 4
   PCAN_API_VERSION = 5
   PCAN CHANNEL VERSION = 6
    PCAN BUSOFF AUTORESET = 7
    PCAN LISTEN ONLY = 8
    PCAN LOG LOCATION = 9
   PCAN_LOG_STATUS = 10
    PCAN LOG CONFIGURE = 11
    PCAN LOG TEXT = 12
   PCAN_CHANNEL_CONDITION = 13
   PCAN HARDWARE NAME = 14
   PCAN_RECEIVE_STATUS = 15
    PCAN CONTROLLER NUMBER = 16
    PCAN TRACE LOCATION = 17
    PCAN TRACE STATUS = 18
    PCAN TRACE SIZE = 19
    PCAN TRACE CONFIGURE = 20
   PCAN_CHANNEL_IDENTIFYING = 21
   PCAN CHANNEL FEATURES = 22
   PCAN BITRATE ADAPTING = 23
   PCAN BITRATE INFO = 24
    PCAN BITRATE INFO FD = 25
    PCAN BUSSPEED NOMINAL = 26
   PCAN BUSSPEED DATA = 27
    PCAN IP ADDRESS = 28
    PCAN LAN SERVICE STATUS = 29
    PCAN\_ALLOW\_STATUS\_FRAMES = 30
    PCAN ALLOW RTR FRAMES = 31
   PCAN_ALLOW_ERROR_FRAMES = 32
   PCAN INTERFRAME DELAY = 33
```

```
PCAN_ACCEPTANCE_FILTER_11BIT = 34
PCAN_ACCEPTANCE_FILTER_29BIT = 35
End Enum
```

Bereich Python

TPCANParameter = c_ubyte

```
PCAN_DEVICE_NUMBER = TPCANParameter(0x01)
PCAN_5VOLTS_POWER = TPCANParameter(0x02)
PCAN_RECEIVE_EVENT = TPCANParameter(0x03)
PCAN\_MESSAGE\_FILTER = TPCANParameter(0x04)
PCAN API VERSION = TPCANParameter(0x05)
PCAN_CHANNEL_VERSION = TPCANParameter(0x06)
PCAN_BUSOFF_AUTORESET = TPCANParameter(0x07)
PCAN_LISTEN_ONLY = TPCANParameter(0x08)
PCAN_LOG_LOCATION = TPCANParameter(0x09)
PCAN\_LOG\_STATUS = TPCANParameter(0x0A)
PCAN\_LOG\_CONFIGURE = TPCANParameter(0x0B)
PCAN_LOG_TEXT = TPCANParameter(0x0C)
PCAN_CHANNEL_CONDITION = TPCANParameter(0x0D)
PCAN_HARDWARE_NAME = TPCANParameter(0x0E)
PCAN_RECEIVE_STATUS = TPCANParameter(0x0F)
PCAN_CONTROLLER_NUMBER = TPCANParameter(0x10)
PCAN_TRACE_LOCATION = TPCANParameter(0x11)
PCAN_TRACE_STATUS = TPCANParameter(0x12)
PCAN_TRACE_SIZE = TPCANParameter(0x13)
PCAN_TRACE_CONFIGURE = TPCANParameter(0x14)
PCAN_CHANNEL_IDENTIFYING = TPCANParameter(0x15)
PCAN_CHANNEL_FEATURES = TPCANParameter(0x16)
PCAN_BITRATE_ADAPTING = TPCANParameter(0x17)
PCAN_BITRATE_INFO = TPCANParameter(0x18)
PCAN_BITRATE_INFO_FD = TPCANParameter(0x19)
PCAN_BUSSPEED_NOMINAL = TPCANParameter(0x1A)
PCAN_BUSSPEED_DATA = TPCANParameter(0x1B)
```

```
PCAN_IP_ADDRESS = TPCANParameter(0x1C)
PCAN_LAN_SERVICE_STATUS = TPCANParameter(0x1D)
PCAN_ALLOW_STATUS_FRAMES = TPCANParameter(0x1E)
PCAN_ALLOW_RTR_FRAMES = TPCANParameter(0x1F)
PCAN_ALLOW_ERROR_FRAMES = TPCANParameter(0x20)
PCAN_INTERFRAME_DELAY = TPCANParameter(0x21)
PCAN_ACCEPTANCE_FILTER_11BIT = TPCANParameter((
PCAN_ACCEPTANCE_FILTER_29BIT = TPCANParameter(()
```

Values

Name	Value	Data Type	Descrip
PCAN_DEVICE_NUMBER	1	Integer	PCAN-l number
PCAN_5VOLTS_POWER	2	Integer	PCAN-F power"
PCAN_RECEIVE_EVENT	3	Handle	PCAN r handler
PCAN_MESSAGE_FILTER	4	Integer	PCAN n parame
PCAN_API_VERSION	5	String	PCAN-E version
PCAN_CHANNEL_VERSION	6	String	PCAN c version
PCAN_BUSOFF_AUTORESET	7	Integer	PCAN " parame
PCAN_LISTEN_ONLY	8	Integer	PCAN " parame
PCAN_LOG_LOCATION	9	String	Director

			files.
PCAN_LOG_STATUS	10	Integer	Debug-I status.
PCAN_LOG_CONFIGURE	11	Integer	Configu debugg((LOG_F
PCAN_LOG_TEXT	12	String	Custom into the
PCAN_CHANNEL_CONDITION	13	Integer	Availabi PCAN-(
PCAN_HARDWARE_NAME	14	String	PCAN " parame
PCAN_RECEIVE_STATUS	15	Integer	"Receiv parame messag
PCAN_CONTROLLER_NUMBER	16	Integer	Index of Controll device.
PCAN_TRACE_LOCATION	17	String	Director trace file
PCAN_TRACE_STATUS	18	Integer	PCAN-1 status.
PCAN_TRACE_SIZE	19	Integer	Configu maximu PCAN-1
PCAN_TRACE_CONFIGURE	20	Integer	Configu trace file (TRACE
PCAN_CHANNEL_IDENTIFYING	21	Integer	USB Ch Identifyi

			status.
PCAN_CHANNEL_FEATURES	22	Integer	Capabil device (
PCAN_BITRATE_ADAPTING	23	Integer	Attachm existing unknow rate.
PCAN_BITRATE_INFO	24	Integer	Current BTR0B ⁻ (Standa
PCAN_BITRATE_INFO_FD	25	String	Current String v
PCAN_BUSSPEED_NOMINAL	26	Integer	Current bus spe bits/sec
PCAN_BUSSPEED_DATA	27	Integer	Current speed ir
PCAN_IP_ADDRESS	28	String	Remote IPv4 for
PCAN_LAN_SERVICE_STATUS	29	Integer	Runninç LAN Se PCAN-(
PCAN_ALLOW_STATUS_FRAMES	30	Integer	"Receiv parame frames.
PCAN_ALLOW_RTR_FRAMES	31	Integer	"Receiv parame frames.
PCAN_ALLOW_ERROR_FRAMES	32	Integer	"Receiv parame

			frames.
PCAN_INTERFRAME_DELAY	33	Integer	Delay, ir betweer frames.
PCAN_ACCEPTANCE_FILTER_11BIT	34	64-Bit Integer	Accepta code an bit CAN
PCAN_ACCEPTANCE_FILTER_29BIT	35	64-Bit Integer	Accepta code an bit CAN

Characteristics

PCAN_DEVICE_NUMBER

Access: R 🕅

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

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, than the value is 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 (FFh), for 32 bits: 429496729 (FFFFFFFh).

PCAN-Device: PCAN-USB.

PCAN_5VOLTS_POWER

Access: R 🕅

Description: This parameter is used on PCAN-PC Card hardware for

switching the external 5V on the D-Sub connector of the PC Card. This is useful when connecting external bus converter modules to the card (AU5790 / TJA1054)).

Possible values: This parameter can have one of these values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

PCAN-Device: PCAN-PC Card, PCAN-HUB.

PCAN_RECEIVE_EVENT

Access: R 🕅

Description: This parameter is used to let the PCAN driver notify an application when CAN messages are placed in its receive queue. 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 \neg <u>CreateEvent</u> or the value **0** (*IntPtr.Zero* in a managed environment). When setting this parameter, the value of **0** resets the parameter in the PCAN driver. When reading the value of **0** indicate that no event handle is set. For more information about reading with events, please refer to the topic <u>Using Events</u>.

Default value: Disabled (0).

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

PCAN_MESSAGE_FILTER

Access: R 🕅

Description: This parameter allows the user to easy configure the message filter of a PCAN channel. With it is possible to fully open or complete close the filter.

Possible values: When setting only two values are possible: **PCAN_FILTER_OPEN**, **PCAN_FILTER_CLOSE**. When reading it is possible to receive a third value, **PCAN_FILTER_CUSTOM**, which indicates that the filter is configured to receive a custom range of IDs. Note that other values will considered invalid.

Default value: Complete opened (PCAN_FILTER_OPEN).

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

PCAN_API_VERSION

Access:

Description: This parameter is used to get information about the PCAN-Basic 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**,**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 PCAN_NONEBUS channel.

PCAN_CHANNEL_VERSION

Access: ℝ

Description: This parameter is used to get version information about the Driver of a PCAN Channel.

Possible values: The value is a null-terminated string which contains version number, driver name and copyright information about the driver used to handle with an specified PCAN channel. The length of

the this text will have a maximum length of 256 bytes (null-termination included) .

Default value: NA.

PCAN-Device: All PCAN devices (excluding PCAN_NONEBUS channel). *Note*: It is not needed to have a PCAN channel initialized before asking for its version.

PCAN_BUSOFF_AUTORESET

Access: R 🕅

Description: This parameter instructs the PCAN driver to reset automatically the CAN controller of a PCAN channel when a bus-off state is detected. Since no communication is possible on a bus-off state, it is useful to let the driver to catch this event automatically and reset the controller, avoiding extra handling of this problem in an end application.

Possible values: This parameter can have one of these values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

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

REMARKS: Reseting the hardware has a duration of ~ 500 milliseconds. After receiving the PCAN_ERROR_BUSOFF error, an application should wait that time before trying to read or write again.

PCAN_LISTEN_ONLY

Access: R 🕅

Description: This parameter allows the user to set a CAN hardware in Listen-Only mode. When this mode is set, the CAN controller doens't take part on active events (eg. transmit CAN messages) but stays in a passive mode (CAN monitor), in which it can analyse the traffic on the CAN bus used by a PCAN channel. See also the Philips Data Sheet "SJA1000 Stand-alone CAN controller".

Possible values: This parameter can have one of these values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

PCAN-Device: All PCAN devices (excluding PCAN_NONEBUS) containing a SJA1000 CAN controller.

REMARKS: This parameter can be used with an initialized or uninitialized channel. Configuring this parameter without having the channel initialized, does a so called "pre-initialization". This means that the channel will be set in Listen-Only mode as soon as possible, after it has been successfully connected, using <u>CAN_Initialize</u> (**classmethod**: <u>Initialize</u>). Once the channel is disconnected, further initializations of this channel are done in normal mode. It is needed to set this parameter again before each initialization process, if the behavior described before is required. This is usefull to avoid or minimize arbitration problems when connecting to a CAN-network.

PCAN_LOG_LOCATION

Access: R 🕅

Description: This value is used to set the folder location on a computer for the Log-File generated by the PCAN-Basic API, within a debug session. Setting this value starts recording debug information automatically. If a debug session is running (a log file is being written), PCAN_LOG_LOCATION instructs the API to close the current log file and to start the process again with the new folder information. Note that the name of the log file cannot be specified, this name is fixed as PCANBasic.log.

Possible values: This value must be a fully-qualified and valid path to an **existing directory** on the executing computer. There is no limit for the length of the string but it is recommended to use a length not bigger than MAX_PATH. For more information see <u>a Naming Files</u>,

Paths, and Namespaces.

Default value: Calling process's folder.

PCAN-Device: Default channel Only (PCAN_NONEBUS).

PCAN_LOG_STATUS

Access: R 🕅

Description: This value is used to control the activity status of a debug session within the PCAN-Basic API. If the log status is set to ON without having set a location for the log file or without having configured the information to be traced, then the session process will start with the default values.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

PCAN-Device: Default channel Only (PCAN_NONEBUS).

PCAN_LOG_CONFIGURE

Access: R 🕅

Description: This value is used to configure the debug information to be included in the log file generated in a debug session within the PCAN-Basic API.

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

• LOG_FUNCTION_DEFAULT: This value is always active. It defines the default information to be traced, which is an unexpected exception like a memory access violation. After having configured the log with more options, this value can be used to reset that configuration, setting the log with its default value again. This kind of entry is marked with the word "EXCEPTION"* in the log file.

- LOG_FUNCTION_ENTRY: This value causes an entry in the log file each time an API function is entered. This kind of entry is marked with the word "ENTRY"* in the log file.
- LOG_FUNCTION_PARAMETERS: This value causes an entry in the log file each time an API function is entered, showing the name of the parameters passed to the function as well as their values. This kind of entry is marked with the word "PARAMETERS"* in the log file.
- LOG_FUNCTION_LEAVE: This value causes an entry in the log file each time an API function is abandoned. This kind of entry is marked with the word "EXIT"* in the log file.
- LOG_FUNCTION_WRITE: This value causes an entry in the log file each time a CAN message is written, using the function CAN_Write (class-method: Write). This kind of entry is marked with the word "CHANNEL 0xXX (Y)"* in the log file, where XX is the channel number in hex notation, and Y the word "OUT"* denoting the direction (outgoing). The complete CAN message is also represented as hex text.
- LOG_FUNCTION_READ: This value causes an entry in the log file each time a CAN message is read, using the functions <u>CAN_Read/CAN_ReadFD</u> (class-methods: <u>Read</u>, <u>ReadFD</u>). This kind of entry is marked with the word "CHANNEL 0xXX (Y)"* in the log file, where XX is the channel number in hex notation, and Y the word "IN"* denoting the direction (incoming). The complete CAN message is also represented as hex text.

* Note that the PCAN-Basic API supports several languages. The log file use the language of the operating system. If this language is not one of the supported languages, than English is used.

* These words are always written in English, independently of the operating system's language.

Default value: Exceptions and Errors (LOG_FUNCTION_DEFAULT).

PCAN-Device: Default channel Only (PCAN_NONEBUS).

PCAN_LOG_TEXT

Access: 🕅

Description: This value is used to insert custom information in the log file generated in a debug session within the PCAN-Basic API. Setting this value starts recording debug information automatically.

This is very useful when it is desired to specially mark places of an application's execution path while debugging PCAN-Basic tasks. Furthermore, an application could use this feature as an own Log file. To do so, just use the default log's configuration (PCAN_LOG_CONFIGURE set to LOG_FUNCTION_DEFAULT) and include the desired information using PCAN_LOG_TEXT. In this way the log file will contain only user-defined debug information. Note that the name of the log file cannot be specified, this name is fixed as PCANBasic.log.

Possible values: This value must be a null-terminated string. There is no limit for the length of the string but it is recommended to use a length not bigger than MAX_PATH. For more information see **7** Naming Files, Paths, and Namespaces.

Default value: NA.

PCAN-Device: Default channel Only (PCAN_NONEBUS).

PCAN_CHANNEL_CONDITION

Access: 🗷

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: PCAN_CHANNEL_UNAVAILABLE, PCAN_CHANNEL_AVAILABLE, PCAN_CHANNEL_OCCUPIED, PCAN_CHANNEL_PCANVIEW.

Default value: N/A.

PCAN-Device: All PCAN devices (excluding PCAN_NONEBUS

channel).

PCAN_HARDWARE_NAME

Access:

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 PCAN_NONEBUS channel).

REMARKS: This parameter can be used with an initialized or uninitialized channel.

PCAN_RECEIVE_STATUS

Access: 🗷 🕅

Description: This parameter helps the user to allow / disallow the reception of messages within a PCAN Channel, regardless of the value of its reception filter. When the "Receive Status" is active (ON), incoming messages are forwarded to the user application through the CAN_Read/CAN_ReadFD functions (class-methods: Read, ReadFD). If "Receive Status" is deactivated (OFF), the incoming messages are disposed from the receive queue and each call to CAN_Read/CAN_ReadFD returns PCAN_ERROR_QRCVEMPTY. The acceptance filter of the channel remains unchanged (other applications working with the same PCAN-Hardware will not be disturbed).

Possible values: This parameter can have one of these values:

PCAN_PARAMETER_OFF, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Activated (PCAN_PARAMETER_ON).

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

REMARKS: This parameter can be used with an initialized or uninitialized channel. Configuring this parameter without having the channel initialized, does a so called "pre-initialization". This means that the channel will set the configured "Receive Status" after it has been successfully connected, using <u>CAN_Initialize</u> (**class-method**: Initialize). Once the channel is disconnected, further initializations of this channel are done with the default value of this parameter (ON). This is usefull to avoid receiving messages immediately after connection, or before the receive filter is configured according with the needs of each application.

PCAN_CONTROLLER_NUMBER

Access:

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: NA.

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

REMARKS: This parameter can be used with an initialized or uninitialized channel.

PCAN_TRACE_LOCATION

Access: 🗷 🕅

Description: This value is used to set the folder location on a computer for the PCAN-Trace file generated by the PCAN-Basic API, within a trace session. If a trace session is active (a trace file is being written), PCAN_TRACE_LOCATION instructs the API to close the current trace file and to start recording data again with the new folder information.

Possible values: This value must be a fully-qualified and valid path to an **existing directory** on the executing computer. There is no limit for the length of the string but it is recommended to use a length not bigger than MAX_PATH. For more information see ¬<u>Naming Files</u>, <u>Paths, and Namespaces</u>. Passing an **empty string** ("", NULL value) instructs the API to use the default value for this parameter.

Default value: Calling process's folder.

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

REMARKS: Note that the name of the trace file cannot be specified. The file uses the name of the current connection (PCAN-Channel's name) plus a file counter (e.g. **PCAN_PCIBUS1_1.trc**), though it can be enhanced by issuing the parameter **PCAN_TRACE_CONFIGURE**.

PCAN_TRACE_STATUS

Access: 🗷 🕅

Description: This value is used to control the activity status of a trace session within the PCAN-Basic API. If the trace status is set to ON without having set a location for the trace file or without having configured the storing mode, then the session process will start with the default values. Trying to activate a trace session can fail if overwriting is not set and a file with the same name already exists.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid. **Default value:** Disabled (PCAN_PARAMETER_OFF).

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

PCAN_TRACE_SIZE

Access: R 🕅

Description: This value is used to set the maximum size, in megabytes, that a single trace file can have. Trying to set the size for a trace file will fail if the trace session is active.

Possible values: A number in the range [1..100], representing the amount of megabytes. Passing a value of **0** instructs the API to use the default value for this parameter. Trying to set a size bigger than 100 will fail.

Default value: 10 megabytes.

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

PCAN_TRACE_CONFIGURE

Access: R 🕅

Description: This value is used to configure the trace process and the file generated in a trace session within the PCAN-Basic API. Trying to configure a trace file will fail if the trace session is active.

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

- **TRACE_FILE_SINGLE**: This value represents the default trace configuration. It defines the use of a single trace file as output. When the tracing process starts, the file will be filled out with messages until the size of the file reaches the configured maximum size (see <u>PCAN_TRACE_SIZE</u>). The tracing process is then automatically stopped.
- **TRACE_FILE_SEGMENTED**: This value indicates the API to
keep tracing information by using several files. When the trace file being used reaches the maximum configured file size (see <u>PCAN_TRACE_SIZE</u>), then a new file is automatically created and the tracing process continues.

TRACE_FILE_DATE: This value instruct the API to use the start date information within the name of the trace file. The format used is YYYYMMDD, four digits for year, the next two for the month, and the last two for the day, e.g.
 "20130228_PCAN_USBBUS_1" for the 28th February 2013. If both, TRACE_FILE_DATE and TRACE_FILE_TIME are configured, the file name starts always with the date:

"20130228140733 PCAN USBBUS1 1.trc".

- TRACE_FILE_TIME: This value instruct the API to use the start time information within the name of the trace file. The format used is HHMMSS, two digits for the hour (24 hours format), the next two for the minutes, and the last two for the seconds, e.g.
 "140733_PCAN_USBBUS_1" for 14:07:33 (02:07:33 PM). If both, TRACE_FILE_DATE and TRACE_FILE_TIME are configured, the file name starts always with the date:
 "20130228140733 PCAN_USBBUS1 1.trc".
- **TRACE_FILE_OVERWRITE**: This value causes the overwriting of an existing trace file when a new trace session is started. If this value is not configured, trying to start a tracing process will fail, if the file name to generate is the same as one used by an existing file.

Default value: TRACE_FILE_SINGLE (Single file, not overwriting, with standard name).

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

PCAN_CHANNEL_IDENTIFYING

Access: R 🕅

Description: This value is used to control the status of the "channel identifying procedure" on USB devices within the PCAN-Basic API. The procedure consists in blinking the LED associated to the given

channel.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

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

REMARKS: This parameter can be used with an initialized or uninitialized channel. This identifying procedure is only available for USB based hardware (PCAN-USB, PCAN-USB Pro and PCAN-USB Hub). The blinking of the LED can be different according to the kind of hardware used (in color and blink rate). Only one channel can blink simultaneously.

PCAN_CHANNEL_FEATURES

Access:

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:

- **FEATURE_FD_CAPABLE**: This value indicates that the hardware represented by a PCAN Channel is FD capable (it supports flexible data rate).
- **FEATURE_DELAY_CAPABLE:** This value indicates that the hardware represented by a PCAN Channel allows the configuration of a delay between sending frames.

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

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

REMARKS: This parameter can be used with an initialized or uninitialized channel. FD Hardware must be initialized with <u>CAN_InitializeFD</u> (class-method: InitializeFD) in order to use their FD capabilities. In same way, the functions <u>CAN_ReadFD</u> (classmethod: <u>ReadFD</u>) and <u>CAN_WriteFD</u> (class-method: <u>WriteFD</u>) have to be used for data transmission.

PCAN_BITRATE_ADAPTING

Access: 🗷 🕅

Description: This value is used to force an initialization process to succeed, even if the PCAN-Channel is being used by a PCAN-View with a different or unknown bit rate. The initialization function will return a PCAN_ERROR_CAUTION error, when the bit rate passed as parameter was different than that being used.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

PCAN-Device: All **Plug-n-Play** PCAN devices (excluding PCAN_NONEBUS channel).

REMARKS: This parameter can be set only on **uninitialized** channels. After connecting, an application can get the bit rate currently used by calling <u>CAN_GetValue</u> (**class-method**: <u>GetValue</u>) with the parameters <u>PCAN_SPEED_QUERY</u>, for standard CAN channels, or <u>PCAN_SPEED_QUERY_FD</u> for CAN-FD channels.

PCAN_BITRATE_INFO

Access:

Description: This value is used to read the currently configured communication speed, as BTR0-BTR1 value, of a PCAN Channel connected as standard CAN.

Possible values: A number in the range [0..65535] (Word Value).

Default value: N/A.

PCAN-Device: All PCAN devices (excluding PCAN_NONEBUS

channel).

REMARKS: This parameter can be used only on PCAN-Channels that have been initialized with the function <u>CAN_Initialize</u> (**class-method:** <u>Initialize</u>).

PCAN_BITRATE_INFO_FD

Access:

Description: This value is used to read the currently configured communication speed, as a parameterized string value (FD bit rate string), of a PCAN Channel connected as CAN FD.

Possible values: a String representing a FD bit rate. See <u>FD Bit rate</u> for more information.

Default value: N/A.

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

REMARKS: This parameter can be used only on PCAN-Channels that have been initialized with the function <u>CAN_InitializeFD</u> (**class-method:** <u>InitializeFD</u>).

PCAN_BUSSPEED_NOMINAL

Access: 🗷

Description: This value is used to read the currently configured nominal CAN Bus speed, as bits/second.

Possible values: a number representing the nominal CAN bus speed being used, as the amount of bits that can be transmitted in a second.

Default value: N/A.

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

PCAN_BUSSPEED_DATA

Access:

Description: This value is used to read the currently configured CAN data speed (Bit Rate Switch), as bits/second.

Possible values: a number representing the CAN data speed configured for BRS, as the amount of bits that can be transmitted in a second.

Default value: N/A.

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

REMARKS: The speed used for data transmission is the same as the nominal speed on devices that don't support flexible data rate.

PCAN_IP_ADDRESS

Access:

Description: This value is used to read the address used by a device for IP communication.

Possible values: a string representing the IP address of a device, in IPv4 format.

Default value: N/A.

PCAN-Device: PCAN-LAN (PCAN-Gateway Ethernet/Wireless devices).

PCAN_LAN_SERVICE_STATUS

Access:

Description: This value is used to get the running status of the System-Service that is part of the Virtual PCAN-Gateway solution.

Possible values: This parameter can have one of these values: SERVICE_STATUS_RUNNING, SERVICE_STATUS_STOPPED.

Default value: N/A.

PCAN-Device: Default channel Only (PCAN_NONEBUS).

REMARKS: This parameter is only relevant in PCAN-LAN environments (using PCAN-Gateway Ethernet/Wireless devices).

PCAN_ALLOW_STATUS_FRAMES

Access: R 🕅

Description: This parameter helps the user to allow / disallow the reception of messages of type "PCAN_MESSAGE_STATUS" within a PCAN Channel. When "PCAN_ALLOW_STATUS_FRAMES" is active (ON), generated Status messages are forwarded from the driver to the user application through the CAN_Read/CAN_ReadFD functions (class-methods: Read, ReadFD). Otherwise, the reception of Status frames is deactivated within the driver for this specific user application.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Enabled (PCAN_PARAMETER_ON).

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

PCAN_ALLOW_RTR_FRAMES

Access: R 🕅

Description: This parameter helps the user to allow / disallow the reception of messages of type "PCAN_MESSAGE_RTR" within a PCAN Channel. When "PCAN_ALLOW_RTR_FRAMES" is active (ON), incoming RTR messages are forwarded to the user application through the <u>CAN_Read/CAN_ReadFD</u> functions (**class-methods:** <u>Read</u>, <u>ReadFD</u>). Otherwise, the reception of RTR frames is deactivated within the driver for this specific user application.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Enabled (PCAN_PARAMETER_ON).

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

PCAN_ALLOW_ERROR_FRAMES

Access: R 🕅

Description: This parameter helps the user to allow / disallow the reception of messages of type "PCAN_MESSAGE_ERRFRAME" within a PCAN Channel. When "PCAN_ALLOW_ERROR_FRAMES" is active (ON), generated Error messages are forwarded from the driver to the user application through the CAN_Read/CAN_ReadFD functions (class-methods: Read, ReadFD). Otherwise, the reception of Error frames is deactivated within the driver for this specific user application.

Possible values: The value must be one of the following values: **PCAN_PARAMETER_OFF**, **PCAN_PARAMETER_ON**. Note that other values will considered invalid.

Default value: Disabled (PCAN_PARAMETER_OFF).

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

PCAN_INTERFRAME_DELAY

Access: R 🕅

Description: This parameter is used to configure a delay, **in microseconds**, between sending frames. When this value is value is greater than 0, the driver includes that value as a pause between each written CAN frame. Otherwise, the CAN frames are sent as fast as possible. **Possible values**: The parameter has a value range between [0..n], where **n** is the maximum value supported by the Firmware. If the maximum value supported by the firmware is lower than the entered one, the value will be truncated.

Default value: 0 (disabled).

PCAN-Device: All FPGA based PCAN devices.

PCAN_ACCEPTANCE_FILTER_11BIT

Access: R 🕅

Description: This parameter is used to configure the reception filter of a PCAN channel with a specific 11-bit acceptance code and mask, as specified for the acceptance filter of the SJA1000 CAN controller. The acceptance **code** and **mask** are coded together in a 64-bit value, each of them using 4 bytes (Intel/Little-Endian format). The acceptance code is stored at the most significant bytes.

Possible values: Both parameter parts, **code** and **mask**, have a value range between [0..**16838**]. This means, the maximum value of this parameter as 64-bit value is **70364449226751**, that is, hexadecimal **00003FFF**00003FFFh. The mask uses the bit value '**1**' as "**don't care bit**".

Default value: 000000000007FFh (no filtering).

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

REMARKS: This parameter is particularly adapted to the SJA1000 CAN controller. Reception filters can also be configured with the function <u>CAN_FilterMessages</u>(class-method: <u>FilterMessages</u>). Note that after a PCAN Channel is initialized, the status of its filter is fully opened. According with the current filter status, setting this parameter causes the following behavior:

- Filter status is PCAN_FILTER_OPEN: The filter is automatically closed and then configured with the given acceptance filter.
- Filter status is PCAN_FILTER_CLOSE: The filter is set to the

given acceptance filter.

• Filter status is PCAN_FILTER_CUSTOM: The filter is expanded with the given acceptance filter. If a different acceptance code is required instead of expanding the current one, the filter has to be closed first before setting the acceptance filter. To do this use the parameter <u>PCAN_MESSAGE_FILTER</u>.

PCAN_ACCEPTANCE_FILTER_29BIT

Access: R 🕅

Description: This parameter is used to configure the reception filter of a PCAN channel with a specific 29-bit acceptance code and mask, as specified for the acceptance filter of the SJA1000 CAN controller. The acceptance **code** and **mask** are coded together in a 64-bit value, each of them using 4 bytes (Intel/Little-Endian format). The acceptance code is stored at the most significant bytes.

Default value: 00000001FFFFFFh (no filtering).

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

REMARKS: This parameter is particularly adapted to the SJA1000 CAN controller. Reception filters can also be configured with the function <u>CAN_FilterMessages</u>(class-method: <u>FilterMessages</u>). Note that after a PCAN Channel is initialized, the status of its filter is fully opened. According with the current filter status, setting this parameter causes the following behavior:

- Filter status is PCAN_FILTER_OPEN: The filter is automatically closed and then configured with the given acceptance filter.
- Filter status is PCAN_FILTER_CLOSE: The filter is set to the given acceptance filter.

 Filter status is PCAN_FILTER_CUSTOM: The filter is expanded with the given acceptance filter. If a different acceptance code is required instead of expanding the current one, the filter has to be closed first before setting the acceptance filter. To do this use the parameter <u>PCAN_MESSAGE_FILTER</u>.

■ See Also

<u>CAN_GetValue</u> (class-method: <u>GetValue</u>) <u>CAN_SetValue</u> (class-method: <u>SetValue</u>)

Parameter Value Definitions

Naming Files, Paths, and Namespaces

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send $\underline{\text{feedback}}$ to this documentation

🕇 Home

< ^ >

TPCANMessageType

Represents the type of a CAN message. According with the programming language, this type can be a group of defined values or an enumeration.

Syntax

⊟ C++	
#define	TPCANMessageType BYTE
<pre>#define #define #define #define #define #define #define #define #define #define #define</pre>	PCAN_MESSAGE_STANDARD 0x00 PCAN_MESSAGE_RTR 0x01 PCAN_MESSAGE_EXTENDED 0x02 PCAN_MESSAGE_FD 0x04 PCAN_MESSAGE_BRS 0x08 PCAN_MESSAGE_ESI 0x10 PCAN_MESSAGE_ERRFRAME 0x40 PCAN_MESSAGE_STATUS 0x80
🗏 Pascal O	0
{\$Z1} TPCANMes PCAN PCAN PCAN PCAN PCAN PCAN PCAN PCAN	ssageType = (N_MESSAGE_STANDARD = \$00, N_MESSAGE_RTR = \$01, N_MESSAGE_EXTENDED = \$02, N_MESSAGE_FD = \$04, N_MESSAGE_BRS = \$08, N_MESSAGE_ESI = \$10, N_MESSAGE_ESI = \$10, N_MESSAGE_ERRFRAME = \$40, N_MESSAGE_STATUS = \$80
⊢ C #	

```
[Flags]
public enum TPCANMessageType : byte
{
    PCAN_MESSAGE_STANDARD = 0x00,
    PCAN_MESSAGE_RTR = 0x01,
    PCAN_MESSAGE_EXTENDED = 0x02,
    PCAN_MESSAGE_FD = 0x04,
    PCAN_MESSAGE_BRS = 0x08,
    PCAN_MESSAGE_ESI = 0x10,
    PCAN_MESSAGE_ERRFRAME = 0x40,
    PCAN_MESSAGE_STATUS = 0x80,
```

□ C++ / CLR

```
[Flags]
public enum class TPCANMessageType : Byte
{
    PCAN_MESSAGE_STANDARD = 0x00,
    PCAN_MESSAGE_RTR = 0x01,
    PCAN_MESSAGE_EXTENDED = 0x02,
    PCAN_MESSAGE_FD = 0x04,
    PCAN_MESSAGE_BRS = 0x08,
    PCAN_MESSAGE_ESI = 0x10,
    PCAN_MESSAGE_ESI = 0x10,
    PCAN_MESSAGE_ERRFRAME = 0x40,
    PCAN_MESSAGE_STATUS = 0x80,
}
```

```
};
```

Visual Basic

```
<Flags()> _
Public Enum TPCANMessageType As Byte
    PCAN_MESSAGE_STANDARD = &H0
    PCAN_MESSAGE_RTR = &H1
    PCAN_MESSAGE_EXTENDED = &H2
    PCAN_MESSAGE_FD = &H4
```

```
PCAN_MESSAGE_BRS = &H8
PCAN_MESSAGE_ESI = &H10
PCAN_MESSAGE_ERRFRAME = &H40
PCAN_MESSAGE_STATUS = &H80
```

End Enum

Bereich Python

```
TPCANMessageType = c_ubyte
```

```
PCAN_MESSAGE_STANDARD = TPCANMessageType(0x00)
PCAN_MESSAGE_RTR = TPCANMessageType(0x01)
PCAN_MESSAGE_EXTENDED = TPCANMessageType(0x02)
PCAN_MESSAGE_FD = TPCANMessageType(0x04)
PCAN_MESSAGE_BRS = TPCANMessageType(0x08)
PCAN_MESSAGE_ESI = TPCANMessageType(0x10)
PCAN_MESSAGE_ERRFRAME = TPCANMessageType(0x40)
PCAN_MESSAGE_STATUS = TPCANMessageType(0x80)
```

Remarks

Several message types can be combined (Bit mask).

Note that messages with type PCAN_MESSAGE_FD, PCAN_MESSAGE_BRS, PCAN_MESSAGE_ESI, or a combination of them, can only be sent/received using the FD functions <u>CAN_ReadFD</u> and <u>CAN_WriteFD</u> (class-methods: <u>ReadFD</u>, <u>WriteFD</u>).

Values

Name	Value	Description
PCAN_MESSAGE_STANDARD	0	The PCAN message is a CAN Standard Frame (11-bit identifier).
PCAN_MESSAGE_RTR	1	The PCAN message

		is a CAN Remote- Transfer-Request Frame.
PCAN_MESSAGE_EXTENDED	2	The PCAN message is a CAN Extended Frame (29-bit identifier).
PCAN_MESSAGE_FD	4	The PCAN message represents a FD frame in terms of CiA specifications.
PCAN_MESSAGE_BRS	8	The PCAN message represents a FD bit rate switch (CAN data at a higher bit rate).
PCAN_MESSAGE_ESI	16	The PCAN message represents a FD error state indicator(CAN FD transmitter was error active).
PCAN_MESSAGE_ERRFRAME	64	The PCAN message represents an error frame. See <u>Error</u> <u>Frames</u> for more information.
PCAN_MESSAGE_STATUS	128	The PCAN message represents a PCAN status message.

∃ See Also

CAN_Read (class-method: Read)

CAN_ReadFD (class-method: ReadFD)

<u>CAN_Write</u> (class-method: <u>Write</u>) <u>CAN_WriteFD</u> (class-method: <u>WriteFD</u>)

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🔒 Home

< ^ >

ТРСАНТуре

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

Syntax

```
= C++
#define TPCANType BYTE
#define PCAN_TYPE_ISA 0x01
#define PCAN_TYPE_ISA_SJA 0x09
#define PCAN_TYPE_ISA_PHYTEC 0x04
#define PCAN_TYPE_DNG 0x02
#define PCAN_TYPE_DNG_EPP 0x03
#define PCAN_TYPE_DNG_SJA 0x05
#define PCAN_TYPE_DNG_SJA_EPP 0x06
```

```
PCAN_TYPE_ISA_PHYTEC = $04,
PCAN_TYPE_DNG = $02,
PCAN_TYPE_DNG_EPP = $03,
PCAN_TYPE_DNG_SJA = $05,
PCAN_TYPE_DNG_SJA_EPP = $06
```

);

□ C#

public enum TPCANType : byte

```
{
    PCAN_TYPE_ISA = 0x01,
    PCAN_TYPE_ISA_SJA = 0x09,
    PCAN_TYPE_ISA_PHYTEC = 0x04,
    PCAN_TYPE_DNG = 0x02,
    PCAN_TYPE_DNG_EPP = 0x03,
    PCAN_TYPE_DNG_SJA = 0x05,
    PCAN_TYPE_DNG_SJA_EPP = 0x06,
}
```

```
□ C++ / CLR
```

}

```
public enum class TPCANType : Byte
{
    PCAN_TYPE_ISA = 0x01,
    PCAN_TYPE_ISA_SJA = 0x09,
    PCAN_TYPE_ISA_PHYTEC = 0x04,
    PCAN_TYPE_DNG = 0x02,
    PCAN_TYPE_DNG_EPP = 0x03,
    PCAN_TYPE_DNG_SJA = 0x05,
    PCAN_TYPE_DNG_SJA_EPP = 0x06,
};
```

```
Visual Basic
```

```
Public Enum TPCANType As Byte
    PCAN_TYPE_ISA = &H1
    PCAN_TYPE_ISA_SJA = &H9
    PCAN_TYPE_ISA_PHYTEC = &H4
    PCAN_TYPE_DNG = &H2
    PCAN_TYPE_DNG_EPP = &H3
    PCAN_TYPE_DNG_SJA = &H5
    PCAN_TYPE_DNG_SJA_EPP = &H6
End Enum
```

B Python

```
TPCANType = c_ubyte
```

```
PCAN_TYPE_ISA = TPCANType(0x01)
PCAN_TYPE_ISA_SJA = TPCANType(0x09)
PCAN_TYPE_ISA_PHYTEC = TPCANType(0x04)
PCAN_TYPE_DNG = TPCANType(0x02)
PCAN_TYPE_DNG_EPP = TPCANType(0x03)
PCAN_TYPE_DNG_SJA = TPCANType(0x05)
PCAN_TYPE_DNG_SJA_EPP = TPCANType(0x06)
```

Values

Name	Value	Description
PCAN_TYPE_ISA	1	PCAN-ISA 82C200.
PCAN_TYPE_ISA_SJA	9	PCAN-ISA SJA1000.
PCAN_TYPE_ISA_PHYTEC	4	PHYTEC ISA .
PCAN_TYPE_DNG	2	PCAN-Dongle 82C200.
PCAN_TYPE_DNG_EPP	3	PCAN-Dongle EPP 82C200.
PCAN_TYPE_DNG_SJA	5	PCAN-Dongle SJA1000.
PCAN_TYPE_DNG_SJA_EPP	6	PCAN-Dongle EPP SJA1000.

■ See Also

PCAN_Initialize (class-method: Initialize)

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

Home

< ^ >

TPCANMode

Represents a PCAN filter mode. According with the programming language, this type can be a group of defined values or an enumeration.

Syntax

∃ C++
#define TPCANMode BYTE
<pre>#define PCAN_MODE_STANDARD PCAN_MESSAGE_STANDAM #define PCAN_MODE_EXTENDED PCAN_MESSAGE_EXTENDM</pre>
B Pascal OO
{\$Z1}
TPCANMode = (
<pre>PCAN_MODE_STANDARD = Byte(PCAN_MESSAGE_STAN</pre>
<pre>PCAN_MODE_EXTENDED = Byte(PCAN_MESSAGE_EXTENDED = Byte(PCAN_MESSAGE_E</pre>

```
);
```

```
B C#
public enum TPCANMode : byte
{
    PCAN_MODE_STANDARD = TPCANMessageType.PCAN_
    PCAN_MODE_EXTENDED = TPCANMessageType.PCAN_
}
```

```
□ C++ / CLR
```

```
public enum class TPCANMode : Byte
{
     PCAN_MODE_STANDARD = TPCANMessageType::PCAN
     PCAN_MODE_EXTENDED = TPCANMessageType::PCAN
};
```

Visual Basic

```
Public Enum TPCANMode As Byte
    PCAN_MODE_STANDARD = TPCANMessageType.PCAN_
    PCAN_MODE_EXTENDED = TPCANMessageType.PCAN_
End Enum
```

Bereich Python

TPCANMode = c_ubyte

```
PCAN_MODE_STANDARD = PCAN_MESSAGE_STANDARD
PCAN_MODE_EXTENDED = PCAN_MESSAGE_EXTENDED
```

Values

Name	Value	Description
PCAN_MODE_STANDARD	0	Mode is Standard (11-bit identifier).
PCAN_MODE_EXTENDED	2	Mode is Extended (29-bit identifier).

∃ See Also

CAN_FilterMessages (class-method: FilterMessages)

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

Home
 Hom

< ^ >

TPCANBaudrate

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

Syntax

⊟ C++		
#define	TPCANBaudrate WORD	
#define	PCAN_BAUD_1M 0x0014	
#define	PCAN_BAUD_800K 0x0016	
#define	PCAN_BAUD_500K 0x001C	
#define	PCAN_BAUD_250K 0x011C	
#define	PCAN_BAUD_125K 0x031C	
#define	PCAN_BAUD_100K 0x432F	
#define	PCAN_BAUD_95K 0xC34E	
#define	PCAN_BAUD_83K 0x852B	
#define	PCAN_BAUD_50K 0x472F	
#define	PCAN_BAUD_47K 0x1414	
#define	PCAN_BAUD_33K 0x8B2F	
#define	PCAN_BAUD_20K 0x532F	
#define	PCAN_BAUD_10K 0x672F	
#define	PCAN_BAUD_5K 0x7F7F	
∫\$721		
$\chi \Psi = \zeta$		
	$\frac{1}{1000} = \frac{1}{1000}$	
PLA	N BAUD IM = 30014	

PCAN_BAUD_1M = \$0014, PCAN_BAUD_800K = \$0016, PCAN_BAUD_500K = \$001C, PCAN_BAUD_250K = \$011C,

```
PCAN_BAUD_125K = $031C,
PCAN_BAUD_100K = $432F,
PCAN_BAUD_95K = $C34E,
PCAN_BAUD_83K = $852B,
PCAN_BAUD_50K = $472F,
PCAN_BAUD_50K = $472F,
PCAN_BAUD_47K = $1414,
PCAN_BAUD_33K = $8B2F,
PCAN_BAUD_20K = $532F,
PCAN_BAUD_20K = $532F,
PCAN_BAUD_10K = $672F,
PCAN_BAUD_5K = $7F7F
```

);

□ C#

```
public enum TPCANBaudrate : ushort
{
    PCAN_BAUD_1M = 0 \times 0014,
    PCAN_BAUD_800K = 0x0016,
    PCAN BAUD 500K = 0 \times 001C,
    PCAN_BAUD_250K = 0x011C,
    PCAN_BAUD_125K = 0x031C,
    PCAN_BAUD_100K = 0x432F,
    PCAN_BAUD_95K = 0xC34E,
    PCAN_BAUD_83K = 0x852B,
    PCAN_BAUD_50K = 0x472F,
    PCAN_BAUD_47K = 0x1414,
    PCAN_BAUD_33K = 0x8B2F,
    PCAN BAUD 20K = 0x532F,
    PCAN_BAUD_10K = 0x672F,
    PCAN_BAUD_5K = 0x7F7F,
}
```

```
□ C++ / CLR
```

public enum class TPCANBaudrate : UInt16
{



};

Visual Basic

```
Public Enum TPCANBaudrate As UInt16
    PCAN_BAUD_1M = \&H14
    PCAN BAUD 800K = \&H16
    PCAN_BAUD_500K = \&H1C
    PCAN_BAUD_250K = \&H11C
    PCAN BAUD 125K = &H31C
    PCAN_BAUD_100K = \&H432F
    PCAN BAUD 95K = \&C34E
    PCAN BAUD 83K = \&852B
    PCAN BAUD 50K = \&H472F
    PCAN BAUD 47K = \&1414
    PCAN BAUD 33K = \&8B2F
    PCAN BAUD 20K = \&H532F
    PCAN BAUD 10K = \&H672F
    PCAN BAUD 5K = \&H7F7F
End Enum
```

B Python

TPCANBaudrate = c_ushort

<pre>PCAN_BAUD_1M = TPCANBaudrate(0x0014)</pre>
<pre>PCAN_BAUD_800K = TPCANBaudrate(0x0016)</pre>
<pre>PCAN_BAUD_500K = TPCANBaudrate(0x001C)</pre>
<pre>PCAN_BAUD_250K = TPCANBaudrate(0x011C)</pre>
<pre>PCAN_BAUD_125K = TPCANBaudrate(0x031C)</pre>
<pre>PCAN_BAUD_100K = TPCANBaudrate(0x432F)</pre>
<pre>PCAN_BAUD_95K = TPCANBaudrate(0xC34E)</pre>
<pre>PCAN_BAUD_83K = TPCANBaudrate(0x852B)</pre>
<pre>PCAN_BAUD_50K = TPCANBaudrate(0x472F)</pre>
<pre>PCAN_BAUD_47K = TPCANBaudrate(0x1414)</pre>
<pre>PCAN_BAUD_33K = TPCANBaudrate(0x8B2F)</pre>
<pre>PCAN_BAUD_20K = TPCANBaudrate(0x532F)</pre>
<pre>PCAN_BAUD_10K = TPCANBaudrate(0x672F)</pre>
$PCAN_BAUD_5K = TPCANBaudrate(0x7F7F)$

■ Values

Name	Value	Description
PCAN_BAUD_1M	20	1 MBit/s.
PCAN_BAUD_800K	22	800 kBit/s.
PCAN_BAUD_500K	28	500 kBit/s.
PCAN_BAUD_250K	284	250 kBit/s.
PCAN_BAUD_125K	796	125 kBit/s.
PCAN_BAUD_100K	17199	100 kBit/s.
PCAN_BAUD_95K	49998	95,238 kBit/s.
PCAN_BAUD_83K	34091	83,333 kBit/s.
PCAN_BAUD_50K	18223	50 kBit/s.
ĺ	1	1

PCAN_BAUD_47K	5140	47,619 kBit/s.
PCAN_BAUD_33K	35631	33,333 kBit/s.
PCAN_BAUD_20K	21295	20 kBit/s.
PCAN_BAUD_10K	26415	10 kBit/s.
PCAN_BAUD_5K	32639	5 kBit/s.

∃ See Also

CAN_Initialize (class-method: Initialize)

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🔒 Home

< ^ >

TPCANBitrateFD

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

Syntax

□ C++

#define TPCANBitrateFD LPSTR

Pascal OO

TPCANBitrateFD = String;

□ C#

using TPCANBitrateFD = System.String;

□ C++ / CLR

#define TPCANBitrateFD System::String^

Visual Basic

Imports TPCANBitrateFD = System.String

Bereich Python

TPCANBitrateFD = c_char_p

🗉 Remarks

.NET Framework programming languages:

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

Aliases are defined in the <u>Peak.Can.Basic</u> 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 <u>Peak.Can.Basic</u> Namespace. Otherwise, just use the native type, which in this case is a String.

C#:

```
using System;
using <u>Peak.Can.Basic;</u>
using TPCANBitrateFD = System.String; // Alias's d
```

Visual Basic:

```
Imports System
Imports <u>Peak.Can.Basic</u>
Imports TPCANBitrateFD = System.String ' Alias dec
```

■ See Also

FB Bit rate Parameter Definitions

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

Home

< ^ >

TPCANTimestampFD

Represents the timestamp of a CAN message with flexible data rate. The time-stamp contains the number of microseconds since the start of Windows.

Syntax

∃ C++			
#define TPCANTimestampFD UINT64			
E Pascal OO			
<pre>TPCANTimestampFD = UInt64;</pre>			
∃ C#			
<pre>using TPCANTimestampFD = System.UInt64;</pre>			
∃ C++ / CLR			
<pre>#define TPCANTimestampFD System::UInt64</pre>			
🛛 Visual Basic			
<pre>Imports TPCANTimestampFD = System.UInt64</pre>			
B Python			
TPCANTimestampFD = c_ulonglong			

Remarks

.NET Framework programming languages:

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

Aliases are defined in the <u>Peak.Can.Basic</u> 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 <u>Peak.Can.Basic</u> Namespace. Otherwise, just use the native type, which in this case is a UInt64.

C#:

```
using System;
using <u>Peak.Can.Basic;</u>
using TPCANTimestampFD = System.UInt64; // Alias's
```

Visual Basic:

```
Imports System
Imports <u>Peak.Can.Basic</u>
Imports TPCANTimestampFD = System.UInt64 ' Alias d
```

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

<^>

Methods

The methods defined for the classes <u>PCANBasic</u> and <u>TPCANBasic</u> are divided in 4 groups of functionality. Note that, with exception of the method version for Python, these methods are static and can be called in the name of the class, without instantiation.

Connection

	Function	Description
* 80	<u>Initialize</u>	Initializes a PCAN Channel.
= ⊘ 80	<u>InitializeFD</u>	Initializes a FD capable PCAN Channel.
= ⊘	<u>Uninitialize</u>	Uninitializes a PCAN Channel.

Configuration

	Function	Description
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	<u>SetValue</u>	Sets a configuration or information value within a PCAN Channel.
=♦ 80	FilterMessages	Configures the message's reception filter of a PCAN Channel.

Information

	Function	Description
2 2	<u>GetValue</u>	Retrieves information from a PCAN

		Channel.
=¢ 83	<u>GetStatus</u>	Retrieves the current BUS status of a PCAN Channel.
2 8	<u>GetErrorText</u>	Gets a descriptive text for an error code.

Communication

	Function	Description
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	<u>Read</u>	Reads a CAN message from the receive queue of a PCAN Channel.
2 00	<u>ReadFD</u>	Reads a CAN message from the receive queue of a FD capable PCAN Channel.
	<u>Write</u>	Transmits a CAN message using a connected PCAN Channel.
=¢ ©	<u>WriteFD</u>	Transmits a CAN message using a connected FD capable PCAN Channel.
=¢ 8	<u>Reset</u>	Resets the receive and transmit queues of a PCAN Channel.

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

▲ Home

< ^ >

Initialize

Initializes a PCAN Channel.

Overloads

	Function	Description
୍ବ ପ୍ରତି	Initialize(TPCANHandle, TPCANBaudrate)	Initializes a Plug-And-Play PCAN Channel.
	Initialize(TPCANHandle, TPCANBaudrate, TPCANType, UInt32, UInt16)	Initializes a Not-Plug-And-Play PCAN Channel.

Copyright $\textcircled{\mbox{\sc c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🔒 Home

< ^ >

Initialize(TPCANHandle, TPCANBaudrate)

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

Syntax

■ Pascal OO
class function Initialize(
 Channel: TPCANHandle;
 Btr0Btr1: TPCANBaudrate
): TPCANStatus; overload;

□ C#

public static extern <u>TPCANStatus</u> <u>Initialize(</u> <u>TPCANHandle</u> Channel, <u>TPCANBaudrate</u> Btr0Btr1);

C++ / CLR

static <u>TPCANStatus</u> Initialize(<u>TPCANHandle</u> Channel, <u>TPCANBaudrate</u> Btr0Btr1);

Bisual Basic

Public Shared Function <u>Initialize(</u> ByVal Channel As <u>TPCANHandle</u>, _ ByVal Btr0Btr1 As <u>TPCANBaudrate</u>) As <u>TPCANS1</u> End Function

Bereich Python

```
def <u>Initialize(</u>
self,
Channel,
Btr0Btr1)
```

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Btr0Btr1	The speed for the communication (BTR0BTR1 code).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_CAUTION:	Indicates that the channel has been initialized but at a different bit rate as the given one.
PCAN_ERROR_ILLHANDLE:	Indicates that the desired PCAN Channel is not valid. Check the list of <u>valid</u> <u>Channels</u> .
PCAN_ERROR_ILLHW:	Indicates that the desired PCAN Channel is not available.
PCAN_ERROR_ILLOPERATION:	Indicates that an action cannot be executed due to the state of the hardware. Possible causes are:
	 The desired PCAN- Channel is a LAN Channel, which uses a

	different bit rate than the specified.
PCAN_ERROR_INITIALIZE:	Indicates that the desired PCAN Channel cannot be connected because it is already in use (PCAN-Basic / PCAN-Light environment).
PCAN_ERROR_NETINUSE:	Indicates that the desired PCAN-Channel is being used with a different bit rate (PCAN-View).
PCAN_ERROR_HWINUSE:	Indicates that the desired PCAN-Channel is being used (CanApi2 connection).
PCAN_ERROR_NODRIVER:	The driver needed for connecting the desired PCAN Channel is not loaded.

Remarks

As indicated by its name, the <u>Initialize</u> method initiates a PCAN 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 PCAN_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCAN Channel means:

- to reserve the Channel for the calling application/process.
- to allocate channel resources, like receive and transmit queues.
- to register/connect the Hardware denoted by the channel handle.
- to check and adapt the bus speed, if the Channel is already

in use. (Only if the Channel was pre-configured as Bitrate Adapting; see: <u>Bitrate-Adapting Parameter</u>).

- to set the channel in Listen-Only mode. (Only *if the channel* was pre-configured as Listen-Only; see: Listen-Only Parameter).
- to open the messages filter for the application/process.
- to set-up the default values of the different parameters (See <u>GetValue</u>).
- to set the Receive Status of the channel. (*Pre-configured* value; see: <u>Receive Status Parameter</u>).

Different than the PCAN-Light API, the Initialization process will fail if an application try to initialize a PCAN-Channel that has been initialized already within the same process.

Take in consideration that initializing a channel causes a reset of the CAN hardware , when the bus status is other than OK. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT, are removed.

PCAN-LAN Channels

A PCAN-LAN channel doesn't allow changing the bit rate using PCAN-Basic. In order to connect a PCAN-LAN Channel it is necessary to know the bit rate of the PCAN-Gateway device that is represented by that channel. If the bit rate is not known, the parameter <u>Bitrate-Adapting</u> should be used.

Python Notes

- Class-Method: Different than the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.
- Plug-&-Play and No-Plug-&-Play hardware: In order to initialize a channel which represents a Plug-&-Play PCAN device, only the Channel-handle and bit rate parameters are needed. The other parameters will be assigned their default values. For No-Plug-&-Play devices, all parameters have to be entered.
Example

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

C#:

```
<u>TPCANStatus</u> result;
StringBuilder strMsg;
// The Plug & Play Channel (PCAN-PCI) is initializ
11
result = PCANBasic.<u>Initialize(PCANBasic.PCAN_PCIBU</u>
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    strMsg = new StringBuilder(256);
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("PCAN-PCI (Ch-2) was initializ
// All initialized channels are released
//
PCANBasic.Uninitialize(PCANBasic.PCAN_NONEBUS);
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
// The Plug & Play Channel (PCAN-PCI) is initializ
//
result = PCANBasic::Initialize(PCANBasic::PCAN_PCI
```

```
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    strMsg = gcnew StringBuilder(256);
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("PCAN-PCI (Ch-2) was initiali
// All initialized channels are released
//
PCANBasic::Uninitialize(PCANBasic::PCAN_NONEBUS);
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
' The Plug & Play Channel (PCAN-PCI) is initialize
' result = PCANBasic.Initialize(PCANBasic.PCAN_PCIBU
If result <> TPCANStatus.PCAN_ERROR_OK Then
    ' An error occurred, get a text describing the
    ' strMsg = New StringBuilder(256)
    PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
    MessageBox.Show(strMsg.ToString)
Else
    MessageBox.Show("PCAN-PCI (Ch-2) was initializ
End If
' All initialized channels are released
'
PCANBasic.<u>Uninitialize(PCANBasic.PCAN_NONEBUS)</u>
```

Pascal OO:

```
var
 result : TPCANStatus;
 strMsg: array [0..256] of Char;
begin
    // The Plug & Play Channel (PCAN-PCI) is initi
    11
    result := TPCANBasic.<u>Initialize(TPCANBasic.PCA</u>
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'PCAN-PCI (Ch-2) was initiali
    // All initialized channels are released
    //
    TPCANBasic.Uninitialize(TPCANBasic.PCAN NONEBU
end;
```

Python:

```
# The Plug & Play Channel (PCAN-PCI) is initialize
#
objPCAN = PCANBasic()
result = objPCAN.Initialize(PCAN_PCIBUS2, PCAN_BAU
if result != PCAN_ERROR_OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.GetErrorText(result)
    print result[1]
else:
    print "PCAN-PCI (Ch-2) was initialized"
# All initialized channels are released
```

#
objPCAN.<u>Uninitialize</u>(PCAN_NONEBUS)

■ See Also

Uninitialize

<u>GetValue</u>

Understanding PCAN-Basic

Plain function Version: <u>CAN_Initialize</u>

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

Initialize(TPCANHandle, TPCANBaudrate, TPCANType, UInt32, UInt16)

Initializes a PCAN Channel which represents a **Not** Plug & Play PCAN-Device.

Syntax

```
■ Pascal OO
class function Initialize(
    Channel: TPCANHandle;
    Btr0Btr1: TPCANBaudrate;
    HwType: TPCANType;
    I0Port: LongWord;
    Interrupt: Word
    ): TPCANStatus; overload;
```

□ C#

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN_;
public static extern TPCANStatus Initialize(
    [MarshalAs(UnmanagedType.U1)]
    TPCANHandle Channel,
    [MarshalAs(UnmanagedType.U2)]
    TPCANBaudrate Btr0Btr1,
    [MarshalAs(UnmanagedType.U1)]
    TPCANType HwType,
    UInt32 IOPort,
    UInt16 Interrupt);
```

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN_:
static TPCANStatus Initialize(
    [MarshalAs(UnmanagedType::U1)]
```

< ^ >

```
TPCANHandle Channel,
    [MarshalAs(UnmanagedType::U2)]
    TPCANBaudrate Btr0Btr1,
    [MarshalAs(UnmanagedType::U1)]
    TPCANType HwType,
    UInt32 IOPort,
    UInt16 Interrupt);
Visual Basic
<DllImport("PCANBasic.dll", EntryPoint:="CAN Ir</pre>
Public Shared Function Initialize( _
    <MarshalAs(UnmanagedType.U1)>
    ByVal Channel As <u>TPCANHandle</u>, ___
    <MarshalAs(UnmanagedType.U2)> _
    ByVal Btr0Btr1 As TPCANBaudrate,
   <MarshalAs(UnmanagedType.U1)> _
   ByVal HwType As <u>TPCANType</u>, ____
   ByVal IOPort As UInt32, _
   ByVal Interrupt As UInt16) As TPCANStatus
End Function
Python
def <u>Initialize</u>(
    self,
    Channel,
```

```
Btr0Btr1,
HwType = <u>TPCANType(0)</u>,
IOPort = c_uint(0),
Interrupt = c_ushort(0))
```

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel

	(see <u>TPCANHandle</u>).
Btr0Btr1	The speed for the communication (BTR0BTR1 code).
НwТуре	The type of hardware and operation mode (see <u>TPCANMode</u>).
IOPort	The I/O address for the parallel port.
Interrupt	Interrupt number of the parallel port.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_CAUTION:	Indicates that the channel has been initialized but at a different bit rate as the given one.
PCAN_ERROR_ILLHANDLE:	Indicates that the desired PCAN Channel is not valid. Check the list of <u>valid</u> <u>Channels</u> .
PCAN_ERROR_ILLHW:	Indicates that the desired PCAN Channel is not available.
PCAN_ERROR_ILLOPERATION:	Indicates that an action cannot be executed due to the state of the hardware. Possible causes are:

	• The desired PCAN- Channel is a LAN Channel, which uses a different bit rate than the specified.
PCAN_ERROR_REGTEST:	Indicates a problem with hardware registration, normally due to wrong values in the parameters ' <i>HwType</i> ', ' <i>IOPort</i> ' and ' <i>Interrupt</i> '.
PCAN_ERROR_INITIALIZE:	Indicates that the desired PCAN channel cannot be connected because it is already in use (PCAN-Basic / PCAN-Light environment).
PCAN_ERROR_NETINUSE:	Indicates that the desired PCAN-Channel is being used with a different bit rate (PCAN-View).
PCAN_ERROR_HWINUSE:	Indicates that the desired PCAN-Channel is being used (CanApi2 connection).
PCAN_ERROR_NODRIVER:	The driver needed for connecting the desired PCAN Channel is not loaded.

Remarks

As indicated by its name, the <u>Initialize</u> method initiates a PCAN 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 PCAN_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCAN Channel means:

- to reserve the Channel for the calling application/process.
- to allocate channel resources, like receive and transmit queues.
- to register/connect the Hardware denoted by the channel handle.
- to check and adapt the bus speed, if the Channel is already in use. (Only if the Channel was pre-configured as Bitrate Adapting; see: Bitrate-Adapting Parameter).
- to set the channel in Listen-Only mode. (Only *if the channel* was pre-configured as Listen-Only; see: Listen-Only Parameter).
- to configure the filter to catch all messages being transmitted in the bus.
- to set-up the default values of the different parameters (See <u>GetValue</u>).
- to set the Receive Status of the channel. (*Pre-configured* value; see: <u>Receive Status Parameter</u>).

Different than the PCAN-Light API, the Initialization process will fail if an application try to initialize a PCAN-Channel that has been initialized already within the same process.

Take in consideration that initializing a channel causes a reset of the CAN hardware , when the bus status is other than OK. In this way errors like BUSOFF, BUSWARNING, and BUSPASSIVE, are removed.

PCAN-LAN Channels

A PCAN-LAN channel doesn't allow changing the bit rate using PCAN-Basic. In order to connect a PCAN-LAN Channel it is necessary to know the bit rate of the PCAN-Gateway device that is represented by that channel. If the bit rate is not known, the parameter <u>Bitrate-Adapting</u> should be used.

Python Notes

• Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.

- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.
- Plug-&-Play and No-Plug-&-Play hardware: In order to initialize a channel which represents a Plug-&-Play PCAN device, only the Channel-handle and bit rate parameters are needed. The other parameters will be assigned their default values. For No-Plug-&-Play devices, all parameters have to be entered.

Example

The following example shows the initialize and uninitialize processes for a Not-Plug-And-Play channel (channel 1 of the PCAN-DNG). In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
// The Not Plug & Play Channel (PCAN-DNG) is initi
11
result = PCANBasic.<u>Initialize(PCANBasic.PCAN_DNGBU</u>
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    strMsg = new StringBuilder(256);
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("PCAN-DNG (Ch-1) was initializ
// All initialized channels are released
```

//
PCANBasic.Uninitialize(PCANBasic.PCAN_NONEBUS);

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
// The Not Plug & Play Channel (PCAN-DNG) is initi
11
result = PCANBasic::<u>Initialize(PCANBasic::PCAN_DNG</u>
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    strMsg = gcnew StringBuilder(256);
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("PCAN-DNG (Ch-1) was initiali
// All initialized channels are released
11
PCANBasic::Uninitialize(PCANBasic::PCAN NONEBUS);
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
' The Not Plug & Play Channel (PCAN-DNG) is initia
' result = PCANBasic.<u>Initialize(PCANBasic.PCAN_DNGBU</u>
If result <> TPCANStatus.PCAN_ERROR_OK Then
' An error occurred, get a text describing the
' strMsg = New StringBuilder(256)
```

```
PCANBasic.GetErrorText(result, 0, strMsg)
MessageBox.Show(strMsg.ToString)
Else
MessageBox.Show("PCAN-DNG (Ch-1) was initializ
End If
' All initialized channels are released
'
PCANBasic.Uninitialize(PCANBasic.PCAN_NONEBUS)
```

Pascal OO:

```
var
result : <u>TPCANStatus;</u>
 strMsg: array [0..256] of Char;
begin
    // The Not Plug & Play Channel (PCAN-DNG) is i
    11
    result := TPCANBasic.Initialize(TPCANBasic.PCA
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'PCAN-DNG (Ch-1) was initiali
    // All initialized channels are released
    //
    TPCANBasic.Uninitialize(TPCANBasic.PCAN_NONEBU
end;
```

Python:

```
# The Not Plug & Play Channel (PCAN-DNG) is initia
#
```

```
objPCAN = PCANBasic()
result = objPCAN.Initialize(PCAN_DNGBUS1, PCAN_BAU
if result != PCAN_ERROR_OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.GetErrorText(result)
    print result[1]
else:
    print "PCAN-DNG (Ch-1) was initialized"
# All initialized channels are released
#
objPCAN.Uninitialize(PCAN_NONEBUS)
```

■ See Also

<u>Uninitialize</u>

GetValue

Understanding PCAN-Basic

Plain function Version: <u>CAN_Initialize</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation PCAN-Basic Documentation

Home

< ^ >

InitializeFD

Initializes a FD capable PCAN Channel.

Syntax

Pascal OO

class function InitializeFD(
 Channel: TPCANHandle;
 BitrateFD: TPCANBitrateFD
): TPCANStatus;

□ C#

□ C++ / CLR

[DllImport("PCANBasic.dll", EntryPoint = "CAN :
static TPCANStatus InitializeFD(
 [MarshalAs(UnmanagedType::U1)]
 TPCANHandle Channel,
 TPCANBitrateFD BitrateFD);

B Visual Basic

⊨ Pyt	hon
def	InitializeFD(self, Channel, BitrateFD)

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).
BitrateFD	The speed for the communication (<u>FD Bitrate</u> string).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_CAUTION:	Indicates that the channel has been initialized but at a different bit rate as the given one.
PCAN_ERROR_ILLHANDLE:	Indicates that the desired PCAN Channel is not valid. Check the list of <u>valid</u> <u>Channels</u> .
PCAN_ERROR_ILLHW:	Indicates that the desired PCAN Channel is not available.
PCAN_ERROR_ILLOPERATION:	Indicates that an action

	 cannot be executed due to the state of the hardware. Possible causes are: The desired PCAN Channel is not FD capable and cannot be initialized using this method. The desired PCAN-Channel is a LAN Channel is a LAN Channel, which uses a different bit rate than the specified.
PCAN_ERROR_INITIALIZE:	Indicates that the desired PCAN Channel cannot be connected because it is already in use (PCAN-Basic / PCAN-Light environment).
PCAN_ERROR_NETINUSE:	Indicates that the desired PCAN-Channel is being used with a different bit rate (PCAN-View).
PCAN_ERROR_HWINUSE:	Indicates that the desired PCAN-Channel is being used (CanApi connection).
PCAN_ERROR_NODRIVER:	The driver needed for connecting the desired PCAN Channel is not loaded.

🗉 Remarks

Note on correspondence of methods:

A Channel that is initialized using InitializeFD must use <u>ReadFD</u> and <u>WriteFD</u> for communication. Calling <u>Read</u> and/or <u>Write</u> will result in a **PCAN_ERROR_ILLOPERATION** error.

As indicated by its name, the InitializeFD method initiates a FD capable PCAN Channel, preparing it for communicate within the CAN bus connected to it. Calls to the API methods will fail if they are used with a Channel handle, different than PCAN_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCAN Channel means:

- to reserve the Channel for the calling application/process.
- to allocate channel resources, like receive and transmit queues.
- to register/connect the Hardware denoted by the channel handle.
- to check and adapt the bus speed, if the Channel is already in use. (Only if the Channel was pre-configured as Bitrate Adapting; see: Bitrate-Adapting Parameter).
- to set the channel in Listen-Only mode. (Only *if the channel* was pre-configured as Listen-Only; see: Listen-Only Parameter).
- to open the messages filter for the application/process.
- to set-up the default values of the different parameters (See <u>GetValue</u>).
- to set the Receive Status of the channel. (*Pre-configured value; see:* <u>Receive Status Parameter</u>).

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

Take in consideration that initializing a channel causes a reset of the CAN hardware , when the bus status is other than OK. In this way errors like BUSOFF, BUSWARNING, and BUSPASSIVE, are removed.

PCAN-LAN Channels

A PCAN-LAN channel doesn't allow changing the bit rate using PCAN-Basic. In order to connect a PCAN-LAN Channel it is

necessary to know the bit rate of the PCAN-Gateway device that is represented by that channel. If the bit rate is not known, the parameter <u>Bitrate-Adapting</u> should be used.

Python Notes

- Class-Method: Different than the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the initialize and uninitialize processes for a FD capable channel (channel 1 of a PCAN-USB Pro FD hardware). In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

C#:

```
string bitrate;
<u>TPCANStatus</u> result;
StringBuilder strMsg;
// Defines a FD Bit rate string with nominal and d
//
bitrate = "f_clock_mhz=24, nom_brp=1, nom_tseg1=17
// The FD capable Channel (PCAN-USB Pro FD) is ini
//
result = PCANBasic.InitializeFD(PCANBasic.PCAN_USB
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
```

```
strMsg = new StringBuilder(256);
PCANBasic.GetErrorText(result, 0, strMsg);
MessageBox.Show(strMsg.ToString());
}
else
MessageBox.Show("PCAN-USB Pro FD (Ch-1) was in
// All initialized channels are released
//
PCANBasic.Uninitialize(PCANBasic.PCAN_NONEBUS);
```

C++/CLR:

```
String^ bitrate;
TPCANStatus result;
StringBuilder^ strMsg;
// Defines a FD Bit rate string with nominal and d
11
bitrate = "f_clock_mhz=24, nom_brp=1, nom_tseg1=17
// The FD capable Channel (PCAN-USB Pro FD) is ini
11
result = PCANBasic::InitializeFD(PCANBasic::PCAN_U
if (result != TPCANStatus::PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    11
    strMsg = gcnew StringBuilder(256);
    PCANBasic::<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("PCAN-USB Pro FD (Ch-1) was i
// All initialized channels are released
11
PCANBasic::Uninitialize(PCANBasic::PCAN_NONEBUS);
```

Visual Basic:

```
Dim bitrate As String
Dim result As TPCANStatus
Dim strMsg As StringBuilder
' Defines a FD Bit rate string with nominal and da
bitrate = "f_clock_mhz=24, nom_brp=1, nom_tseg1=17
' The FD capable Channel (PCAN-USB Pro FD) is init
result = PCANBasic.InitializeFD(PCANBasic.PCAN_USE
If result <> TPCANStatus.PCAN ERROR OK Then
    ' An error occurred, get a text describing the
    strMsg = New StringBuilder(256)
    PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
    MessageBox.Show(strMsg.ToString)
Else
    MessageBox.Show("PCAN-USB Pro FD (Ch-1) was in
End If
' All initialized channels are released
PCANBasic.Uninitialize(PCANBasic.PCAN_NONEBUS)
```

Pascal OO:

```
var
bitrate: String;
result : <u>TPCANStatus;</u>
strMsg: array [0..256] of Char;
begin
    // Defines a FD Bit rate string with nominal a
    //
    bitrate := 'f_clock_mhz=24, nom_brp=1, nom_tse
```

```
// The FD capable Channel (PCAN-USB Pro FD) is
    11
    result := TPCANBasic.InitializeFD(TPCANBasic.F
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'PCAN-USB Pro FD (Ch-1) was i
    // All initialized channels are released
    11
    TPCANBasic.Uninitialize(TPCANBasic.PCAN_NONEBU
end;
```

Python:

```
# Defines a FD Bit rate string with nominal and da
#
bitrate = "f_clock_mhz=24, nom_brp=1, nom_tseg1=17
# The FD capable Channel (PCAN-USB Pro FD) is init
#
objPCAN = PCANBasic()
result = objPCAN.InitializeFD(PCAN_USBBUS1, bitrat
if result != PCAN_ERROR_OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.GetErrorText(result)
    print result[1]
else:
    print "PCAN-USB Pro FD (Ch-1) was initialized"
# All initialized channels are released
```

#
objPCAN.<u>Uninitialize(PCAN_NONEBUS)</u>

■ See Also

Uninitialize

<u>ReadFD</u>

<u>WriteFD</u>

Plain function Version: <u>CAN_InitializeFD</u>

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send $\underline{\text{feedback}}$ to this documentation

PCAN-Basic Documentation

Home

< ^ >

Uninitialize

Uninitializes a PCAN Channel.

Syntax

Bernard Pascal OO

```
class function Uninitialize(
    Channel: <u>TPCANHandle</u>
    ): <u>TPCANStatus;</u>
```

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_L public static extern TPCANStatus Uninitialize([MarshalAs(UnmanagedType.U1)] TPCANHandle Channel);

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN L
static TPCANStatus Uninitialize(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel);
```

Visual Basic

```
Bereich Python
```

```
def Uninitialize(
    self,
```

Channel)

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN
	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 this possibilities:

Single-Release: Given a handle of a PCAN Channel initialized before with the method <u>Initialize</u>. If the given channel can not be found then an error is returned.

Multiple-Release: Giving the handle value PCAN_NONEBUS which instructs the API to search for all channels initialized by the calling application and release them all. This option cause no errors if no hardware were uninitialized.

Transmit-queue at uninitialize: When a PCAN-Basic channel connection is terminated, the underlying hardware's transmit-queue will not immediately be discarded. PCAN-Basic will wait some time before finalizing, so that the hardware has time to send (or try to send) those unsent messages. When the time is up (amount 500 milliseconds), the rest of the messages in the queue (if any) are

discarded.

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the initialize and uninitialize (Single-Release) processes for the PCAN_PCIBUS1 channel. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: To see an example of Multiple-Release, see the <u>Initialize</u> method.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
strMsg = new StringBuilder(256);
// The Plug & Play Channel (PCAN-PCI) is initializ
//
result = PCANBasic.Initialize(PCANBasic.PCAN_PCIBU
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic.GetErrorText(result, 0, strMsg);
    MessageBox.Show(strMsg.ToString());
}
```

```
else
    MessageBox.Show("PCAN-PCI (Ch-1) was initializ
....
// The PCI Channel is released
//
result = PCANBasic.Uninitialize(PCANBasic.PCAN_PCI
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("PCAN-PCI (Ch-1) was released"
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
strMsg = gcnew StringBuilder(256);
// The Plug & Play Channel (PCAN-PCI) is initializ
//
result = PCANBasic::Initialize(PCANBasic::PCAN_PCI
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("PCAN-PCI (Ch-1) was initiali
```

```
// The PCI Channel is released
//
result = PCANBasic::Uninitialize(PCANBasic::PCAN_P
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("PCAN-PCI (Ch-1) was released
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
strMsg = New StringBuilder(256)
' The Plug & Play Channel (PCAN-PCI) is initialize
' result = PCANBasic.<u>Initialize(PCANBasic.PCAN_PCIBU</u>
If result <> TPCANStatus.PCAN_ERROR_OK Then
        ' An error occurred, get a text describing the
        ' An error occurred, get a text describing the
        ' PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox.Show(strMsg.ToString)
Else
        MessageBox.Show("PCAN-PCI (Ch-1) was initializ
End If
.....
' The PCI Channel is released
```

```
result = PCANBasic.Uninitialize(PCANBasic.PCAN_PCI
If result <> TPCANStatus.PCAN_ERROR_OK Then
        ' An error occurred, get a text describing the
        '
        PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox.Show(strMsg.ToString)
Else
        MessageBox.Show("PCAN-PCI (Ch-1) was released"
End If
```

Pascal OO:

```
var
 result : <u>TPCANStatus;</u>
 strMsg: array [0..256] of Char;
begin
    // The Plug & Play Channel (PCAN-PCI) is initi
    11
    result := TPCANBasic.Initialize(TPCANBasic.PCA
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'PCAN-PCI (Ch-1) was initiali
    // The PCI Channel is released
    11
    result := TPCANBasic.Uninitialize(TPCANBasic.P
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
```

```
//
TPCANBasic.GetErrorText(result, 0, strMsg)
MessageBox(0, strMsg, 'Error',MB_OK);
end
else
MessageBox(0,'PCAN-PCI (Ch-1) was released
end;
```

Python:

```
# The Plug & Play Channel (PCAN-PCI) is initialize
#
objPCAN = PCANBasic()
result = objPCAN.<u>Initialize(PCAN_PCIBUS1, PCAN_BAU</u>
if result != PCAN ERROR OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.<u>GetErrorText(result)</u>
    print result[1]
else:
    print "PCAN-PCI (Ch-1) was initialized"
. . . .
# The PCI Channel is released
#
result = objPCAN.Uninitialize(PCAN PCIBUS1)
if result != PCAN ERROR OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.<u>GetErrorText(result)</u>
    print result[1]
else:
    print "PCAN-PCI (Ch-1) was released"
```

■ See Also

Initialize

Plain function Version: <u>CAN_Uninitialize</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\text{feedback}}$ to this documentation

PCAN-Basic Documentation

🔒 Home

< ^ >

Reset

Resets the receive and transmit queues of a PCAN Channel.

Syntax

Pascal OO
 class function Reset(
 Channel: <u>TPCANHandle</u>
): <u>TPCANStatus;
 }
}
</u>

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_F public static extern TPCANStatus Reset([MarshalAs(UnmanagedType.U1)] TPCANHandle Channel);

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN F
static TPCANStatus Reset(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel);
```

Visual Basic

Bereich Python

```
def Reset(
    self,
```

Channel)

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN
	channel was not found in the list of
	initialized channels of the calling
	application.

🗉 Remarks

Calling this method ONLY clear the queues of a Channel. A reset of the CAN controller doesn't take place.

Normally a reset of the CAN Controller is desired when a bus-off occur. In this case an application cannot use the channel to communicate anymore, until the CAN controller is reset. Consider using the PCAN-Basic parameter <u>PCAN_BUSOFF_AUTORESET</u> 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 <u>uninitialize</u> and <u>initialize</u> again the channel used. This causes a hardware reset, but only when no more clients are connected to that channel.

Python Notes

• Class-Method: Unlike the .NET Framework, under Python a

variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.

• Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method Reset on the channel PCAN_PCIBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
TPCANStatus result;
StringBuilder strMsg;
strMsg = new StringBuilder(256);
. . . . . .
// The PCI Channel is reset
//
result = PCANBasic.Reset(PCANBasic.PCAN PCIBUS1);
if (result != TPCANStatus.PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("PCAN-PCI (Ch-1) was reset");
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
strMsg = gcnew StringBuilder(256);
. . . . . .
// The PCI Channel is reset
11
result = PCANBasic::Reset(PCANBasic::PCAN PCIBUS1)
if (result != TPCANStatus::PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("PCAN-PCI (Ch-1) was reset");
```

Visual Basic:

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

Pascal OO:

```
var
 result : <u>TPCANStatus;</u>
 strMsg: array [0..256] of Char;
begin
. . . . . .
    // The PCI Channel is reset
    11
    result := TPCANBasic.Reset(TPCANBasic.PCAN_PCI
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'PCAN-PCI (Ch-1) was reset', '
```

Python:

```
# The PCI Channel is released
#
result = objPCAN.Reset(PCAN_PCIBUS1)
if result != PCAN_ERROR_OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.GetErrorText(result)
```

```
print result[1]
else:
    print "PCAN-PCI (Ch-1) was reset"
```

■ See Also

Read

<u>Write</u>

<u>SetValue</u>

Plain function Version: <u>CAN_Reset</u>

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation
PCAN-Basic Documentation

Home

< ^ >

GetStatus

Gets the current BUS status of a PCAN Channel.

Syntax

Pascal OO

class function GetStatus(
 Channel: <u>TPCANHandle</u>
): <u>TPCANStatus;</u>

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_(
public static extern TPCANStatus GetStatus(
 [MarshalAs(UnmanagedType.U1)]
 TPCANHandle Channel);

□ C++ / CLR

[DllImport("PCANBasic.dll", EntryPoint = "CAN (
static TPCANStatus GetStatus(
 [MarshalAs(UnmanagedType::U1)]
 TPCANHandle Channel);

Visual Basic

Bereich Python

def GetStatus(
 self,

Channel)

B Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a <u>TPCANStatus</u> code. The typical return values are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN Channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSLIGHT:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-light status.
PCAN_ERROR_BUSHEAVY:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-heavy status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_OK:	Indicates that the status of the given PCAN Channel is OK.

🗉 Remarks

When the hardware status is bus-off, an application cannot communicate anymore. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> 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 <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method GetStatus on the channel PCAN_PCIBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
TPCANStatus result;
StringBuilder strMsg;
strMsg = new StringBuilder(256);
.....
// Check the status of the PCI Channel
//
result = PCANBasic.GetStatus(PCANBasic.PCAN_PCIBUS
switch (result)
```



C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
strMsg = gcnew StringBuilder(256);
.....
// Check the status of the PCI Channel
//
result = PCANBasic::GetStatus(PCANBasic::PCAN_PCIE
switch (result)
{
    case TPCANStatus::PCAN_ERROR_BUSLIGHT:
        MessageBox::Show("PCAN-PCI (Ch-1): Handlin
        break;
```



Visual Basic:

```
Dim result As TPCANStatus
Dim strMsg As StringBuilder
strMsg = New StringBuilder(256)
.....
' Check the status of the PCI Channel
'
result = PCANBasic.GetStatus(PCANBasic.PCAN_PCIBUS
Select Case result
    Case TPCANStatus.PCAN_ERROR_BUSLIGHT
    MessageBox.Show("PCAN-PCI (Ch-1): Handling
    Case TPCANStatus.PCAN_ERROR_BUSHEAVY
    MessageBox.Show("PCAN-PCI (Ch-1): Handling
    Case TPCANStatus.PCAN_ERROR_BUSHEAVY
    MessageBox.Show("PCAN-PCI (Ch-1): Handling
    Case TPCANStatus.PCAN_ERROR_BUSOFF
    MessageBox.Show("PCAN-PCI (Ch-1): Handling
    Case TPCANStatus.PCAN_ERROR_OK
    MessageBox.Show("PCAN-PCI (Ch-1): Status i
```

```
Case Else

' An error occurred, get a text describing

'

PCANBasic.<u>GetErrorText</u>(result, 0, strMsg)

MessageBox.Show(strMsg.ToString)

End Select
```

Pascal OO:

```
var
 result : <u>TPCANStatus;</u>
 strMsg: array [0..256] of Char;
begin
. . . . . .
// Check the status of the PCI Channel
11
result := TPCANBasic.GetStatus(TPCANBasic.PCAN_PCI
case result of
    PCAN ERROR BUSLIGHT:
        MessageBox(0, 'PCAN-PCI (Ch-1): Handling a
    PCAN ERROR BUSHEAVY:
        MessageBox(0, 'PCAN-PCI (Ch-1): Handling a
    PCAN_ERROR_BUSOFF:
        MessageBox(0, 'PCAN-PCI (Ch-1): Handling a
    PCAN ERROR OK:
        MessageBox(0, 'PCAN-PCI (Ch-1): Status is C
    else
    begin
        // An error occurred, get a text describin
        //
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end;
end;
```

Python:

```
. . . . . .
# Check the status of the PCI Channel
#
result = objPCAN.GetStatus(PCAN_PCIBUS1)
if result == PCAN ERROR BUSLIGHT:
    print "PCAN-PCI (Ch-1): Handling a BUS-LIGHT s
elif result == PCAN_ERROR_BUSHEAVY:
    print "PCAN-PCI (Ch-1): Handling a BUS-HEAVY s
elif result == PCAN_ERROR_BUSOFF:
    print "PCAN-PCI (Ch-1): Handling a BUS-OFF sta
elif result == PCAN_ERROR_OK:
    print "PCAN-PCI (Ch-1): Status is OK"
else:
    # An error occurred, get a text describing the
    #
    result = objPCAN.GetErrorText(result)
    print result[1]
```

■ See Also

Parameter Value Definitions

TPCANParameter

Plain function Version: CAN_GetStatus

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

PCAN-Basic Documentation	A Home	< ^ >
Read		

Reads a CAN message from the receive queue of a PCAN Channel.

Overloads

	Function	Description
	<u>Read(TPCANHandle,</u> <u>TPCANMsg)</u>	Reads a CAN message from the receive queue.
	<u>Read(TPCANHandle,</u> <u>TPCANMsg,</u> <u>TPCANTimestamp)</u>	Reads a CAN message and its time stamp from the receive queue.
= \$	Read(TPCANHandle)	Reads a CAN message and its time stamp from the receive queue.

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send $\underline{\text{feedback}}$ to this documentation

PCAN-Basic Documentation

🕇 Home

<^>

Read(TPCANHandle, TPCANMsg)

Reads a CAN message from the receive queue of a PCAN Channel.

Syntax

class function <u>Read(</u> Channel: <u>TPCANHandle;</u> var MessageBuffer: <u>TPCANMsg</u>): <u>TPCANStatus</u> ; overload;
⊟ C#
<pre>public static <u>TPCANStatus</u> <u>Read(TPCANHandle</u> Channel, out <u>TPCANMsg</u> MessageBuffer);</pre>
B C++ / CLR
<pre>static TPCANStatus Read(TPCANHandle Channel, TPCANMsg %MessageBuffer);</pre>
∃ Visual Basic
Public Shared Function <u>Read(</u> ByVal Channel As <u>TPCANHandle</u> , _ ByRef MessageBuffer As <u>TPCANMsg</u>) As <u>TPCANS</u> End Function

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

MessageBuffer	A <u>TPCANMsg</u> buffer to store the
	CAN message.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsg</u> structure.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSLIGHT:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-light status.
PCAN_ERROR_BUSHEAVY:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-heavy status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

∃ Remarks

The <u>Read</u> method returns received messages or status messages from the receive queue. It is important to call <u>Read</u> repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See <u>Receive Status Parameter</u> for more information.

The receive queue can contain up to **32767** messages.

If the time when the message was received is needed, use the overloaded <u>Read</u> method.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the <u>Read</u> method. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the <u>Read</u> method in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a method directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSHEAVY, and BUSLIGTH, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset, but only when no more clients are connected to that channel.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, RTR, Error, or Status message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and LEN fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of <u>Read</u> is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error	E C
00h	00h	00h	02h	PCAN_ERROR_OVERRUN	0
00h	00h	00h	04h	PCAN_ERROR_BUSLIGHT	0
00h	00h	00h	08h	PCAN_ERROR_BUSHEAVY	0

00h	00h	00h	10h	PCAN_ERROR_BUSOFF	0

Example

The following example shows the use of method <u>Read</u> on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized and that the following code is an OnTimer event handler method.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
TPCANMsq msg;
strMsg = new StringBuilder(256);
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic.<u>Read(PCANBasic.PCAN_USBBUS1</u>
    if (result != TPCANStatus.PCAN_ERROR_QRCVEMPTY
    {
        // Process the received message
        11
        MessageBox.Show("A message was received");
        ProcessMessage(msg);
    }
    else
```

```
{
    // An error occurred, get a text describin
    //
    PCANBasic.GetErrorText(result, 0, strMsg);
    MessageBox.Show(strMsg.ToString());
    // Here can be decided if the loop has to
    // status is bus-off)
    //
    HandleReadError(result);
  }
// Try to read a message from the receive queue of
// until the queue is empty
//
}while((result & TPCANStatus.PCAN_ERROR_QRCVEMPTY)
```

C++CLR:

```
<u>TPCANStatus</u> result;
StringBuilder^ strMsg;
TPCANMsq msq;
strMsg = gcnew StringBuilder(256);
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic::<u>Read(PCANBasic::PCAN_USBBU</u>
    if (result != TPCANStatus::PCAN_ERROR_QRCVEMPT
    {
        // Process the received message
        11
        MessageBox::Show("A message was received")
        ProcessMessage(msg);
    }
    else
    {
        // An error occurred, get a text describin
```

```
//
PCANBasic::GetErrorText(result, 0, strMsg)
MessageBox::Show(strMsg->ToString());
// Here can be decided if the loop has to
// status is bus-off)
//
HandleReadError(result);
}
// Try to read a message from the receive queue of
// until the queue is empty
//
}while((result & TPCANStatus::PCAN_ERROR_QRCVEMPTY
```

Visual Basic:

```
Dim result As TPCANStatus
Dim strMsg As StringBuilder
Dim msg As TPCANMsg
strMsg = New StringBuilder(256)
Do
    ' Check the receive queue for new messages
    result = PCANBasic.Read(PCANBasic.PCAN USBBUS1
    If result <> TPCANStatus.PCAN ERROR ORCVEMPTY
        MessageBox.Show("A message was received")
        ProcessMessage(msg)
    Else
        ' An error occurred, get a text describing
        PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox.Show(strMsg.ToString())
        ' Here can be decided if the loop has to b
        ' status is bus-off)
        HandleReadError(result)
    End If
```

```
' Try to read a message from the receive queue of
' until the queue is empty
'
Loop While ((result And TPCANStatus.PCAN_ERROR_QRC
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    msg: <u>TPCANMsg;</u>
begin
    repeat
        // Check the receive queue for new message
        11
        result := TPCANBasic.<u>Read(TPCANBasic.PCAN_</u>
        If (result <> PCAN_ERROR_QRCVEMPTY) Then
        begin
            // Process the received message
            11
            MessageBox(0, 'A message was received',
            ProcessMessage(msg);
        end
        else
        begin
            // An error occurred, get a text descr
            11
            TPCANBasic.GetErrorText(result, 0, str
            MessageBox(0, strMsg, 'Error', MB_OK);
            // Here can be decided if the loop has
            // status is bus-off)
            11
            HandleReadError(result);
        end:
    // Try to read a message from the receive queu
    // until the queue is empty
    11
    until ((<u>TPCANStatus</u>(Integer(result) AND Intege
```

∃ See Also

<u>Write</u>

Using Events

Error Frames

Plain function Version: CAN_Read

Copyright $\textcircled{\mbox{\scriptsize opt}}$ 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\mbox{feedback}}$ to this documentation

1 Home

くへと

Read(TPCANHandle, TPCANMsg, TPCANTimestamp)

Reads a CAN message and its time stamp from the receive queue of a PCAN Channel.

Syntax

B Pascal OO	
class function <u>Read(</u>	
Channel: <u>TPCANHandle</u> ;	
<pre>var MessageBuffer: <u>TPCANMsg;</u></pre>	
var TimestampBuffer: <u>TPCANTimestamp</u>	
): <u>TPCANStatus</u> ; overload;	

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_F public static extern TPCANStatus Read([MarshalAs(UnmanagedType.U1)] TPCANHandle Channel, out TPCANMsg MessageBuffer, ref TPCANTimestamp TimestampBuffer);

```
□ C++ / CLR
```

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN F
static TPCANStatus Read(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel,
    TPCANMsg %MessageBuffer,
    TPCANTimestamp %TimestampBuffer);
```

Visual Basic

<DllImport("PCANBasic.dll", EntryPoint:="CAN_Re</pre>

```
Public Shared Function Read( _
        <MarshalAs(UnmanagedType.U1)> _
        ByVal Channel As <u>TPCANHandle</u>, _
        ByRef MessageBuffer As <u>TPCANMsg</u>, _
        ByRef TimestampBuffer As <u>TPCANTimestamp</u>) As
End Function
```

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsg</u> buffer to store the CAN message.
TimestampBuffer	A <u>TPCANTimestamp</u> buffer to get the reception time of the message.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

	rs he int
PCAN_ERROR_INITIALIZE: Indicates that the given PCA channel was not found in the list of initialized channels of the calling application.	λN e

PCAN_ERROR_BUSLIGHT:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-light status.
PCAN_ERROR_BUSHEAVY:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-heavy status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

🗉 Remarks

The <u>Read</u> method returns received messages or status messages from the receive queue. It is important to call <u>Read</u> repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See <u>Receive Status Parameter</u> for more information.

The receive queue can contain up to **32767** messages.

If the time when the message was received is not needed, use the overloaded <u>Read</u> method.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the <u>Read</u> method. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the <u>Read</u> method in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a method directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSHEAVY, and BUSLIGTH, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset, but only when no more clients are connected to that channel.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, RTR, Error, or Status message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and LEN fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of <u>Read</u> is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error	E C
00h	00h	00h	02h	PCAN_ERROR_OVERRUN	0

00h	00h	00h	04h	PCAN_ERROR_BUSLIGHT	0
00h	00h	00h	08h	PCAN_ERROR_BUSHEAVY	0
00h	00h	00h	10h	PCAN_ERROR_BUSOFF	0

Example

The following example shows the use of method <u>Read</u> on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized and that the following code is an OnTimer event handler method.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
TPCANMsg msg;
```

```
TPCANTimestamp time;
strMsg = new StringBuilder(256);
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic.<u>Read(PCANBasic.PCAN_USBBUS1</u>
    if (result != TPCANStatus.PCAN_ERROR_QRCVEMPTY
    {
        // Process the received message
        11
        MessageBox.Show("A message was received");
        ProcessMessage(msg, time);
    }
    else
    {
        // An error occurred, get a text describin
        11
        PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
        MessageBox.Show(strMsg.ToString());
        // Here can be decided if the loop has to
        // status is bus-off)
        11
        HandleReadError(result);
    }
// Try to read a message from the receive queue of
// until the queue is empty
11
}while((result & TPCANStatus.PCAN_ERROR_QRCVEMPTY)
```

C++CLR:

```
<u>TPCANStatus</u> result;
StringBuilder^ strMsg;
<u>TPCANMsg</u> msg;
<u>TPCANTimestamp</u> time;
```

```
strMsg = gcnew StringBuilder(256);
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic::<u>Read(PCANBasic::PCAN_USBBU</u>
    if (result != TPCANStatus::PCAN_ERROR_QRCVEMPT
    {
        // Process the received message
        11
        MessageBox::Show("A message was received")
        ProcessMessage(msg, time);
    }
    else
    {
        // An error occurred, get a text describin
        11
        PCANBasic::GetErrorText(result, 0, strMsg)
        MessageBox::Show(strMsg->ToString());
        // Here can be decided if the loop has to
        // status is bus-off)
        11
        HandleReadError(result);
    }
// Try to read a message from the receive queue of
// until the queue is empty
11
}while((result & TPCANStatus::PCAN_ERROR_QRCVEMPTY
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim msg As <u>TPCANMsg</u>
Dim time As <u>TPCANTimestamp</u>
```

```
strMsg = New StringBuilder(256)
Do
    ' Check the receive queue for new messages
    result = PCANBasic.<u>Read(PCANBasic.PCAN_USBBUS1</u>
    If result <> TPCANStatus.PCAN_ERROR_QRCVEMPTY
        MessageBox.Show("A message was received")
        ProcessMessage(msg, time)
    Else
        ' An error occurred, get a text describing
        PCANBasic.<u>GetErrorText</u>(result, 0, strMsg)
        MessageBox.Show(strMsg.ToString())
        ' Here can be decided if the loop has to b
          status is bus-off)
        1.1
        HandleReadError(result);
    End If
' Try to read a message from the receive queue of
' until the queue is empty
Loop While ((result And TPCANStatus.PCAN_ERROR_QRC
```

Pascal OO:

```
var
    result : TPCANStatus;
    strMsg: array [0..256] of Char;
    msg: TPCANMsg;
    time: TPCANTimestamp;
begin
    repeat
    // Check the receive queue for new message
    //
    result := TPCANBasic.Read(TPCANBasic.PCAN_
    If (result <> PCAN_ERROR_QRCVEMPTY) Then
    begin
```

```
// Process the received message
        11
        MessageBox(0, 'A message was received',
        ProcessMessage(msg, time);
    end
    else
    begin
        // An error occurred, get a text descr
        11
        TPCANBasic.GetErrorText(result, 0, str
        MessageBox(0, strMsg, 'Error', MB_OK);
        // Here can be decided if the loop has
        // status is bus-off)
        11
        HandleReadError(result);
    end;
// Try to read a message from the receive queu
// until the queue is empty
11
until ((<u>TPCANStatus</u>(Integer(result) AND Intege
```

See Also

<u>Write</u>

Using Events

Error Frames

Plain function Version: CAN_Read

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

PCAN-Basic Documentation

Home

< ^ >

Read(TPCANHandle)

Reads a CAN message and its time stamp from the receive queue of a PCAN Channel.

Syntax

⊟ Pyt ł	างท
def	<u>Read(</u> self, Channel)

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a 3-touple. The order of the returned values is as follow:

[0]: The method's return value as a <u>TPCANStatus</u> code.

PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsg</u> structure.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the

	list of initialized channels of the calling application.
PCAN_ERROR_BUSLIGHT:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-light status.
PCAN_ERROR_BUSHEAVY:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-heavy status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

[1]: A <u>TPCANMsg</u> structure with the CAN message read.

[2]: A <u>TPCANTimestamp</u> structure with the time when a message was read.

🗉 Remarks

The Read method returns received messages or status messages from the receive queue. It is important to call Read repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See Receive Status Parameter for more information.

The receive queue can contain up to **32767** messages.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the <u>Read</u>

method. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the <u>Read</u> method in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a method directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSHEAVY, and BUSLIGTH, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset, but only when no more clients are connected to that channel.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, RTR, Error, or Status message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and LEN fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of <u>Read</u> is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error	E C
00h	00h	00h	02h	PCAN_ERROR_OVERRUN	0
00h	00h	00h	04h	PCAN_ERROR_BUSLIGHT	0
00h	00h	00h	08h	PCAN_ERROR_BUSHEAVY	0
00h	00h	00h	10h	PCAN_ERROR_BUSOFF	0

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of method <u>Read</u> on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized and that the following code is executed periodically.

Python:

```
readResult = PCAN ERROR OK,
while (readResult[0] & PCAN_ERROR_QRCVEMPTY) != PC
    # Check the receive queue for new messages
    #
    readResult = objPCAN.<u>Read(PCAN_USBBUS1)</u>
    if readResult[0] != PCAN_ERROR_QRCVEMPTY:
        # Process the received message
        #
        print "A message was received"
        ProcessMessage(result[1], result[2]) # Poss
    else:
        # An error occurred, get a text describing
        #
        result = objPCAN.<u>GetErrorText(readResult[0</u>
        print result[1]
        HandleReadError(readResult[0]) # Possible
```

■ See Also

<u>Write</u>

Using Events

Error Frames

Plain function Version: CAN_Read

Send feedback to this documentation

PCAN-Basic Documentation

🔒 Home

<^>

ReadFD

Reads a CAN message from the receive queue of a FD capable PCAN Channel.

Overloads

	Function	Description
	ReadFD(TPCANHandle, TPCANMsgFD)	Reads a CAN message from the receive queue .
= ♦ =	ReadFD(TPCANHandle, TPCANMsgFD, TPCANTimestampFD)	Reads a CAN message and its time stamp from the receive queue.
₩	ReadFD(TPCANHandle)	Reads a CAN message and its time stamp from the receive queue.

Copyright $\textcircled{\sc c}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

PCAN-Basic Documentation

🕇 Home

< ^ >

ReadFD(TPCANHandle, TPCANMsgFD)

Reads a CAN message from the receive queue of a FD capable PCAN Channel.

Syntax

■ Pascal OO
class function ReadFD(
 Channel: TPCANHandle;
 var MessageBuffer: TPCANMsgFD
):TPCANStatus; overload;

□ C#

public static <u>TPCANStatus</u> <u>ReadFD(</u>
 <u>TPCANHandle</u> Channel,
 out <u>TPCANMsgFD</u> MessageBuffer)

□ C++ / CLR

static <u>TPCANStatus</u> <u>ReadFD(</u>
 <u>TPCANHandle</u> Channel,
 <u>TPCANMsgFD</u> %MessageBuffer);

Visual Basic

Public Shared Function <u>ReadFD(</u> ByVal Channel As <u>TPCANHandle</u>, _ ByRef MessageBuffer As <u>TPCANMsgFD</u>) As <u>TPCAM</u> End Function

Parameters

Parameters	Description
Channel	The handle of a FD capable

	PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsgFD</u> buffer to store the CAN message.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsgFD</u> structure.
PCAN_ERROR_ILLOPERATION:	Indicates that the PCAN Channel passed to the method was not initialized using InitializeFD (plain function: CAN_InitializeFD).
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSWARNING:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- warning status.
PCAN_ERROR_BUSPASSIVE:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- passive status.

PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

Remarks

The use of <u>Read</u> and <u>ReadFD</u> are mutually exclusive. The PCAN Channel passed to this method **must be** initialized using <u>InitializeFD</u> (**plain function**: <u>CAN_InitializeFD</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

If the time when the message was received is needed, use the overloaded <u>ReadFD</u> method.

The <u>ReadFD</u> method returns received messages or status messages from the receive queue. It is important to call <u>ReadFD</u> repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See <u>Receive Status Parameter</u> for more information.

The receive queue can contain up to **32767** messages.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the <u>ReadFD</u> method. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the <u>ReadFD</u> method in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is
received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSWARNING, and BUSPASSIVE, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, FD, RTR, Error, or Status message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and DLC fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of <u>ReadFD</u> is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error
00h	00h	00h	02h	PCAN_ERROR_OVERRUN

00h	00h	00h	08h	PCAN_ERROR_BUSWARNING
00h	04h	00h	00h	PCAN_ERROR_BUSPASSIVE
00h	00h	00h	10h	PCAN_ERROR_BUSOFF

Example

The following example shows the use of method <u>ReadFD</u> on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized using the method <u>InitializeFD</u> and that the following code is an OnTimer event handler method.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
TPCANMsgFD msg;
strMsg = new StringBuilder(256);
```

```
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic.<u>ReadFD(PCANBasic.PCAN_USBBU</u>
    if (result != TPCANStatus.PCAN_ERROR_QRCVEMPTY
    {
        // Process the received message
        11
        MessageBox.Show("A message was received");
        ProcessMessage(msg);
    }
    else
    {
        // An error occurred, get a text describin
        11
        PCANBasic.GetErrorText(result, 0, strMsg);
        MessageBox.Show(strMsg.ToString());
        // Here can be decided if the loop has to
        // status is bus-off)
        11
        HandleReadError(result);
// Try to read a message from the receive queue of
// until the queue is empty
11
}while((result & TPCANStatus.PCAN_ERROR_QRCVEMPTY)
```

C++CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
TPCANMsgFD msg;
strMsg = gcnew StringBuilder(256);
do
```

```
{
    // Check the receive queue for new messages
    11
    result = PCANBasic::<u>ReadFD(PCANBasic::PCAN_USE</u>
    if (result != TPCANStatus::PCAN_ERROR_QRCVEMPT
    {
        // Process the received message
        11
        MessageBox::Show("A message was received")
        ProcessMessage(msg);
    }
    else
    {
        // An error occurred, get a text describin
        11
        PCANBasic::<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox::Show(strMsg->ToString());
        // Here can be decided if the loop has to
        // status is bus-off)
        11
        HandleReadError(result);
    }
// Try to read a message from the receive queue of
// until the queue is empty
11
}while((result & TPCANStatus::PCAN_ERROR_QRCVEMPTY
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim msg As <u>TPCANMsgFD</u>
strMsg = New StringBuilder(256)
Do
    ' Check the receive queue for new messages
    '
```

```
result = PCANBasic.<u>ReadFD(PCANBasic.PCAN_USBBU</u>
    If result <> TPCANStatus.PCAN_ERROR_QRCVEMPTY
          Process the received message
        MessageBox.Show("A message was received")
        ProcessMessage(msg)
    Else
        ' An error occurred, get a text describing
        PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox.Show(strMsg.ToString())
        ' Here can be decided if the loop has to b
        ' status is bus-off)
        HandleReadError(result)
    End If
 Try to read a message from the receive queue of
 until the queue is empty
Loop While ((result And TPCANStatus.PCAN ERROR QRC
```

Pascal OO:

```
end
    else
    begin
        // An error occurred, get a text descr
        11
        TPCANBasic.GetErrorText(result, 0, str
        MessageBox(0, strMsg, 'Error', MB_OK);
        // Here can be decided if the loop has
        // status is
                       bus-off)
        11
        HandleReadError(result);
    end;
// Try to read a message from the receive queu
// until the queue is empty
11
until ((<u>TPCANStatus</u>(Integer(result) AND Intege
```

See Also

<u>WriteFD</u>

Using Events

Error Frames

Plain function Version: <u>CAN_ReadFD</u>

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

< ^ >

ReadFD(TPCANHandle, TPCANMsgFD, TPCANTimestampFD)

Reads a CAN message and its time stamp from the receive queue of a FD capable PCAN Channel.

Syntax

Pascal OO
class function ReadFD(
 Channel: TPCANHandle;
 var MessageBuffer: TPCANMsgFD;
 var TimestampBuffer: TPCANTimestampFD
):TPCANStatus; overload;

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN F public static extern TPCANStatus ReadFD([MarshalAs(UnmanagedType.U1)] TPCANHandle Channel, out TPCANMsgFD MessageBuffer, out TPCANTimestampFD TimestampBuffer);

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN_F
static TPCANStatus ReadFD(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel,
    TPCANMsgFD %MessageBuffer,
    TPCANTimestampFD %TimestampBuffer);
```

Visual Basic

<DllImport("PCANBasic.dll", EntryPoint:="CAN_Re</pre>

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsgFD</u> buffer to store the CAN message.
TimestampBuffer	A <u>TPCANTimestampFD</u> buffer to get the reception time of the message.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsgFD</u> structure.
PCAN_ERROR_ILLOPERATION:	Indicates that the PCAN Channel passed to the method was not initialized

	using <u>InitializeFD</u> (plain function: <u>CAN_InitializeFD</u>).
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSWARNING:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- warning status.
PCAN_ERROR_BUSPASSIVE:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- passive status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

Remarks

The use of <u>Read</u> and <u>ReadFD</u> are mutually exclusive. The PCAN Channel passed to this method **must be** initialized using <u>InitializeFD</u> (**plain function**: <u>CAN_InitializeFD</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

If the time when the message was received is not needed, use the overloaded <u>ReadFD</u> method.

The <u>ReadFD</u> method returns received messages or status messages from the receive queue. It is important to call <u>ReadFD</u> repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See <u>Receive Status Parameter</u> for more information.

The receive queue can contain up to **32767** messages.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the <u>ReadFD</u> method. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the <u>ReadFD</u> method in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSWARNING, and BUSPASSIVE, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, FD, RTR, Error, or Status message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see

Error Frames).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the

TPCANMsg.MSGTYPE field, the message is a Status message. The ID and DLC fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of **ReadFD** is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error
00h	00h	00h	02h	PCAN_ERROR_OVERRUN
00h	00h	00h	08h	PCAN_ERROR_BUSWARNING
00h	04h	00h	00h	PCAN_ERROR_BUSPASSIVE
00h	00h	00h	10h	PCAN_ERROR_BUSOFF

Example

The following example shows the use of method <u>ReadFD</u> on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized using the method <u>InitializeFD</u> and that the following code is an OnTimer event handler method.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
TPCANMsqFD msq;
<u>TPCANTimestampFD</u> time;
strMsg = new StringBuilder(256);
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic.<u>ReadFD(PCANBasic.PCAN_USBBU</u>
    if (result != TPCANStatus.PCAN_ERROR_QRCVEMPTY
    {
        // Process the received message
        11
        MessageBox.Show("A message was received");
        ProcessMessage(msg);
    }
    else
    {
        // An error occurred, get a text describin
        11
        PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
        MessageBox.Show(strMsg.ToString());
        // Here can be decided if the loop has to
```

```
// status is bus-off)
    //
    HandleReadError(result);
    }
// Try to read a message from the receive queue of
// until the queue is empty
//
}while((result & TPCANStatus.PCAN_ERROR_QRCVEMPTY)
```

C++CLR:

```
<u>TPCANStatus</u> result;
StringBuilder^ strMsg;
TPCANMsqFD msq;
TPCANTimestampFD time;
strMsg = gcnew StringBuilder(256);
do
{
    // Check the receive queue for new messages
    11
    result = PCANBasic::<u>ReadFD(PCANBasic::PCAN_USE</u>
    if (result != TPCANStatus::PCAN_ERROR_QRCVEMPT
    {
        // Process the received message
        11
        MessageBox::Show("A message was received")
        ProcessMessage(msg);
    }
    else
    {
        // An error occurred, get a text describin
        11
        PCANBasic::<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox::Show(strMsg->ToString());
        // Here can be decided if the loop has to
        // status is bus-off)
```

```
//
HandleReadError(result);
}
// Try to read a message from the receive queue of
// until the queue is empty
//
}while((result & TPCANStatus::PCAN_ERROR_QRCVEMPTY
```

Visual Basic:

```
Dim result As TPCANStatus
Dim strMsg As StringBuilder
Dim msg As TPCANMsgFD
Dim time As <u>TPCANTimestampFD</u>
strMsg = New StringBuilder(256)
Do
    ' Check the receive queue for new messages
    result = PCANBasic.<u>ReadFD(PCANBasic.PCAN_USBBU</u>
    If result <> TPCANStatus.PCAN ERROR QRCVEMPTY
        MessageBox.Show("A message was received")
        ProcessMessage(msg)
    Else
        ' An error occurred, get a text describing
        PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox.Show(strMsg.ToString())
          Here can be decided if the loop has to b
          status is bus-off)
        HandleReadError(result)
    End If
' Try to read a message from the receive queue of
 until the queue is empty
Loop While ((result And TPCANStatus.PCAN_ERROR_QRC
```

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    msg: <u>TPCANMsqFD;</u>
    time: TPCANTimestampFD;
begin
    repeat
        // Check the receive queue for new message
        11
        result := TPCANBasic.<u>ReadFD(TPCANBasic.PCA</u>
        If (result <> PCAN_ERROR_QRCVEMPTY) Then
        begin
            // Process the received message
            11
            MessageBox(0, 'A message was received',
            ProcessMessage(msg);
        end
        else
        begin
            // An error occurred, get a text descr
            11
            TPCANBasic.GetErrorText(result, 0, str
            MessageBox(0, strMsg, 'Error', MB_OK);
            // Here can be decided if the loop has
            // status is
                           bus-off)
            11
            HandleReadError(result);
        end:
    // Try to read a message from the receive queu
    // until the queue is empty
    11
    until ((<u>TPCANStatus</u>(Integer(result) AND Intege
```

■ See Also

WriteFD Using Events Error Frames

Plain function Version: <u>CAN_ReadFD</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

PCAN-Basic Documentation

🕇 Home

<^>

ReadFD(TPCANHandle)

Reads a CAN message and its time stamp from the receive queue of a FD capable PCAN Channel.

Syntax

⊟ Pyt	ion
def	<u>ReadFD(</u> self, Channel)

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a 3-touple. The order of the returned values is as follow:

[0]: The method's return value as a <u>TPCANStatus</u> code.

PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsgFD</u> structure.

PCAN_ERROR_ILLOPERATION:	Indicates that the PCAN Channel passed to the method was not initialized using InitializeFD (plain function: CAN_InitializeFD).
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSWARNING:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- warning status.
PCAN_ERROR_BUSPASSIVE:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- passive status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

[1]: A <u>TPCANMsgFD</u> structure with the CAN message read.

[2]: A <u>TPCANTimestampFD</u> value with the time when a message was read.

🗉 Remarks

The use of <u>Read</u> and <u>ReadFD</u> are mutually exclusive. The PCAN Channel passed to this method **must be** initialized using <u>InitializeFD</u> (**plain function**: <u>CAN_InitializeFD</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

The ReadFD method returns received messages or status messages from the receive queue. It is important to call ReadFD repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See Receive Status Parameter for more information.

The receive queue can contain up to **32767** messages.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the <u>ReadFD</u> method. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the <u>ReadFD</u> method in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSWARNING, and BUSPASSIVE, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, FD, RTR, Error, or Status

message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and DLC fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of <u>ReadFD</u> is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error
00h	00h	00h	02h	PCAN_ERROR_OVERRUN
00h	00h	00h	08h	PCAN_ERROR_BUSWARNING
00h	04h	00h	00h	PCAN_ERROR_BUSPASSIVE
00h	00h	00h	10h	PCAN_ERROR_BUSOFF



Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of method <u>ReadFD</u> on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized using the method <u>InitializeFD</u> and that the following code is executed periodically.

Python:

```
readResult = PCAN_ERROR_OK,
while (readResult[0] & PCAN_ERROR_QRCVEMPTY) != PC
  # Check the receive queue for new messages
  #
  readResult = objPCAN.ReadFD(PCAN_USBBUS1)
  if readResult[0] != PCAN_ERROR_QRCVEMPTY:
    # Process the received message
    #
    print "A message was received"
    ProcessMessage(result[1],result[2]) # Poss
  else:
    # An error occurred, get a text describing
```

result = objPCAN.<u>GetErrorText(readResult[@print result[1]</u> HandleReadError(readResult[0]) # Possible

■ See Also

<u>WriteFD</u>

Using Events

Error Frames

Plain function Version: CAN_ReadFD

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

PCAN-Basic Documentation

🕇 Home

< ^ >

Write

Transmits a CAN message.

Syntax

∃ Pascal OO

```
class function Write(
    Channel: <u>TPCANHandle;</u>
    var MessageBuffer: <u>TPCANMsg</u>
): <u>TPCANStatus;</u>
```

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_N public static extern TPCANStatus Write([MarshalAs(UnmanagedType.U1)] TPCANHandle Channel, ref TPCANMsg MessageBuffer);

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN V
static TPCANStatus Write(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel,
    TPCANMsg %MessageBuffer);
```

B Visual Basic

⊨ Pyt	hon
def	Write(self, Channel, MessageBuffer)

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsg</u> buffer containing the CAN message to be sent.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsg</u> structure.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QXMTFULL:	Indicates that the transmit

🗉 Remarks

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a method directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSHEAVY, and BUSLIGTH, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method Write on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
<u>TPCANStatus</u> result;
StringBuilder strMsg;
```

```
<u>TPCANMsg</u> msg;
```

```
strMsg = new StringBuilder(256);
// A CAN message is created. The Data field (8 byt
// of the message must also be created
11
msg = new <u>TPCANMsg();</u>
msg.DATA = new Byte[8];
// A CAN message is configured
11
msg.ID = 0 \times 100;
msg.MSGTYPE = TPCANMessageType.PCAN MESSAGE STANDA
msg.LEN = 3;
msq.DATA[0] = 1;
msg.DATA[1] = 2;
msg.DATA[2] = 3;
// The message is sent using the PCAN-USB Channel
11
result = PCANBasic.Write(PCANBasic.PCAN USBBUS1, r
if (result != TPCANStatus.PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic GetErrorText(result, 0, strMsg);
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("Message sent successfully");
```

C++/CLR:

```
<u>TPCANStatus</u> result;
StringBuilder^ strMsg;
<u>TPCANMsg</u>^ msg;
```

```
strMsg = gcnew StringBuilder(256);
// A CAN message is created. The Data field (8 byt
// of the message must also be created
11
msg = gcnew TPCANMsg();
msg->DATA = gcnew array<Byte>(8);
// A CAN message is configured
11
msg \rightarrow ID = 0x100;
msg->MSGTYPE = TPCANMessageType::PCAN MESSAGE STAN
msq->LEN = 3;
msq - DATA[0] = 1;
msg->DATA[1] = 2;
msg->DATA[2] = 3;
// The message is sent using the PCAN-USB Channel
11
result = PCANBasic::Write(PCANBasic::PCAN USBBUS1,
if (result != TPCANStatus::PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("Message sent successfully");
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim msg As <u>TPCANMsg</u>
strMsg = New StringBuilder(256)
```

```
' A CAN message is created. The Data field (8 byte
 of the message must also be created
msg = New <u>TPCANMsg</u>
msg.DATA = CType(Array.CreateInstance(GetType(Byte
' A CAN message is configured
msg.ID = \&H100
msg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDA
msg.LEN = 3
msq.DATA(0) = 1
msg.DATA(1) = 2
msq.DATA(2) = 3
' The message is sent using the PCAN-USB Channel 1
result = PCANBasic.Write(PCANBasic.PCAN_USBBUS1, m
If result <> TPCANStatus.PCAN ERROR OK Then
    ' An error occurred, get a text describing the
    PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
    MessageBox.Show(strMsg.ToString)
Else
    MessageBox.Show("Message sent successfully")
End If
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    msg: <u>TPCANMsg;</u>
begin
    // A CAN message is configured
    //
    msg.ID := $100;
    msg.MSGTYPE := PCAN_MESSAGE_STANDARD;
```

```
msg.LEN := 3;
msg.DATA[0] := 1;
msg.DATA[1] := 2;
msg.DATA[2] := 3;
// The message is sent using the PCAN-USB Chan
11
result := TPCANBasic.Write(TPCANBasic.PCAN_USE
If (result <> PCAN_ERROR_OK) Then
begin
    // An error occurred, get a text describin
    11
    TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
    MessageBox(0, strMsg, 'Error', MB_OK);
end
else
    MessageBox(0, 'Message sent successfully', '
```

Python:

```
# A CAN message is configured
#
msg = TPCANMsg()
msq.ID = 0x100
msq.MSGTYPE = PCAN MESSAGE STANDARD
msq.LEN = 3
msq.DATA[0] = 1
msg.DATA[1] = 2
msg.DATA[2] = 3
   The message is sent using the PCAN-USB Channel
#
#
result = objPCAN.Write(PCAN USBBUS1,msq)
if result != PCAN ERROR OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.<u>GetErrorText(result)</u>
    print result
```

```
else:
print "Message sent successfully"
```

■ See Also

Read

<u>SetValue</u>

Plain function Version: CAN_Write

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

PCAN-Basic Documentation

🔒 Home

< ^ >

WriteFD

Transmits a CAN message using a FD capable PCAN Channel.

Syntax

Pascal OO
class function WriteFD(
 Channel: TPCANHandle;
 var MessageBuffer: TPCANMsgFD
): TPCANStatus;

C#
[DllImport("PCANBasic.dll", EntryPoint = "CAN_V
public static extern TPCANStatus WriteFD(
 [MarshalAs(UnmanagedType.U1)]
 TPCANHandle Channel,

```
ref TPCANMsgFD MessageBuffer);
```

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN V
static TPCANStatus WriteFD(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel,
    TPCANMsgFD %MessageBuffer);
```

B Visual Basic

def	WriteFD(self, Channel, MessageBuffer)		

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsgFD</u> buffer containing the CAN message to be sent.

■ Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsgFD</u> structure.
PCAN_ERROR_ILLOPERATION:	Indicates that the PCAN Channel passed to the method was not initialized using InitializeFD (plain function: CAN_InitializeFD).
PCAN_ERROR_INITIALIZE:	Indicates that the given

	PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QXMTFULL:	Indicates that the transmit queue of the Channel is full.

🗉 Remarks

The use of <u>Write</u> and WriteFD are mutually exclusive. The PCAN Channel passed to this method **must be** initialized using <u>InitializeFD</u> (**plain function**: <u>CAN_InitializeFD</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a method directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSWARNING, and BUSPASSIVE, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in

a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method WriteFD on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized using the method <u>InitializeFD</u>.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
TPCANMsqFD msq;
strMsg = new StringBuilder(256);
// A CAN message is created. The Data field (64 by
// of the message must also be created
11
msg = new <u>TPCANMsqFD();</u>
msg.DATA = new Byte[64];
// A CAN message is configured
11
msg.ID = 0 \times 100;
msg.MSGTYPE = TPCANMessageType.PCAN MESSAGE STANDA
// DLC 9 means 12 data bytes
11
msg.DLC = 9;
for(byte i=0; i < 12; i++)</pre>
    msg.DATA[i] = i;
  The message is sent using the PCAN-USB Channel
```

```
//
result = PCANBasic.WriteFD(PCANBasic.PCAN_USBBUS1,
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic.GetErrorText(result, 0, strMsg);
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("Message sent successfully");
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
TPCANMsgFD^ msg;
strMsg = gcnew StringBuilder(256);
// A CAN message is created. The Data field (64 by
// of the message must also be created
11
msg = gcnew <u>TPCANMsgFD();</u>
msg->DATA = gcnew array<Byte>(64);
// A CAN message is configured
11
msg \rightarrow ID = 0 \times 100;
msg->MSGTYPE = TPCANMessageType::PCAN_MESSAGE_STAN
// DLC 9 means 12 data bytes
11
msq - DLC = 9;
for(Byte i=0; i < 12; i++)</pre>
    msg->DATA[i] = i;
// The message is sent using the PCAN-USB Channel
```

```
//
result = PCANBasic::WriteFD(PCANBasic::PCAN_USBBUS
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("Message sent successfully");
```

Visual Basic:

```
Dim result As TPCANStatus
Dim strMsg As StringBuilder
Dim msg As TPCANMsgFD
strMsg = New StringBuilder(256)
' A CAN message is created. The Data field (64 byt
' of the message must also be created
msg = New <u>TPCANMsgFD</u>
msg.DATA = CType(Array.CreateInstance(GetType(Byte))
' A CAN message is configured
msq.ID = \&H100
msg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDA
' DLC 9 means 12 data bytes
r.
msq.DLC = 9
For i As Byte = 0 To 11
    msg.DATA(i) = i
Next
```
Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    msg: <u>TPCANMsqFD;</u>
    I: Integer;
begin
    // A CAN message is configured
    11
    msg.ID := $100;
    msg.MSGTYPE := TPCANMessageType(Byte(PCAN_MESS
    // DLC 9 means 12 data bytes
    11
    msq.DLC := 9;
    for I:=0 To 11 do
        msg.DATA[I] := I;
    // The message is sent using the PCAN-USB Chan
    11
    result := TPCANBasic.WriteFD(TPCANBasic.PCAN_U
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        17
```

```
TPCANBasic.GetErrorText(result, 0, strMsg)
MessageBox(0, strMsg, 'Error',MB_OK);
end
else
MessageBox(0,'Message sent successfully','
```

Python:

```
# A CAN message is configured
#
msg = TPCANMsgFD()
msg.ID = 0x100
msg.MSGTYPE = PCAN_MESSAGE_STANDARD.value | PCAN_M
# DLC 9 means 12 data bytes
#
msg.DLC = 9
for i in range(12):
    msg.DATA[i] = i
   The message is sent using the PCAN-USB Channel
#
#
result = objPCAN.WriteFD(PCAN_USBBUS1,msg)
if result != PCAN ERROR OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.<u>GetErrorText(result)</u>
    print result
else:
    print "Message sent successfully"
```

■ See Also

ReadFD SetValue

Plain function Version: <u>CAN_WriteFD</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

PCAN-Basic Documentation

Home
 Hom

<^>

GetValue

Retrieves information from a PCAN Channel.

Overloads

	Function	Description
	<u>GetValue(TPCANHandle, TPCANParameter,</u> <u>String, UInt32)</u>	Retrieves information from a PCAN Channel in text form.
≈ ♦ 8 3	<u>GetValue(TPCANHandle, TPCANParameter,</u> <u>UInt32, UInt32</u>)	Retrieves information from a PCAN Channel in numeric form (32-Bit).
≈ ♦ 8 3	<u>GetValue(TPCANHandle, TPCANParameter,</u> <u>UInt64, UInt32</u>)	Retrieves information from a PCAN Channel in numeric form (64-Bit).
- :	GetValue(TPCANHandle,TPCANParameter)	Retrieves information from a PCAN Channel.

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

< ^ >

GetValue(TPCANHandle, TPCANParameter, String, UInt32)

Retrieves information from a PCAN Channel in text form.

Syntax

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_(public static extern TPCANStatus GetValue([MarshalAs(UnmanagedType.U1)] TPCANHandle Channel, [MarshalAs(UnmanagedType.U1)] TPCANParameter Parameter, StringBuilder StringBuffer, UInt32 BufferLength);

□ C++ / CLR

[DllImport("PCANBasic.dll", EntryPoint = "CAN_(
static TPCANStatus GetValue(
 [MarshalAs(UnmanagedType::U1)]
 TPCANHandle Channel,
 [MarshalAs(UnmanagedType::U1)]
 TPCANParameter Parameter,
 StringBuilder^ StringBuffer,

UInt32 BufferLength);

Visual Basic

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the information to be retrieved (see <u>TPCANParameter</u>).
StringBuffer	The buffer to return the required string value.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL: Indicates that the

	parameters passed to the method are invalid. Check the parameter 'StringBuffer'; it should point to a valid null- terminated string buffer.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be read.

Remarks

Use the method <u>GetValue</u> to get information about PCAN environment as parameters like the Message Filter and values like the availability of a PCAN-Channel. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

More information about the parameters and values that can be read can be found in <u>Parameter Value Definitions</u>.

Example

The following example shows the use of the method <u>GetValue</u> on the channel PCAN_USBBUS1 to get the PCAN-Channel version text. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
TPCANStatus result;
StringBuilder strMsg;
strMsg = new StringBuilder(256);
// The version of the PCAN-USB Channel 1 is asked.
11
result = PCANBasic.<u>GetValue(PCANBasic.PCAN_USBBUS1</u>
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
    // Show the version message
    11
    MessageBox.Show(strMsg.ToString());
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
strMsg = gcnew StringBuilder(256);
// The version of the PCAN-USB Channel 1 is asked.
//
result = PCANBasic::GetValue(PCANBasic::PCAN_USBBU
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
```

```
PCANBasic::GetErrorText(result, 0, strMsg);
MessageBox::Show(strMsg->ToString());
}
else
    // Show the version message
    //
MessageBox::Show(strMsg->ToString());
```

Visual Basic:

Pascal OO:

```
var
    result: <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
begin
    // The version of the PCAN-USB Channel 1 is as
    //
```

∃ See Also

<u>SetValue</u> <u>TPCANParameter</u>

Parameter Value Definitions

Plain function Version: <u>CAN_GetValue</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

1 Home

< ^ >

GetValue(TPCANHandle, TPCANParameter, UInt32, UInt32)

Retrieves information from a PCAN Channel in numeric form.

Syntax

∃ Pascal OO
class function <u>GetValue(</u>
Channel: <u>TPCANHandle</u> ;
<pre>Parameter: <u>TPCANParameter;</u></pre>
NumericBuffer: PLongWord;
BufferLength: LongWord
): <u>TPCANStatus</u> ; overload;

□ C#

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN_(
public static extern TPCANStatus GetValue(
    [MarshalAs(UnmanagedType.U1)]
    TPCANHandle Channel,
    [MarshalAs(UnmanagedType.U1)]
    TPCANParameter Parameter,
    out UInt32 NumericBuffer,
    UInt32 BufferLength);
```

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN (
static TPCANStatus GetValue(
   [MarshalAs(UnmanagedType::U1)]
   TPCANHandle Channel,
   [MarshalAs(UnmanagedType::U1)]
   TPCANParameter Parameter,
   UInt32 %NumericBuffer,
```

UInt32 BufferLength);

Visual Basic

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the information to be retrieved (see <u>TPCANParameter</u>).
NumericBuffer	The buffer to return the required numeric value.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL: Indicates that the

	parameters passed to the method are invalid. Check the parameter 'NumericBuffer'; it should point to an integer buffer.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be read.

Remarks

Use the method <u>GetValue</u> to get information about PCAN environment as parameters like the Message Filter and values like the availability of a PCAN-Channel. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

More information about the parameters and values that can be read can be found in <u>Parameter Value Definitions</u>.

Example

The following example shows the use of the method <u>GetValue</u> on the channel PCAN_USBBUS1 to check if the Message Filter is fully opened. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
TPCANStatus result;
StringBuilder strMsg;
UInt32 iBuffer;
strMsg = new StringBuilder(256);
// The status of the message filter of the PCAN-US
11
result = PCANBasic.<u>GetValue(PCANBasic.PCAN_USBBUS1</u>
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic GetErrorText(result, 0, strMsg);
    MessageBox.Show(strMsg.ToString());
}
else
    // A text is shown giving information about th
    //
    switch(iBuffer)
    {
        case PCANBasic.PCAN_FILTER_OPEN:
            MessageBox.Show("The message filter fo
            break;
        case PCANBasic.PCAN_FILTER_CLOSE:
            MessageBox.Show("The message filter fo
            break;
        case PCANBasic.PCAN FILTER CUSTOM:
            MessageBox.Show("The message filter fo
            break;
    }
```

C++/CLR:

TPCANStatus result;

```
StringBuilder^ strMsg;
UInt32 iBuffer;
strMsg = gcnew StringBuilder(256);
// The status of the message filter of the PCAN-US
11
result = PCANBasic::<u>GetValue(PCANBasic::PCAN_USBBU</u>
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    // A text is shown giving information about th
    //
    switch(iBuffer)
    {
        case PCANBasic::PCAN FILTER OPEN:
            MessageBox::Show("The message filter f
            break;
        case PCANBasic::PCAN FILTER CLOSE:
            MessageBox::Show("The message filter f
            break;
        case PCANBasic::PCAN FILTER CUSTOM:
            MessageBox::Show("The message filter f
            break;
    }
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim iBuffer As UInt32
strMsg = New StringBuilder(256)
```

```
' The status of the message filter of the PCAN-USE
result = PCANBasic.<u>GetValue(PCANBasic.PCAN_USBBUS1</u>
If result <> TPCANStatus.PCAN ERROR OK Then
    ' An error occurred, get a text describing the
    PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
    MessageBox.Show(strMsg.ToString())
Else
    ' A text is shown giving information about the
    Select Case iBuffer
        Case PCANBasic.PCAN_FILTER_OPEN
            MessageBox.Show("The message filter fo
        Case PCANBasic.PCAN FILTER CLOSE
            MessageBox.Show("The message filter fo
        Case PCANBasic.PCAN FILTER CUSTOM
            MessageBox.Show("The message filter fo
    End Select
End If
```

Pascal OO:

```
end
else
begin
    // A text is shown giving information abou
    //
    if iBuffer = TPCANBasic.PCAN_FILTER_OPEN t
        MessageBox(0, 'The message filter
    if iBuffer = TPCANBasic.PCAN_FILTER_CLOSE
        MessageBox(0, 'The message filter
    if iBuffer = TPCANBasic.PCAN_FILTER_CUSTOM
        MessageBox(0, 'The message filter
    end;
```

■ See Also

<u>SetValue</u>

TPCANParameter

Parameter Value Definitions

Plain function Version: <u>CAN_GetValue</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🕇 Home

< ^ >

GetValue(TPCANHandle, TPCANParameter, UInt64, UInt32)

Retrieves information from a PCAN Channel in numeric form.

Pascal OO class function <u>GetValue(</u> Channel: TPCANHandle; Parameter: TPCANParameter; NumericBuffer: PUInt64; BufferLength: LongWord): <u>TPCANStatus;</u> overload; **□** C# [DllImport("PCANBasic.dll", EntryPoint = "CAN_(public static extern <u>TPCANStatus</u> <u>GetValue(</u> [MarshalAs(UnmanagedType.U1)] TPCANHandle Channel, [MarshalAs(UnmanagedType.U1)] **TPCANParameter** Parameter, out UInt64 NumericBuffer, UInt32 BufferLength); □ C++ / CLR [DllImport("PCANBasic.dll", EntryPoint = "CAN_C static TPCANStatus GetValue([MarshalAs(UnmanagedType::U1)] TPCANHandle Channel, [MarshalAs(UnmanagedType::U1)] **TPCANParameter** Parameter, UInt64 %NumericBuffer, UInt32 BufferLength);

Visual Basic

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the information to be retrieved (see <u>TPCANParameter</u>).
NumericBuffer	The buffer to return the required 64-bit numeric value.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the
	parameters passed to the
	method are invalid. Check
	the parameter

	'NumericBuffer'; it should point to a 64-bit integer buffer.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be read.

Remarks

Use the method <u>GetValue</u> to get information about PCAN environment as parameters like an Acceptance Filter. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

More information about the parameters and values that can be read can be found in <u>Parameter Value Definitions</u>.

Example

The following example shows the use of the method <u>GetValue</u> on the channel PCAN_USBBUS1 to retrieve the configured message filter as 11-bit acceptance code and mask. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
<u>TPCANStatus</u> result;
StringBuilder strMsg;
UInt64 i64Buffer;
UInt32 iCode, iMask;
strMsg = new StringBuilder(256);
// The 11-bit acceptance filter of the PCAN-USB Ch
11
result = PCANBasic.GetValue(PCANBasic.PCAN USBBUS1
if (result != TPCANStatus.PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
{
    // Code and mask are extracted from the 64-bit
    11
    iCode = Convert.ToUInt32((i64Buffer >> 32);
    iMask = Convert.ToUInt32(i64Buffer & UInt32.Ma
    // A text is shown giving information about th
    11
    MessageBox.Show(string.Format("Configured 11-b
}
```

C++/CLR:

```
<u>TPCANStatus</u> result;
StringBuilder^ strMsg;
UInt64 i64Buffer;
UInt32 iCode, iMask;
```

```
strMsg = gcnew StringBuilder(256);
// The 11-bit acceptance filter of the PCAN-USB Ch
11
result = PCANBasic::<u>GetValue(PCANBasic::PCAN_USBBU</u>
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox::Show(strMsg->ToString());
}
else
{
    // Code and mask are extracted from the 64-bit
    11
    iCode = Convert::ToUInt32((i64Buffer >> 32);
    iMask = Convert::ToUInt32(i64Buffer & UInt32::
    // A text is shown giving information about th
    11
    MessageBox::Show(String::Format("Configured 11
}
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim i64Buffer As UInt64
Dim iCode As UInt32
Dim iMask As UInt32
strMsg = New StringBuilder(256)
' The 11-bit acceptance filter of the PCAN-USB Cha
' result = PCANBasic.<u>GetValue(PCANBasic.PCAN_USBBUS1</u>
If result <> TPCANStatus.PCAN_ERROR_OK Then
```

```
' An error occurred, get a text describing the
'
PCANBasic.GetErrorText(result, 0, strMsg)
MessageBox.Show(strMsg.ToString())
Else
' Code and mask are extracted from the 64-bit
'
iCode = Convert.ToUInt32(i64Buffer >> 32)
iMask = Convert.ToUInt32(i64Buffer And UInt32.
' A text is shown giving information about the
MessageBox.Show(String.Format("Configured 11-b
End If
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    i64Buffer: UInt64;
    iCode, iMask: LongWord;
begin
    // The 11-bit acceptance filter of the PCAN-US
    //
    result := TPCANBasic.<u>GetValue(TPCANBasic.PCAN</u>
    If (result <> PCAN ERROR OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
    begin
        // Code and mask are extracted from the 64
        11
        iCode := i64Buffer shr 32;
```

```
iMask := (i64Buffer And $FFFFFFF);
    // A text is shown giving information abou
    //
    MessageBox(0, PChar(Format('Configured 11-
    end;
end;
```

■ See Also

<u>SetValue</u>

TPCANParameter

Parameter Value Definitions

Plain function Version: <u>CAN_GetValue</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send feedback to this documentation

PCAN-Basic Documentation

🕇 Home

< ^ >

GetValue(TPCANHandle,TPCANParameter)

Retrieves information from a PCAN Channel.

Syntax

⊨ Pyt	hon
def	<u>GetValue(</u> self, Channel, Parameter)

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the information to be retrieved (see <u>TPCANParameter</u>).

Returns

The return value is a 2-touple. The order of the returned values is as follow:

[0]: The method's return value as a <u>TPCANStatus</u> code.

PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the
	parameters passed to the
	method are invalid. Check
	the parameter

	'NumericBuffer'; it should point to an integer buffer.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be read.

[1]: The requested parameter value (the type of the value depends on the <u>TPCANParameter</u> requested).

Remarks

Use the method GetValue to get information about PCAN environment as parameters like the Message Filter and values like the availability of a PCAN-Channel. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

More information about the parameters and values that can be read can be found in <u>Parameter Value Definitions</u>.

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in

a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method <u>GetValue</u> on the channel PCAN_USBBUS1 to check if the Message Filter is fully opened. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

Python:

```
# The status of the message filter of the PCAN-USB
#
result = objPCAN.<u>GetValue(PCAN_USBBUS1,PCAN_MESSAG</u>
if result[0] != PCAN_ERROR_OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.<u>GetErrorText(result)</u>
    print result
else:
    # A text is shown giving information about the
    #
    if result[1] == PCAN_FILTER_OPEN:
        print "The message filter for the PCAN-USB
    elif result[1] == PCAN FILTER CLOSE:
        print "The message filter for the PCAN-USB
    elif result[1] == PCAN_FILTER_CUSTOM:
        print "The message filter for the PCAN-USB
```

■ See Also

<u>SetValue</u> <u>TPCANParameter</u> <u>Parameter Value Definitions</u>

Plain function Version: <u>CAN_GetValue</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\text{feedback}}$ to this documentation

PCAN-Basic Documentation

🕇 Home

< ^ >

SetValue

Sets a configuration or information value within a PCAN Channel.

Overloads

	Function	Description
=¢ @	SetValue(TPCANHandle, TPCANParameter, String, UInt32)	Sets a configuration or information string value within a PCAN Channel.
= ♦ 80	SetValue(TPCANHandle, TPCANParameter, UInt32, UInt32)	Sets a configuration or information numeric value within a PCAN Channel (32-Bit).
	SetValue(TPCANHandle, TPCANParameter, UInt64, UInt32)	Sets a configuration or information numeric value within a PCAN Channel (64-Bit).
≡ ∳	SetValue(TPCANHandle, TPCANParameter, Object)	Sets a configuration or information value within a PCAN Channel.

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

くへと

SetValue(TPCANHandle, TPCANParameter, String, UInt32)

Sets a configuration or information string value within a PCAN Channel.

Syntax

Pascal OO
class function <u>SetValue</u> (
Channel: <u>TPCANHandle</u> ;
Parameter: <u>TPCANParameter</u> ;
StringBuffer: PChar;
BufferLength: LongWord
): <u>TPCANStatus;</u> overload;
∃ C#
[DllImport("PCANBasic.dll", EntryPoint = " <u>CAN_</u>
public static extern <u>TPCANStatus</u> <u>SetValue(</u>
[MarshalAs(UnmanagedType.U1)]
<u>TPCANHandle</u> Channel,
[MarshalAs(UnmanagedType.U1)]
TPCANParameter Parameter,
[MarshalAs(UnmanagedType.LPStr,SizeParamInc
string StringBuffer,
UInt32 BufferLength);
C++/CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN Static TPCANStatus SetValue(
    [MarshalAs(UnmanagedType::U1)]
    TPCANHandle Channel,
    [MarshalAs(UnmanagedType::U1)]
```

```
TPCANParameter Parameter,
[MarshalAs(UnmanagedType::LPStr,SizeParamIr
String^ StringBuffer,
UInt32 BufferLength);
```

Visual Basic

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the value to be set (see <u>TPCANParameter</u>).
StringBuffer	The buffer containing the string value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is

returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the method are invalid. Check the parameter 'StringBuffer'; it should point to a valid null- terminated string buffer.
PCAN_ERROR_CAUTION:	The configuration of a parameter failed due to a no more existing channel. The parameter has been reset on all existing channels.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be set.
PCAN_ERROR_ILLOPERATION:	An underlying process that is generated by a call to this method with the current parameters, is temporarily not allowed. The configuration in relation to the used <u>TPCANParameter</u> must be checked.

🗉 Remarks

Use the method <u>SetValue</u> to set configuration information or environment values of a PCAN Channel as parameters like the Message Filter and values like a custom entry in the log file of PCAN-Basic. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

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

Example

The following example shows the use of the method <u>SetValue</u> on the channel PCAN_NONEBUS to set (and activate) the path for the log file of a PCAN-Basic's debug session. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is not needed to have an initialized PCAN channel for using the Log functionality.

C#:

```
TPCANStatus result;
StringBuilder strMsg;
string strBuffer;
strMsg = new StringBuilder(256);
// The path for the Log file is set.
// Note that this parameter is set using the
// default Channel (PCAN_NONEBUS)
//
strBuffer = "C:\\Users\\Admin\\Desktop";
result = PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS
if (result != TPCANStatus.PCAN_ERROR_OK)
```

```
{
    // An error occurred, get a text describing th
    //
    PCANBasic.<u>GetErrorText(result, 0, strMsg);
    MessageBox.Show(strMsg.ToString());
}
else
    MessageBox.Show("Log path was successfully set</u>
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
String^ strBuffer;
strMsg = gcnew StringBuilder(256);
// The path for the Log file is set.
// Note that this parameter is set using the
// default Channel (PCAN_NONEBUS)
11
strBuffer = "C:\\Users\\Admin\\Desktop";
result = PCANBasic::<u>SetValue(PCANBasic::PCAN_NONEB</u>
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("Log path was successfully se
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim strBuffer As String
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    strBuffer: string;
begin
    // The path for the Log file is set.
    // Note that this parameter is set using the
    // default Channel (PCAN NONEBUS)
    11
    strBuffer := 'C:\\Users\Keneth\Desktop';
    result := TPCANBasic.SetValue(TPCANBasic.PCAN
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
```

else MessageBox(0, 'Log path was successfully s

■ See Also

<u>GetValue</u>

TPCANParameter

Parameter Value Definitions

Plain function Version: <u>CAN_SetValue</u>

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation
🕇 Home

< ^ >

SetValue(TPCANHandle, TPCANParameter, UInt32, UInt32)

Sets a configuration or information numeric value within a PCAN Channel.

Syntax



[MarshalAs(UnmanagedType::U1)]
TPCANHandle Channel,
[MarshalAs(UnmanagedType::U1)]
TPCANParameter Parameter,

```
UInt32 %NumericBuffer,
UInt32 BufferLength);
```

Bisual Basic

Parameters

Parameters	Description	
Channel	The handle of a PCAN Channe (see <u>TPCANHandle</u>).	
Parameter	The code of the value to be set (see <u>TPCANParameter</u>).	
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 <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL: Indicates that the

	parameters passed to the method are invalid. Check the parameter 'NumericBuffer'; it should point to an integer buffer.
PCAN_ERROR_CAUTION:	The configuration of a parameter failed due to a no more existing channel. The parameter has been reset on all existing channels.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be set.
PCAN_ERROR_ILLOPERATION:	An underlying process that is generated by a call to this method with the current parameters, is temporarily not allowed. The configuration in relation to the used <u>TPCANParameter</u> must be checked.

∃ Remarks

Use the method $\underline{\text{SetValue}}$ to set configuration information or

environment values of a PCAN Channel as parameters like the Message Filter and values like a custom entry in the log file of PCAN-Basic. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

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

Example

The following example shows the use of the method <u>SetValue</u> on the channel PCAN_USBBUS1 to close the message filter. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
TPCANStatus result;
StringBuilder strMsg;
UInt32 iBuffer;
strMsg = new StringBuilder(256);
// The message filter is closed
11
iBuffer = PCANBasic.PCAN_FILTER_CLOSE;
result = PCANBasic.<u>SetValue(PCANBasic.PCAN_USBBUS1</u>
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
```

MessageBox.Show("The filter was successfully c

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
UInt32 iBuffer;
strMsg = gcnew StringBuilder(256);
// The message filter is closed
11
iBuffer = PCANBasic::PCAN_FILTER_CLOSE;
result = PCANBasic::SetValue(PCANBasic::PCAN_USBBU
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("The filter was successfully
```

Visual Basic:

```
PCANBasic.GetErrorText(result, 0, strMsg)
MessageBox.Show(strMsg.ToString)
Else
MessageBox.Show("The filter was successfully c
End If
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    iBuffer: LongWord;
begin
    // The message filter is closed
    11
    iBuffer := TPCANBasic.PCAN_FILTER_CLOSE;
    result := TPCANBasic.<u>SetValue(TPCANBasic.PCAN</u>
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText</u>(result, 0, strMsg)
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'The filter was successfully
```

■ See Also

<u>GetValue</u>

TPCANParameter

Parameter Value Definitions

Plain function Version: <u>CAN_SetValue</u>

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🕇 Home

< ^ >

SetValue(TPCANHandle, TPCANParameter, UInt64, UInt32)

Sets a configuration or information numeric value within a PCAN Channel.

Syntax



TPCANParameter Parameter,

```
UInt64 %NumericBuffer,
UInt32 BufferLength);
```

Bisual Basic

Parameters

Parameters	Description	
Channel	The handle of a PCAN Channe (see <u>TPCANHandle</u>).	
Parameter	The code of the value to be set (see <u>TPCANParameter</u>).	
NumericBuffer	The buffer containing the 64-bit numeric value to be set.	
BufferLength	The length in bytes of the given buffer.	

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL: Indicates that the

	parameters passed to the method are invalid. Check the parameter 'NumericBuffer'; it should point to a 64-bit integer buffer.
PCAN_ERROR_CAUTION:	The configuration of a parameter failed due to a no more existing channel. The parameter has been reset on all existing channels.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be set.
PCAN_ERROR_ILLOPERATION:	An underlying process that is generated by a call to this method with the current parameters, is temporarily not allowed. The configuration in relation to the used <u>TPCANParameter</u> must be checked.

∃ Remarks

Use the method <u>SetValue</u> to set configuration information or environment values of a PCAN Channel as parameters like an Acceptance Filter. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

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

Example

The following example shows the use of the method <u>SetValue</u> on the channel PCAN_USBBUS1 to set a message filter as a 11-bit acceptance code and mask, allowing only IDs from 0x100 to 0x103. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
TPCANStatus result;
StringBuilder strMsg;
UInt64 i64Buffer;
strMsg = new StringBuilder(256);
// The acceptance code and mask are packed as 64-b
//
i64Buffer = 0x100; // Acceptance code
i64Buffer <<= 32;
i64Buffer |= 0x003; // Acceptance mask
result = PCANBasic.SetValue(PCANBasic.PCAN_USBBUS1
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    PCANBasic.GetErrorText(result, 0, strMsg);
```

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

MessageBox.Show("The 11-bit acceptance filter

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
UInt64 i64Buffer;
strMsg = gcnew StringBuilder(256);
// The acceptance code and mask are packed as 64-b
11
i64Buffer = 0x100; // Acceptance code
i64Buffer <<= 32;
i64Buffer |= 0x003; // Acceptance mask
result = PCANBasic::<u>SetValue(PCANBasic::PCAN_USBBU</u>
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox::Show(strMsg->ToString());
}
else
    MessageBox::Show("The 11-bit acceptance filter
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim i64Buffer As UInt64
strMsg = New StringBuilder(256)
' The acceptance code and mask are packed as 64-bi
```

```
i64Buffer = &H100 ' Acceptance code
i64Buffer <<= 32
i64Buffer = i64Buffer Or &H3 ' Acceptance mask
result = PCANBasic.SetValue(PCANBasic.PCAN_USBBUS1
If result <> TPCANStatus.PCAN_ERROR_OK Then
    ' An error occurred, get a text describing the
    '
    PCANBasic.GetErrorText(result, 0, strMsg)
    MessageBox.Show(strMsg.ToString)
Else
    MessageBox.Show("The 11-bit acceptance filter
End If
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    i64Buffer: UInt64;
begin
    // The acceptance code and mask are packed as
    11
    i64Buffer := $100;
                                    // Acceptance c
    i64Buffer := i64Buffer shl 32;
    i64Buffer := i64Buffer Or $3; // Acceptance m
    result := TPCANBasic.SetValue(TPCANBasic.PCAN
    If (result <> PCAN ERROR OK) Then
    begin
        // An error occurred, get a text describin
        //
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
        MessageBox(0, 'The 11-bit acceptance filte
end;
```

∃ See Also

<u>GetValue</u>

TPCANParameter

Parameter Value Definitions

Plain function Version: <u>CAN_SetValue</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

SetValue(TPCANHandle, TPCANParameter, Object)

Sets a configuration or information value within a PCAN Channel.

Syntax

⊨ Pyt	∃ Python		
def	<u>SetValue(</u> self, Channel, Parameter, Buffer)		

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the value to be set (see <u>TPCANParameter</u>).
Buffer	The buffer containing the value to be set.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the
	parameters passed to the
	method are invalid. Check

< ^ >

	the parameter 'Buffer'; it should point to a buffer of a type which is accepted by the parameter being configured.
PCAN_ERROR_CAUTION:	The configuration of a parameter failed due to a no more existing channel. The parameter has been reset on all existing channels.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be set.
PCAN_ERROR_ILLOPERATION:	An underlying process that is generated by a call to this method with the current parameters, is temporarily not allowed. The configuration in relation to the used <u>TPCANParameter</u> must be checked.

∃ Remarks

Use the method $\underline{\text{SetValue}}$ to set configuration information or

environment values of a PCAN Channel as parameters like the Message Filter and values like a custom entry in the log file of PCAN-Basic. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

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

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method <u>SetValue</u> on the channel PCAN_NONEBUS to set (and activate) the path for the log file of a PCAN-Basic's debug session. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is not needed to have an initialized PCAN channel for using the Log functionality.

Python:

```
# The path for the Log file is set.
# Note that this parameter is set using the
# default Channel (PCAN_NONEBUS)
#
strBuffer = "C:\Users\Admin\Desktop"
result = objPCAN.SetValue(PCAN_NONEBUS,PCAN_LOG_LO
if result != PCAN_ERROR_OK:
```

```
# An error occurred, get a text describing the
#
result = objPCAN.GetErrorText(result)
print result[1]
else:
    print "Log path was successfully set"
```

■ See Also

<u>GetValue</u>

TPCANParameter

Parameter Value Definitions

Plain function Version: <u>CAN_SetValue</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

PCAN-Basic Documentation

🕇 Home

 $\langle \land \rangle$

FilterMessages

Configures the reception filter.

Syntax

Bascal OO
<pre>class function FilterMessages(Channel: TPCANHandle; FromID: LongWord; ToID: LongWord; Mode: TPCANMode): TPCANStatus;</pre>
∃ C#
<pre>[DllImport("PCANBasic.dll", EntryPoint = "CAN public static extern TPCANStatus FilterMessage [MarshalAs(UnmanagedType.U1)] TPCANHandle Channel, UInt32 FromID, UInt32 ToID, [MarshalAs(UnmanagedType.U1)] TPCANMode Mode);</pre>
C++/CLR
<pre>[DllImport("PCANBasic.dll", EntryPoint = "CAN static TPCANStatus FilterMessages([MarshalAs(UnmanagedType::U1)] TPCANHandle Channel, UInt32 FromID, UInt32 ToID, [MarshalAs(UnmanagedType::U1)] TPCANMode Mode);</pre>

Visual Basic

B Python

def FilterMessages(
 self,
 Channel,
 FromID,
 ToID,
 Mode)

Parameters

Parameters	Description	
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).	
FromID	The lowest CAN ID wanted to be received.	
ToID	The highest CAN ID wanted to be received.	
Mode	The type of the filter being set (see <u>TPCANMode</u>).	

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN
	channel was not found in the list of
	initialized channels of the calling
	application.

Remarks

Note that after a PCAN Channel is initialized, the status of its filter is fully opened. According with the current filter status, calling this method causes the following behavior:

- Filter status is PCAN_FILTER_OPEN: The filter is automatically closed and then configured with the given range of IDs passed to this function [FromID, ToID].
- Filter status is PCAN_FILTER_CLOSE: The filter is set to the given range of IDs passed to this function [FromID, ToID] .
- Filter status is PCAN_FILTER_CUSTOM: The filter is expanded with the given range of Ids [FromID, ToID]. If a smaller or different range is required than a range that has been configured before, the filter has to be closed first before calling the method FilterMessages. To do this use the method <u>SetValue</u>.

The parameter '*Mode*' indicates which kind of ID is being used to register the new filter range. There are two possible values, Standard (11-bit identifier) or Extended (29-bit identifier). Standard frames are using the bit positions 28 to 18 of the Acceptance Mask/Code registers in the SJA1000 CAN controller. Drivers for 82C200 CAN controllers have to shift the bits down to positions 10 to 0.

Take in account that configuring the message filter cause the CAN controller to enter the <u>Reset</u> state. This will affect other applications that communicate with the same PCAN hardware.

Notes:

- 1. There is only one filter for standard and extended CAN messages. It seems that the ID from a standard message uses the most significant 11 bits (bit 18 to 28) of the 29 bits. I.e. the standard ID 400h is also received by indicating an extended ID 10000000h. For this reason it is <u>not recommended</u> to mix standard and extended filters, since it can increase the risk of receiving unwanted messages.
- 2. Multiple calls of **FilterMessages** expand the reception filter.
- 3. It is not guaranteed that an application only receives CAN messages in the range of **FromID** to **ToID**. This is caused by the operating principle of the SJA1000's acceptance filter. See also Philips Data Sheet "SJA1000 Stand-alone CAN-controller".

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method FilterMessages on the channel PCAN_USBBUS1 to receive a custom range of IDs. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C#:

```
<u>TPCANStatus</u> result;
StringBuilder strMsg;
UInt32 iBuffer;
strMsg = new StringBuilder(256);
```

```
// The message filter is closed first to ensure th
11
iBuffer = PCANBasic.PCAN_FILTER_CLOSE;
result = PCANBasic.SetValue(PCANBasic.PCAN USBBUS1
if (result != TPCANStatus.PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
    MessageBox.Show(strMsg.ToString());
}
else
{
    // The message filter is configured to receive
    11
    result = PCANBasic.FilterMessages(PCANBasic.PC
    if (result != TPCANStatus.PCAN ERROR OK)
    {
        // An error occurred, get a text describin
        11
        PCANBasic.<u>GetErrorText(result, 0, strMsg);</u>
        MessageBox.Show(strMsg.ToString());
    }
    else
        MessageBox.Show("Filter successfully confi
}
```

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
UInt32 iBuffer;
strMsg = gcnew StringBuilder(256);
// The message filter is closed first to ensure th
```

```
//
iBuffer = PCANBasic::PCAN_FILTER_CLOSE;
result = PCANBasic::<u>SetValue(PCANBasic::PCAN_USBBU</u>
if (result != TPCANStatus::PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    PCANBasic::GetErrorText(result, 0, strMsg);
    MessageBox::Show(strMsg->ToString());
}
else
{
    // The message filter is configured to receive
    11
    result = PCANBasic::FilterMessages(PCANBasic::
    if (result != TPCANStatus::PCAN ERROR OK)
    {
        // An error occurred, get a text describin
        11
        PCANBasic::<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox::Show(strMsg->ToString());
    }
    else
        MessageBox::Show("Filter successfully conf
}
```

Visual Basic:

```
Dim result As <u>TPCANStatus</u>
Dim strMsg As StringBuilder
Dim iBuffer As UInt32
strMsg = New StringBuilder(256)
' The message filter is closed first to ensure th
'
iBuffer = PCANBasic.PCAN_FILTER_CLOSE
result = PCANBasic.SetValue(PCANBasic.PCAN_USBBUS1
```

```
If result <> TPCANStatus.PCAN_ERROR_OK Then
    ' An error occurred, get a text describing the
    PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
    MessageBox.Show(strMsg.ToString)
Else
    ' The message filter is configured to receive
    result = PCANBasic.FilterMessages(PCANBasic.PC
    If result <> TPCANStatus.PCAN_ERROR_OK Then
        ' An error occurred, get a text describing
        τ.
        PCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox.Show(strMsg.ToString)
    Else
        MessageBox.Show("Filter successfully confi
    End If
End If
```

Pascal OO:

```
var
    result : <u>TPCANStatus;</u>
    strMsg: array [0..256] of Char;
    iBuffer: LongWord;
begin
    // The message filter is closed first to ensu
    11
    iBuffer := TPCANBasic.PCAN_FILTER_CLOSE;
    result := TPCANBasic.<u>SetValue(TPCANBasic.PCAN_</u>
    If (result <> PCAN_ERROR_OK) Then
    begin
        // An error occurred, get a text describin
        11
        TPCANBasic.<u>GetErrorText(result, 0, strMsg)</u>
        MessageBox(0, strMsg, 'Error', MB_OK);
    end
    else
```

Python:

```
The message filter is closed first to ensure th
#
#
result = objPCAN.<u>SetValue(PCAN_USBBUS1,PCAN_MESSAG</u>
if result != PCAN ERROR OK:
    # An error occurred, get a text describing the
    #
    result = objPCAN.<u>GetErrorText(result)</u>
    print result[1]
else:
    # The message filter is configured to receive
    #
    result = objPCAN.FilterMessages(PCAN_USBBUS1,2
    if result != PCAN ERROR OK:
        # An error occurred, get a text describing
        #
        result = objPCAN.GetErrorText(result)
        print result[1]
    else:
        print "Filter successfully configured for
```

∃ See Also

<u>SetValue</u>

Plain function Version: <u>CAN_FilterMessages</u>

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\text{feedback}}$ to this documentation

PCAN-Basic Documentation

🕇 Home

< ^ >

GetErrorText

Gets a descriptive text for an error code.

Overloads

	Function	Description
	<u>GetErrorText(TPCANStatus,UInt16,String)</u>	Gets a descriptive text for an error code.
-=•	GetErrorText(TPCANStatus, int)	Gets a descriptive text for an error code.

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

PCAN-Basic Documentation

🔒 Home

< ^ >

GetErrorText(TPCANStatus,UInt16,String)

Gets a descriptive text for an error code.

Syntax

Pascal OO
class function GetErrorText(
 AnError: TPCANStatus;
 Language: Word;
 StringBuffer: PChar
): TPCANStatus;

□ C#

[DllImport("PCANBasic.dll", EntryPoint = "CAN_(public static extern TPCANStatus GetErrorText([MarshalAs(UnmanagedType.U4)] TPCANStatus AnError, UInt16 Language, StringBuilder StringBuffer);

□ C++ / CLR

```
[DllImport("PCANBasic.dll", EntryPoint = "CAN_(
static TPCANStatus GetErrorText(
   [MarshalAs(UnmanagedType::U4)]
   TPCANStatus AnError,
   UInt16 Language,
   StringBuilder^ StringBuffer);
```

Visual Basic

```
ByVal AnError As <u>TPCANStatus</u>, _
ByVal Language As UInt16, _
ByVal StringBuffer As StringBuilder) As <u>TP(</u>
End Function
```

Parameters

Parameters	Description
AnError	A <u>TPCANStatus</u> error code.
Language	Indicates a "Primary language ID".
StringBuffer	A buffer for a null-terminated char array.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	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 support the following languages:

Language	Primary Language ID
	n

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

Example

The following example shows the use of the method <u>GetErrorText</u> 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#:

MessageBox.Show(strMsg.ToString());

C++/CLR:

```
TPCANStatus result;
StringBuilder^ strMsg;
strMsg = gcnew StringBuilder(256);
// Gets the description text for PCAN_ERROR_INITIA
//
result = PCANBasic::GetErrorText(TPCANStatus::PCAN
if (result != TPCANStatus::PCAN_ERROR_OK)
    // An error occurred, show a message indicatin
    //
    MessageBox::Show("Error when recovering Error-
else
    MessageBox::Show(strMsg->ToString());
```

Visual Basic:

Pascal OO:

```
var
result : <u>TPCANStatus;</u>
strMsg: array [0..256] of Char;
begin
// Gets the description text for PCAN_ERROR_IN
//
result := TPCANBasic.<u>GetErrorText(PCAN_ERROR_I</u>
If (result <> PCAN_ERROR_OK) Then
    // An error occurred, show a message indic
    //
    MessageBox(0, Error when recovering Error-
else
    MessageBox(0, strMsg,'Success', MB_OK);
```

See Also

Primary Language ID

Plain function Version: <u>CAN_GetErrorText</u>

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

PCAN-Basic Documentation

🔒 Home

< ^ >

GetErrorText(TPCANStatus, int)

Gets a descriptive text for an error code.

Syntax

```
    Python

def GetErrorText(
    self,
    Error,
    Language = 0)
```

Parameters

Parameters	Description
Error	A <u>TPCANStatus</u> error code.
Language	Indicates a "Primary language ID".

Returns

The return value is a 2-touple. The order of the returned values is as follow:

[0]: The method's return value as a <u>TPCANStatus</u> code.

PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	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.

[1]: The text corresponding to the given <u>TPCANStatus</u> code.

Remarks

The "Primary language IDs" are codes used by Windows OS from Microsoft, to identify a human language. The PCAN-Basic API currently support 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)

Python Notes

- Class-Method: Unlike the .NET Framework, under Python a variable has to be instantiated with an object of type <u>PCANBasic</u> in order to use the API functionality.
- Python's first argument convention: Under Python, 'self' is a parameter that is automatically included within the call of this method, within a <u>PCANBasic</u> object and hasn't to be indicated in a method call. This parameter represents the calling object itself.

Example

The following example shows the use of the method <u>GetErrorText</u> to get the description of an error. The language of the description's text will be in Spanish.

Python:

```
# Gets the description text for PCAN_ERROR_INITIAL
#
objPCAN = PCANBasic()
result = objPCAN.GetErrorText(PCAN_ERROR_INITIALIZ
if result[0] != PCAN_ERROR_OK:
    # An error occurred, show a message indicating
    #
    print "Error when recovering Error-Code's desc
else:
    print result[1]
```

■ See Also

Primary Language ID

Plain function Version: CAN_GetErrorText

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation
1 Home

< ^ >

Functions

The functions of the PCAN-Basic API are divided in 4 groups of functionality:

Connection

	Function	Description
	CAN_Initialize	Initializes a PCAN Channel.
	<u>CAN_InitializeFD</u>	Initializes a FD capable PCAN Channel.
- :	<u>CAN_Uninitialize</u>	Uninitializes a PCAN Channel.

Configuration

	Function	Description
=∳	CAN_SetValue	Sets a configuration or information value within a PCAN Channel.
.≡�	CAN_FilterMessages	Configures the message's reception filter of a PCAN Channel.

Information

	Function	Description
≡	CAN_GetValue	Retrieves information from a PCAN Channel.
≡ ∳	CAN_GetStatus	Retrieves the current BUS status of a PCAN Channel.
=0		

CAN_GetErrorText	Returns a descriptive text for an error
	code.

Communication

	Function	Description
≡ ∳	CAN_Read	Reads a CAN message from the receive queue of a PCAN Channel.
≡ ∳	CAN_ReadFD	Reads a CAN message from the receive queue of a FD capable PCAN Channel.
.≘∳	<u>CAN_Write</u>	Transmits a CAN message using a connected PCAN Channel.
	CAN_WriteFD	Transmits a CAN message using a connected FD capable PCAN Channel.
≡ ∳	CAN_Reset	Resets the receive and transmit queues of a PCAN Channel.

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

< ^ > Home
 Hom **CAN_Initialize** Initializes a PCAN Channel. **Syntax □** C++ #ifdef __cplusplus #define _DEF_ARG =0 // Using of default argur #else #define _DEF_ARG #endif TPCANStatus __stdcall CAN_Initialize(TPCANHandle Channel, TPCANBaudrate Btr0Btr1, TPCANType HwType _DEF_ARG, DWORD IOPort _DEF_ARG, WORD Interrupt _DEF_ARG);

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Btr0Btr1	The speed for the communication (BTR0BTR1 code).
НwТуре	The type of the Non-Plug-and- Play hardware and its operation

	mode.
IOPort	The I/O address for the parallel port of the Non-Plug-and-Play hardware.
Interrupt	The Interrupt number of the parallel port of the Non-Plug-and-Play hardware.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_CAUTION:	Indicates that the channel has been initialized but at a different bit rate as the given one.
PCAN_ERROR_ILLHANDLE:	Indicates that the desired PCAN Channel is not valid. Check the list of <u>valid</u> <u>Channels</u> .
PCAN_ERROR_ILLHW:	indicates that the desired PCAN Channel is not available.
PCAN_ERROR_ILLOPERATION:	Indicates that an action cannot be executed due to the state of the hardware. Possible causes are:
	• The desired PCAN- Channel is a LAN Channel, which uses a different bit rate than the specified.

PCAN_ERROR_REGTEST:	(Not-Plug-And-Play Only) Indicates a problem with hardware registration, normally due to wrong values in the parameters ' <i>HwType</i> ', ' <i>IOPort</i> ' and ' <i>Interrupt</i> '.
PCAN_ERROR_INITIALIZE:	Indicates that the desired PCAN channel cannot be connected because it is already in use (PCAN-Basic / PCAN-Light environment).
PCAN_ERROR_NETINUSE:	Indicates that the desired PCAN-Channel is being used with a different bit rate (PCAN-View).
PCAN_ERROR_HWINUSE:	Indicates that the desired PCAN-Channel is being used (CanApi2 connection).
PCAN_ERROR_NODRIVER:	The driver needed for connecting the desired PCAN Channel is not loaded.

Remarks

Note on correspondence of Functions:

A Channel that is initialized using CAN_Initialize must use <u>CAN_Read</u> and <u>CAN_Write</u> for communication. Calling <u>CAN_ReadFD</u> and/or <u>CAN_WriteFD</u> will result in a **PCAN_ERROR_ILLOPERATION** error.

As indicated by its name, the CAN_Initialize function initiates a PCAN Channel, preparing it for communicate within the CAN bus connected to it. Calls to the API functions will fail if they are used with a Channel handle, different than PCAN_NONEBUS, that has not been initialized jet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCAN Channel means:

- to reserve the Channel for the calling application/process.
- to allocate channel resources, like receive and transmit queues.
- to register/connect the Hardware denoted by the channel handle.
- to check and adapt the bus speed, if the Channel is already in use. (Only if the Channel was pre-configured as Bitrate Adapting; see: <u>Bitrate-Adapting Parameter</u>).
- to set the channel in Listen-Only mode. (Only *if the channel was pre-configured as Listen-Only; see:* Listen-Only Parameter).
- to open the messages filter for the application/process.
- to set-up the default values of the different parameters (See <u>CAN_GetValue</u>).
- to set the Receive Status of the channel. (*Pre-configured value; see:* <u>Receive Status Parameter</u>).

Different than the PCAN-Light API, the Initialization process will fail if an application try to initialize a PCAN-Channel that has been initialized already within the same process.

The PCAN-Basic API use the same function for initializations of both, Plug-And-Play and Not-Plug-And-Play hardware. The CAN_Initialize function has three additional parameters that are only for the connection of Non-Plug-And-Play hardware. With Plug-And-Play hardware, however, only two parameters are to be supplied. The remaining three are not evaluated.

Take in consideration that initializing a channel causes a reset of the CAN hardware , when the bus status is other than OK. In this way errors like BUSOFF, BUSHEAVY, and BUSLIGHT, are removed.

PCAN-LAN Channels

A PCAN-LAN channel doesn't allow changing the bit rate using PCAN-Basic. In order to connect a PCAN-LAN Channel it is necessary to know the bit rate of the PCAN-Gateway device that is represented by that channel. If the bit rate is not known, the parameter <u>Bitrate-Adapting</u> should be used.

Example

The following example shows the initialize and uninitialize processes for a Plug-And-Play channel (channel 2 of a PCAN-PCI hardware) and for a Not-Plug-And-Play channel (channel 1 of the PCAN-DNG). In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

C++:

```
TPCANStatus result;
char strMsg[256];
// The Plug & Play Channel (PCAN-PCI) is initializ
11
result = CAN_Initialize(PCAN_PCIBUS2, PCAN_BAUD_500
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
else
    MessageBox("PCAN-PCI (Ch-2) was initialized");
// The Not Plug & Play Channel (PCAN-Dongle) is in
11
result = CAN_Initialize(PCAN_DNGBUS1, PCAN_BAUD_50
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    CAN GetErrorText(result, 0, strMsg);
```

```
MessageBox(strMsg);
}
else
   MessageBox("PCAN-Dongle (Ch-1) was initialized
....
// All initialized channels are released
//
CAN Uninitialize(PCAN_NONEBUS);
```

■ See Also

CAN_Uninitialize CAN_GetValue Understanding PCAN-Basic

Class-method Version: Initialize

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\text{feedback}}$ to this documentation

🕇 Home

<^>

CAN_InitializeFD

Initializes a FD capable PCAN Channel.

Syntax

■ C++
TPCANStatus __stdcall CAN_InitializeFD(
TPCANHandle Channel,
TPCANBitrateFD BitrateFD);

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).
BitrateFD	The speed for the communication (<u>FD Bitrate</u> string).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_CAUTION:	Indicates that the channel has been initialized but at a different bit rate as the given one.
PCAN_ERROR_ILLHANDLE:	Indicates that the desired PCAN Channel is not valid. Check the list of <u>valid</u>

	<u>Channels</u> .
PCAN_ERROR_ILLHW:	Indicates that the desired PCAN Channel is not available.
PCAN_ERROR_ILLOPERATION:	 Indicates that an action cannot be executed due to the state of the hardware. Possible causes are: The desired PCAN Channel is not FD capable and cannot be initialized using this method. The desired PCAN- Channel is a LAN Channel is a LAN Channel, which uses a different bit rate than the specified.
PCAN_ERROR_INITIALIZE:	Indicates that the desired PCAN channel cannot be connected because it is already in use (PCAN-Basic / PCAN-Light environment).
PCAN_ERROR_NETINUSE:	Indicates that the desired PCAN-Channel is being used with a different bit rate (PCAN-View).
PCAN_ERROR_HWINUSE:	Indicates that the desired PCAN-Channel is being used (CanApi connection).
PCAN_ERROR_NODRIVER:	The driver needed for

Remarks

Note on correspondence of functions:

A Channel that is initialized using CAN_InitializeFD must use <u>CAN_ReadFD</u> and <u>CAN_WriteFD</u> for communication. Calling <u>CAN_Read</u> and/or <u>CAN_Write</u> will result in a **PCAN_ERROR_ILLOPERATION** error.

As indicated by its name, the CAN_InitializeFD function initiates a FD capable PCAN Channel, preparing it for communicate within the CAN bus connected to it. Calls to the API functions will fail if they are used with a Channel handle, different than PCAN_NONEBUS, that has not been initialized yet. Each initialized channel should be released when it is not needed anymore.

Initializing a PCAN Channel means:

- to reserve the Channel for the calling application/process.
- to allocate channel resources, like receive and transmit queues.
- to register/connect the Hardware denoted by the channel handle.
- to check and adapt the bus speed, if the Channel is already in use. (Only if the Channel was pre-configured as Bitrate Adapting; see: <u>Bitrate-Adapting Parameter</u>).
- to set the channel in Listen-Only mode. (Only *if the channel* was pre-configured as Listen-Only; see: Listen-Only Parameter).
- to open the messages filter for the application/process.
- to set-up the default values of the different parameters (See <u>CAN_GetValue</u>).
- to set the Receive Status of the channel. (*Pre-configured value; see:* <u>Receive Status Parameter</u>).

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

Take in consideration that initializing a channel causes a reset of the CAN hardware , when the bus status is other than OK. In this way errors like BUSOFF, BUSWARNING, and BUSPASSIVE, are removed.

PCAN-LAN Channels

A PCAN-LAN channel doesn't allow changing the bit rate using PCAN-Basic. In order to connect a PCAN-LAN Channel it is necessary to know the bit rate of the PCAN-Gateway device that is represented by that channel. If the bit rate is not known, the parameter <u>Bitrate-Adapting</u> should be used.

Example

The following example shows the initialize and uninitialize processes for a FD capable channel (channel 1 of a PCAN-USB Pro FD hardware). In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

C++:

```
TPCANStatus result;
char strMsg[256];
TPCANBitrateFD bitrate;
// Defines a FD Bit rate string with nominal and d
//
bitrate = "f_clock_mhz=24, nom_brp=1, nom_tseg1=17
// The FD capable Channel (PCAN-USB Pro FD) is ini
//
result = CAN_InitializeFD(PCAN_USBBUS1, bitrate);
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    CAN_GetErrorText(result, 0, strMsg);
```

```
MessageBox(strMsg);
}
else
MessageBox("PCAN-USB Pro FD (Ch-1) was initial
```

■ See Also

```
CAN_Uninitialize
```

CAN_ReadFD

CAN_WriteFD

Class-method Version: InitializeFD

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

🕇 Home

< ^ >

CAN_Uninitialize

Uninitializes a PCAN Channel.

Syntax

B C++
TPCANStatus __stdcall CAN_Uninitialize(
 TPCANHandle Channel
);

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN
	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 this possibilities:

Single-Release: Given a handle of a PCAN Channel initialized before with <u>CAN_Initialize</u>. If the given channel can not be found then an error is returned.

Multiple-Release: Giving the handle value PCAN_NONEBUS which instructs the API to search for all channels initialized by the calling application and release them all. This option cause no errors if no hardware were uninitialized.

Transmit-queue at uninitialize: When a PCAN-Basic channel connection is terminated, the underlying hardware's transmit-queue will not immediately be discarded. PCAN-Basic will wait some time before finalizing, so that the hardware has time to send (or try to send) those unsent messages. When the time is up (amount 500 milliseconds), the rest of the messages in the queue (if any) are discarded.

Example

The following example shows the initialize and uninitialize (Single-Release) processes for the PCAN_PCIBUS1 channel. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: To see an example of Multiple-Release, see the <u>CAN_Initialize</u> function.

C++:

```
TPCANStatus result;
char strMsg[256];
// The PCI Channel is initialized
//
result = CAN_Initialize(PCAN_PCIBUS1,PCAN_BAUD_500
if(result != PCAN_ERROR_0K)
{
    // An error occurred, get a text describing th
    //
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
```

```
else
    MessageBox("PCAN-PCI (Ch-1) was initialized");
....
// The PCI Channel is released
//
result = CAN_Uninitialize(PCAN_PCIBUS1);
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    <u>CAN_GetErrorText</u>(result, 0, strMsg);
    MessageBox(strMsg);
}
else
    MessageBox("PCAN-PCI (Ch-1) was released");
```

■ See Also

CAN_Initialize

Class-method Version: Uninitialize

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

< ^ >

CAN_Reset

Resets the receive and transmit queues of a PCAN Channel.

Syntax

⊟ C++	
TPCANStatus	
TPCAN	<u>Handle</u> Channel
);	

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN
	channel was not found in the list of
	initialized channels of the calling
	application.

Remarks

Calling this function ONLY clear the queues of a Channel. A reset of the CAN controller doesn't take place.

Normally a reset of the CAN Controller is desired when a bus-off occur. In this case an application cannot use the channel to communicate anymore, until the CAN controller is reset. Consider using the PCAN-Basic parameter <u>PCAN_BUSOFF_AUTORESET</u> 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 <u>uninitialize</u> and <u>initialize</u> again the channel used. This causes a hardware reset, but only when no more clients are connected to that channel.

Example

The following example shows the use of CAN_Reset on the channel PCAN_PCIBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C++:

```
TPCANStatus result;
char strMsg[256];
// The PCI Channel is reset
//
result = CAN_Reset(PCAN_PCIBUS1);
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
else
    MessageBox("PCAN-PCI (Ch-1) was reset");
```

■ See Also

<u>CAN_Read</u> <u>CAN_Write</u> <u>CAN_SetValue</u>

Class-method Version: Reset

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🕇 Home

< ^ >

CAN_GetStatus

Gets the current BUS status of a PCAN Channel.

Syntax

- Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).

Returns

The return value is a <u>TPCANStatus</u> code. The typical return values are:

PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN Channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSLIGHT:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-light status.
PCAN_ERROR_BUSHEAVY:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-heavy status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the

	given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_OK:	Indicates that the status of the given PCAN Channel is OK.

Remarks

When the hardware status is bus-off, an application cannot communicate anymore. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> 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 <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

Example

The following example shows the use of CAN_GetStatus on the channel PCAN_PCIBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C++:

```
TPCANStatus result;
char strMsg[256];
// Check the status of the PCI Channel
//
result = CAN_GetStatus(PCAN_PCIBUS1);
switch(result)
{
    case PCAN_ERROR_BUSLIGHT:
        MessageBox("PCAN-PCI (Ch-1): Handling a BU
```



■ See Also

Parameter Value Definitions

TPCANParameter

Class-method Version: GetStatus

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

🕇 Home

< ^ >

CAN_Read

Reads a CAN message from the receive queue of a PCAN Channel.

Syntax



Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsg</u> buffer to store the CAN message.
TimestampBuffer	A <u>TPCANTimestamp</u> buffer to get the reception time of the message.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters
	passed to the function are invalid. Check the value of the

MessageBuffer; it should point to a <u>TPCANMsg</u> structure.
Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
Indicates a bus error within the given PCAN Channel. The hardware is in bus-light status.
Indicates a bus error within the given PCAN Channel. The hardware is in bus-heavy status.
Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
Indicates that the receive queue of the Channel is empty.

🗉 Remarks

Specifying the value of NULL for the parameter TimetampBuffer causes reading a message without timestamp, when the reception time is not desired. An "Illegal Parameter Value" error will be returned when the MessageBuffer is wrong or the TimestampBuffer contains a value different than NULL and provokes an internal error, eg. accessing its memory.

The use of CAN_Read and <u>CAN_ReadFD</u> are mutually exclusive. The PCAN Channel passed to this function **must be** initialized using <u>CAN_Initialize</u> (**class-method**: <u>Initialize</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

The CAN_Read function returns received messages or status

messages from the receive queue. It is important to call CAN_Read repeatedly until the queue is empty. In case there are no more messages in queue, the value PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See <u>Receive Status</u> <u>Parameter</u> for more information.

The receive queue can contain up to **32767** messages.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the CAN_Read function. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the CAN_Read function in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSHEAVY, and BUSLIGTH, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, RTR, Error, or Status message. This value should be checked every time a message has been read successfully. If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and LEN fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of **CAN_Read** is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error	E C
00h	00h	00h	02h	PCAN_ERROR_OVERRUN	0
00h	00h	00h	04h	PCAN_ERROR_BUSLIGHT	0
00h	00h	00h	08h	PCAN_ERROR_BUSHEAVY	0
00h	00h	00h	10h	PCAN_ERROR_BUSOFF	0

Example

The following example shows the use of CAN_Read on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized and that the following code is an OnTimer event handler function.

```
C++:
```

```
TPCANMsq msq;
TPCANTimestamp timestamp;
TPCANStatus result;
char strMsg[256];
do
{
    // Check the receive queue for new messages
    11
    result = CAN_Read(PCAN_USBBUS1,&msg,&timestamp
    if(result != PCAN_ERROR_QRCVEMPTY)
    {
        // Process the received message
        11
        MessageBox("A message was received");
        ProcessMessage(msg)
    }
    else
    {
        // An error occurred, get a text describin
        // and handle the error
        11
        CAN GetErrorText(result, 0, strMsg);
        MessageBox(strMsg);
        // Here can be decided if the loop has to
        // status is bus-off)
```

// HandleReadError(result); } // Try to read a message from the receive queue of // until the queue is empty // }while((result & PCAN_ERROR_QRCVEMPTY) != PCAN_ERR

■ See Also

<u>CAN_Write</u> <u>Using Events</u>

Error Frames

Class-method Version: Read

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

Home

< ^ >

CAN_ReadFD

Reads a CAN message from the receive queue of a FD capable PCAN Channel.

Syntax

```
■ C++
TPCANStatus __stdcall CAN_ReadFD(
    TPCANHandle Channel,
    TPCANMsgFD* MessageBuffer,
    TPCANTimestampFD *TimestampBuffer);
```

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsgFD</u> buffer to store the CAN message.
TimestampBuffer	A <u>TPCANTimestampFD</u> buffer to get the reception time of the message.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters
	passed to the function are

	invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsgFD</u> structure.
PCAN_ERROR_ILLOPERATION:	Indicates that the PCAN Channel passed to the function was not initialized using <u>CAN_InitializeFD</u> (class-method: <u>InitializeFD</u>).
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSWARNING:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- warning status.
PCAN_ERROR_BUSPASSIVE:	Indicates a bus error within the given PCAN Channel. The hardware is in bus- passive status.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QRCVEMPTY:	Indicates that the receive queue of the Channel is empty.

🗉 Remarks

Specifying the value of NULL for the parameter TimetampBuffer causes reading a message without timestamp, when the reception

time is not desired. An "Illegal Parameter Value" error will be returned when the MessageBuffer is wrong or the TimestampBuffer contains a value different than NULL and provokes an internal error, eg. accessing its memory.

The use of <u>CAN_Read</u> and CAN_ReadFD are mutually exclusive. The PCAN Channel passed to this function **must be** initialized using <u>CAN_InitializeFD</u> (**class-method**: <u>InitializeFD</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

The CAN_ReadFD function returns received messages or status messages from the receive queue. It is important to call CAN_ReadFD repeatedly until the queue is empty. In case there are no more messages in queue, the value

PCAN_ERROR_QRCVEMPTY is returned. The error code PCAN_ERROR_QRCVEMPTY is also returned if the reception of messages is disabled. See <u>Receive Status Parameter</u> for more information.

The receive queue can contain up to **32767** messages.

There are two possibilities for reading messages from the receive queue of a Channel:

Time-Triggered Reading: Consists in periodically calls to the CAN_ReadFD function. Typically, an application start a timer that every 50 or 100 milliseconds check for messages, calling the CAN_ReadFD function in a loop until the value of PCAN_ERROR_QRCVEMTY or another error condition is reached.

Event-Triggered Reading: Consists in reacting to a notification sent by the PCAN driver to a registered application, when a message is received and inserted in its receive queue. See <u>Using Events</u> to obtain more information about reading with events.

About bus errors / Status messages

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSWARNING, and BUSPASSIVE, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

The message type (see <u>TPCANMessageType</u>) of a CAN message indicates if the message is a 11-bit, 29-bit, FD, RTR, Error, or Status message. This value should be checked every time a message has been read successfully.

If the bit <u>PCAN_MESSAGE_ERRFRAME</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is an Error frame (see <u>Error Frames</u>).

If the bit <u>PCAN_MESSAGE_STATUS</u> is set in the <u>TPCANMsg.MSGTYPE</u> field, the message is a Status message. The ID and DLC fields do not contain valid data. The first 4 data bytes of the message contain the Error Code. The MSB of the Error Code is in data byte 0, the LSB is in data byte 3. If a status message was read the return value of **CAN_ReadFD** is also the error code.

Examples:

Data0	Data1	Data2	Data3	Error
00h	00h	00h	02h	PCAN_ERROR_OVERRUN
00h	00h	00h	08h	PCAN_ERROR_BUSWARNING

00h	04h	00h	00h	PCAN_ERROR_BUSPASSIVE
00h	00h	00h	10h	PCAN_ERROR_BUSOFF

Example

The following example shows the use of CAN_ReadFD on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized using <u>CAN_InitializeFD</u> and that the following code is an OnTimer event handler function.

C++:

```
TPCANMsgFD msg;
TPCANTimestampFD timestamp;
TPCANStatus result;
char strMsg[256];
do
{
    // Check the receive queue for new messages
    //
    result = CAN_ReadFD(PCAN_USBBUS1,&msg,&timesta
    if(result != PCAN_ERROR_QRCVEMPTY)
    {
        // Process the received message
```

```
11
        MessageBox("A message was received");
        ProcessMessage(msg)
    }
    else
    {
        // An error occurred, get a text describin
        // and handle the error
        11
        CAN GetErrorText(result, 0, strMsg);
        MessageBox(strMsg);
        // Here can be decided if the loop has to
        // status is bus-off)
        11
        HandleReadError(result);
    }
// Try to read a message from the receive queue of
// until the queue is empty
//
}while((result & PCAN_ERROR_QRCVEMPTY) != PCAN_ERR
```

See Also

CAN_InitializeFD CAN_WriteFD Using Events Error Frames

Class-method Version: ReadFD

```
Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.
Send <u>feedback</u> to this documentation
```

🕇 Home

< ^ >

CAN_Write

Transmits a CAN message.

Syntax

```
    C++

    TPCANStatus __stdcall CAN_Write(
        TPCANHandle Channel,
        TPCANMsg* MessageBuffer
);
```

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsg</u> buffer containing the CAN message to be sent.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the function are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsg</u> structure.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of

	the calling application.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QXMTFULL:	Indicates that the transmit queue of the Channel is full.

Remarks

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSHEAVY, and BUSLIGTH, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset, but only when no more clients are connected to that channel.

Example

The following example shows the use of CAN_Write on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C++:

```
<u>TPCANMsg</u> msg;
<u>TPCANStatus</u> result;
char strMsg[256];
```
```
// A CAN message is configured
11
msg.ID = 0 \times 100;
msg.MSGTYPE = PCAN_MESSAGE_STANDARD;
msg.LEN = 3;
msg.DATA[0] = 1;
msg.DATA[1] = 2;
msg.DATA[2] = 3;
// The message is sent using the PCAN-USB Channel
11
result = CAN_Write(PCAN_USBBUS1, &msg);
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
else
    MessageBox("Message sent successfully");
```

■ See Also

CAN_Read

CAN_SetValue

Class-method Version: Write

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send $\underline{\text{feedback}}$ to this documentation

🕇 Home

<^>

CAN_WriteFD

Transmits a CAN message using a connected FD capable PCAN Channel.

Syntax

```
■ C++
TPCANStatus __stdcall CAN_WriteFD(
TPCANHandle Channel,
TPCANMsgFD* MessageBuffer);
```

Parameters

Parameters	Description
Channel	The handle of a FD capable PCAN Channel (see <u>TPCANHandle</u>).
MessageBuffer	A <u>TPCANMsgFD</u> buffer containing the CAN message to be sent.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the function are invalid. Check the value of the MessageBuffer; it should point to a <u>TPCANMsgFD</u> structure.

PCAN_ERROR_ILLOPERATION:	Indicates that the PCAN Channel passed to the function was not initialized using <u>CAN_InitializeFD</u> (class-method: InitializeFD).
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_BUSOFF:	Indicates a bus error within the given PCAN Channel. The hardware is in bus-off status.
PCAN_ERROR_QXMTFULL:	Indicates that the transmit queue of the Channel is full.

Remarks

The use of <u>CAN_Write</u> and CAN_WriteFD are mutually exclusive. The PCAN Channel passed to this function **must be** initialized using <u>CAN_InitializeFD</u> (**class-method**: <u>InitializeFD</u>). Otherwise the error PCAN_ERROR_ILLOPERATION is returned.

If a bus-off error occur, an application cannot use the channel to communicate anymore, until the CAN controller is reset. With PCAN-Basic it is not possible to reset the CAN controller through a function directly. Consider using the PCAN-Basic property <u>PCAN_BUSOFF_AUTORESET</u> which instructs the API to automatically reset the CAN controller when a bus-off state is detected.

Another way to reset errors like BUSOFF, BUSWARNING, and BUSPASSIVE, is to <u>uninitialize</u> and <u>initialise</u> again the channel used. This causes a hardware reset.

Example

The following example shows the use of CAN_WriteFD on the channel PCAN_USBBUS1. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is assumed that the channel was already initialized using <u>CAN_InitializeFD</u>.

C++:

```
TPCANMsqFD msg;
TPCANStatus result;
char strMsg[256];
// A CAN FD message is configured
11
msg.ID = 0 \times 100;
msg.MSGTYPE = PCAN_MESSAGE_STANDARD | PCAN_MESSAGE
// DLC 9 means 12 data bytes
//
msq.DLC = 9;
for(int i=0; i < 12; i++)</pre>
    msg.DATA[i] = i;
// The message is sent using the PCAN-USB Channel
11
result = CAN_WriteFD(PCAN_USBBUS1, &msg);
if(result != PCAN ERROR OK)
{
    // An error occurred, get a text describing th
    11
    CAN GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
else
    MessageBox("Message sent successfully");
```

∃ See Also

CAN_InitializeFD CAN_ReadFD

Class-method Version: WriteFD

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🕇 Home

< ^ >

CAN_GetValue

Retrieves information from a PCAN Channel.

Syntax



Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the information to be retrieved (see <u>TPCANParameter</u>).
Buffer	The buffer to return the required value.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the function are invalid. Check the parameter 'Buffer'; it should point to a valid data container for the requested value.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be read.

Remarks

Use the function CAN_GetValue to get information about PCAN environment as parameters like the Message Filter and values like the availability of a PCAN-Channel. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

More information about the parameters and values that can be read can be found in <u>Parameter Value Definitions</u>.

Example

The following example shows the use of CAN_GetValue on the channel PCAN_USBBUS1 to check if the Message Filter is fully opened. In case of failure, the returned code will be translated to a

text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C++:

```
int iBuffer;
TPCANStatus result;
char strMsg[256];
// The status of the message filter of the PCAN-US
11
result = CAN_GetValue(PCAN_USBBUS1, PCAN_MESSAGE_F
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    11
    CAN GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
else
{
    // A text is shown giving information about th
    11
    switch(result != PCAN_ERROR_OK)
    {
        case PCAN FILTER OPEN:
            MessageBox("The message filter for the
            break;
        case PCAN FILTER CLOSE:
            MessageBox("The message filter for the
            break;
        case PCAN FILTER CUSTOM:
            MessageBox("The message filter for the
            break;
    }
}
```

∃ See Also

<u>CAN_SetValue</u> <u>TPCANParameter</u>

Parameter Value Definitions

Class-method Version: GetValue

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

🕇 Home

< ^ >

CAN_SetValue

Sets a configuration or information value within a PCAN Channel.

Syntax



Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
Parameter	The code of the value to be set (see <u>TPCANParameter</u>).
Buffer	The buffer containing the value to be set.
BufferLength	The length in bytes of the given buffer.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	Indicates that the parameters passed to the function are invalid. Check the parameter 'Buffer'; it should point to a valid data container for the requested value.
PCAN_ERROR_CAUTION:	The configuration of a parameter failed due to a no more existing channel. The parameter has been reset on all existing channels.
PCAN_ERROR_INITIALIZE:	Indicates that the given PCAN channel was not found in the list of initialized channels of the calling application.
PCAN_ERROR_ILLPARAMTYPE:	Indicates that the requested information is not available for the given PCAN Channel. Check the value of 'Parameter'; some values are not available for all PCAN-Channels or cannot be set.
PCAN_ERROR_ILLOPERATION:	An underlying process that is generated by a call to this function with the current parameters, is temporarily not allowed. The configuration in relation to the used <u>TPCANParameter</u> must be checked.

∃ Remarks

Use the function CAN_SetValue to set configuration information or environment values of a PCAN Channel as parameters like the Message Filter and values like a custom entry in the log file of PCAN-Basic. Take in account that not all parameters are supported for all PCAN-Channels. The access's type of the parameters can also be different.

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

Example

The following example shows the use of CAN_SetValue on the channel PCAN_USBBUS1 to insert a text into the PCAN-Basic log file. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

Note: It is not needed to have an initialized PCAN channel for using the Log functionality.

C++:

```
TPCANStatus result;
char strMsg[256];
// Sets a text to be included in the Log file of t
//
strcpy(strMsg, "This is a custom text from an appl
// Inserts the given text into the Log file of the
// Note: If the Log functionality is disabled, thi
//activate the log process.
//
result = CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_TEXT,
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
```

```
CAN_GetErrorText(result, 0, strMsg);
MessageBox(strMsg);
}
else
MessageBox("The text was successfully logged i
```

■ See Also

CAN_GetValue

TPCANParameter

Parameter Value Definitions

Class-method Version: <u>SetValue</u>

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

🔒 Home

<^>

CAN_FilterMessages

Configures the reception filter.

Syntax

⊟ C++	
TPCANStatus	_stdcall CAN_FilterMessages(
TPCANH	<u>landle</u> Channel,
DWORD	FromID,
DWORD	TOID,
TPCANM	<u>lode</u> Mode
);	

Parameters

Parameters	Description
Channel	The handle of a PCAN Channel (see <u>TPCANHandle</u>).
FromID	The lowest CAN ID wanted to be received.
ToID	The highest CAN ID wanted to be received.
Mode	The type of the filter being set (see <u>TPCANType</u>).

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_INITIALIZE:

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

Remarks

Note that after a PCAN Channel is initialized, the status of its filter is fully opened. According with the current filter status, calling this function causes the following behavior:

- Filter status is PCAN_FILTER_OPEN: The filter is automatically closed and then configured with the given range of IDs passed to this function [FromID, ToID].
- Filter status is PCAN_FILTER_CLOSE: The filter is set to the given range of IDs passed to this function [FromID, ToID] .
- Filter status is PCAN_FILTER_CUSTOM: The filter is expanded with the given range of Ids [FromID, ToID]. If a smaller or different range is required than a range that has been configured before, the filter has to be closed first before calling the CAN_FilterMessages function. To do this use the function <u>CAN_SetValue</u>.

The parameter '*Mode*' indicates which kind of ID is being used to register the new filter range. There are two possible values, Standard (11-bit identifier) or Extended (29-bit identifier). Standard frames are using the bit positions 28 to 18 of the Acceptance Mask/Code registers in the SJA1000 CAN controller. Drivers for 82C200 CAN controllers have to shift the bits down to positions 10 to 0.

Take in account that configuring the message filter cause the CAN controller to enter the <u>Reset</u> state. This will affect other applications that communicate with the same PCAN hardware.

Notes:

 There is only one filter for standard and extended CAN messages. It seems that the ID from a standard message uses the most significant 11 bits (bit 18 to 28) of the 29 bits. I.e. the standard ID 400h is also received by indicating an extended ID 10000000h. For this reason it is <u>not recommended</u> to mix standard and extended filters, since it can increase the risk of receiving unwanted messages.

- 2. Multiple calls of **CAN_FilterMessages** expand the reception filter.
- 3. It is not guaranteed that an application only receives CAN messages in the range of **FromID** to **ToID**. This is caused by the operating principle of the SJA1000's acceptance filter. See also Philips Data Sheet "SJA1000 Stand-alone CAN-controller".

Example

The following example shows the use of CAN_FilterMessages on the channel PCAN_USBBUS1 to receive a custom range of IDs. In case of failure, the returned code will be translated to a text (according with the operating system language) in English, German, Italian, French or Spanish, and it will be shown to the user.

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

C++:

```
TPCANStatus result;
char strMsg[256];
DWORD iBuffer;
// The message filter is closed first to ensure th
//
iBuffer = PCAN_FILTER_CLOSE;
result = CAN_SetValue(PCAN_USBBUS1,PCAN_MESSAGE_FI
if(result != PCAN_ERROR_OK)
{
    // An error occurred, get a text describing th
    //
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
else
```

{
 // The message filter is configured to receive
 //
 result = CAN_FilterMessages(PCAN_USBBUS1, 2, 5
 if(result != PCAN_ERROR_OK)
 {
 // An error occurred, get a text describin
 //
 CAN_GetErrorText(result, 0, strMsg);
 MessageBox(strMsg);
 }
 else
 MessageBox("Filter successfully configured
}

∃ See Also

CAN_SetValue

Class-method Version: FilterMessages

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🕇 Home

< ^ >

CAN_GetErrorText

Returns a descriptive text for an error code.

Syntax



Parameters

Parameters	Description
Error	A <u>TPCANStatus</u> error code.
Language	Indicates a "Primary language ID".
Buffer	A buffer for a null-terminated char array.

Returns

The return value is a <u>TPCANStatus</u> code. PCAN_ERROR_OK is returned on success. The typical errors in case of failure are:

PCAN_ERROR_ILLPARAMVAL:	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 support the following languages:

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

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

Example

The following example shows the use of CAN_GetErrorText to get the description of an error. The language of the description's text will be the same used by the operating system.

C++:

```
TPCANStatus result;
char strMsg[256];
// Gets the description text for PCAN_ERROR_INITIA
```

∃ See Also

Primary Language ID

Class-method Version: GetErrorText

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{feedback}$ to this documentation

🔒 Home

<^>

Definitions

The PCAN-Basic API defines the following values:

Name	Description
PCAN Handle Definitions	Defines the handles for the different PCAN Channels.
Parameter Value Definitions	Defines the possible values for setting and getting PCAN's environment information with the functions <u>CAN_SetValue</u> and <u>CAN_GetValue</u> .
FD Bit rate Parameter Definitions	Defines the different configuration parameters used to create a Flexible Data rate string for FD capable PCAN- Channels initialization.

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

1 Home

< ^ >

PCAN Handle Definitions

Defines the handles for the different PCAN buses (Channels) within a class. These values are used as parameter where a <u>TPCANHandle</u> is needed.

Default handle value:

	Туре	Constant	Value	Description
=	<u>TPCANHandle</u>	PCAN_NONEBUS	0	Undefined/defau value for a PCAN bus.

Handles for the ISA Bus (Not Plug & Play):

	Туре	Constant	Value	Description
Ξ	<u>TPCANHandle</u>	PCAN_ISABUS1	0x21	PCAN-ISA interface, channel 1.
Ξ	<u>TPCANHandle</u>	PCAN_ISABUS2	0x22	PCAN-ISA interface, channel 2.
Ξ	<u>TPCANHandle</u>	PCAN_ISABUS3	0x23	PCAN-ISA interface, channel 3.
Ξ	<u>TPCANHandle</u>	PCAN_ISABUS4	0x24	PCAN-ISA interface, channel 4.
=	<u>TPCANHandle</u>	PCAN_ISABUS5	0x25	PCAN-ISA interface,

				channel 5.
	<u>TPCANHandle</u>	PCAN_ISABUS6	0x26	PCAN-ISA interface, channel 6.
I	<u>TPCANHandle</u>	PCAN_ISABUS7	0x27	PCAN-ISA interface, channel 7.
-	<u>TPCANHandle</u>	PCAN_ISABUS8	0x28	PCAN-ISA interface, channel 8.

Handles for the **Dongle** Bus (Not Plug & Play):

	Туре	Constant	Value	Description
Ξ	TPCANHandle	PCAN_DNGBUS1	0x31	PCAN- Dongle/LPT interface, channel 1.

Handles for the **PCI** Bus:

	Туре	Constant	Value	Description
=	TPCANHandle	PCAN_PCIBUS1	0x41	PCAN-PCI interface, channel 1.
Ξ	<u>TPCANHandle</u>	PCAN_PCIBUS2	0x42	PCAN-PCI interface, channel 2.
=	<u>TPCANHandle</u>	PCAN_PCIBUS3	0x43	PCAN-PCI interface, channel 3.

-				
u	<u>TPCANHandle</u>	PCAN_PCIBUS4	0x44	PCAN-PCI interface, channel 4.
-	<u>TPCANHandle</u>	PCAN_PCIBUS5	0x45	PCAN-PCI interface, channel 5.
-	<u>TPCANHandle</u>	PCAN_PCIBUS6	0x46	PCAN-PCI interface, channel 6.
	<u>TPCANHandle</u>	PCAN_PCIBUS7	0x47	PCAN-PCI interface, channel 7.
	<u>TPCANHandle</u>	PCAN_PCIBUS8	0x48	PCAN-PCI interface, channel 8.
I	<u>TPCANHandle</u>	PCAN_PCIBUS9	0x409	PCAN-PCI interface, channel 9.
-	<u>TPCANHandle</u>	PCAN_PCIBUS10	0x40A	PCAN-PCI interface, channel 10.
-	<u>TPCANHandle</u>	PCAN_PCIBUS11	0x40B	PCAN-PCI interface, channel 11.
a	<u>TPCANHandle</u>	PCAN_PCIBUS12	0x40C	PCAN-PCI interface, channel 12.
	<u>TPCANHandle</u>	PCAN_PCIBUS13	0x40D	PCAN-PCI interface, channel 13.
=				

	<u>TPCANHandle</u>	PCAN_PCIBUS14	0x40E	PCAN-PCI interface, channel 14.
Ξ	<u>TPCANHandle</u>	PCAN_PCIBUS15	0x40F	PCAN-PCI interface, channel 15.
Ξ	TPCANHandle	PCAN_PCIBUS16	0x410	PCAN-PCI interface, channel 16.

Handles for the **USB** Bus:

	Туре	Constant	Value	Description
I	TPCANHandle	PCAN_USBBUS1	0x51	PCAN-USB interface, channel 1.
-	<u>TPCANHandle</u>	PCAN_USBBUS2	0x52	PCAN-USB interface, channel 2.
н	<u>TPCANHandle</u>	PCAN_USBBUS3	0x53	PCAN-USB interface, channel 3.
H	<u>TPCANHandle</u>	PCAN_USBBUS4	0x54	PCAN-USB interface, channel 4.
H	<u>TPCANHandle</u>	PCAN_USBBUS5	0x55	PCAN-USB interface, channel 5.
Ξ	<u>TPCANHandle</u>	PCAN_USBBUS6	0x56	PCAN-USB interface, channel 6.

Ξ	<u>TPCANHandle</u>	PCAN_USBBUS7	0x57	PCAN-USB interface, channel 7.
=	<u>TPCANHandle</u>	PCAN_USBBUS8	0x58	PCAN-USB interface, channel 8.
=	<u>TPCANHandle</u>	PCAN_USBBUS9	0x509	PCAN-USB interface, channel 9.
	<u>TPCANHandle</u>	PCAN_USBBUS10	0x50A	PCAN-USB interface, channel 10.
Ξ	<u>TPCANHandle</u>	PCAN_USBBUS11	0x50B	PCAN-USB interface, channel 11.
=	<u>TPCANHandle</u>	PCAN_USBBUS12	0x50C	PCAN-USB interface, channel 12.
=	<u>TPCANHandle</u>	PCAN_USBBUS13	0x50D	PCAN-USB interface, channel 13.
=	<u>TPCANHandle</u>	PCAN_USBBUS14	0x50E	PCAN-USB interface, channel 14.
Ξ	<u>TPCANHandle</u>	PCAN_USBBUS15	0x50F	PCAN-USB interface, channel 15.
Ξ	<u>TPCANHandle</u>	PCAN_USBBUS16	0x510	PCAN-USB interface, channel 16.

Handles for the **PC_Card** Bus:

	Туре	Constant	Value	Description
H	<u>TPCANHandle</u>	PCAN_PCCBUS1	0x61	PCAN-PC Card interface, channel 1.
u	<u>TPCANHandle</u>	PCAN_PCCBUS2	0x62	PCAN-PC Card interface, channel 2.

Handles for the LAN Bus:

-LAN Ice, el 1.
-LAN .ce, el 2.
-LAN .ce, el 3.
-LAN .ce, el 4.
-LAN .ce, el 5.
-LAN .ce, el 6.

н	<u>TPCANHandle</u>	PCAN_LANBUS7	0x807	PCAN-LAN interface, channel 7.
-	<u>TPCANHandle</u>	PCAN_LANBUS8	0x808	PCAN-LAN interface, channel 8.
	<u>TPCANHandle</u>	PCAN_LANBUS9	0x809	PCAN-LAN interface, channel 9.
H	<u>TPCANHandle</u>	PCAN_LANBUS10	0x80A	PCAN-LAN interface, channel 10.
=	<u>TPCANHandle</u>	PCAN_LANBUS11	0x80B	PCAN-LAN interface, channel 11.
Ξ	<u>TPCANHandle</u>	PCAN_LANBUS12	0x80C	PCAN-LAN interface, channel 12.
Ξ	<u>TPCANHandle</u>	PCAN_LANBUS13	0x80D	PCAN-LAN interface, channel 13.
=	<u>TPCANHandle</u>	PCAN_LANBUS14	0x80E	PCAN-LAN interface, channel 14.
-	<u>TPCANHandle</u>	PCAN_LANBUS15	0x80F	PCAN-LAN interface, channel 15.
	<u>TPCANHandle</u>	PCAN_LANBUS16	0x810	PCAN-LAN interface, channel 16.

🗉 Remarks

The PCAN_NONEBUS is a value used as default channel value. It is used for general purposes as using and configuring the Log capabilities of the PCAN-Basic API. It can also be used to remove all channel connections made by an application.

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

Hardware Type and Channels:

Not Plug & Play: The hardware channels of this kind are used as registered. This mean, for example, it is allowed to register the PCAN_ISABUS3 without having registered PCAN_ISA1 and PCAN_ISA2. It is a decision of each user, how to associate a PCAN-Channel (logical part) and a port/interrupt pair (physical part).

Plug & Play: For hardware handles of PCI, USB and PC-Card, the availability of the channels is determined by the count of hardware connected to a computer in a given moment, in conjunction with their internal handle. This mean, that having four PCAN-USB connected to a computer will let the user to connect the channels PCAN_USBBUS1 to PCAN_USBBUS4. The association of each channel with a hardware is managed internally using the handle of a hardware.

Python:

The definitions of these values have the following form: <u>TPCANHandle(handle)</u> where *handle* is the value contained in the column with the same name. e.g. *PCAN_PCCBUS1* is defined as <u>TPCANHandle(0x61)</u>.

■ See Also

Parameter Value Definitions

Send feedback to this documentation

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

🔒 Home

<^>

Parameter Value Definitions

Defines the possible values for setting and getting PCAN's environment information with the functions <u>CAN_SetValue</u> and <u>CAN_GetValue</u>.

Activation values:

	Туре	Constant	Value	Description
=	Int32	PCAN_PARAMETER_OFF	0	The PCAN parameter is not set (inactive).
=	Int32	PCAN_PARAMETER_ON	1	The PCAN parameter is set (active).

Filter values:

	Туре	Constant	Value	Description
	Int32	PCAN_FILTER_CLOSE	0	The PCAN filter is closed. No messages will be received.
=	Int32	PCAN_FILTER_OPEN	1	The PCAN filter is fully opened. All messages will be received.
	Int32	PCAN_FILTER_CUSTOM	2	The PCAN filter is custom configured. Only registered

Channel Availability values:

	Туре	Constant	Value
н	Int32	PCAN_CHANNEL_UNAVAILABLE	0
н	Int32	PCAN_CHANNEL_AVAILABLE	1
	Int32	PCAN_CHANNEL_OCCUPIED	2
ш	Int32	PCAN_CHANNEL_PCANVIEW	PCAN_CHANNEL Or PCAN_CHANNEL

μ	μ	, p	v

Log-Configuration values*:

	Туре	Constant	Value	Descript
	Int32	LOG_FUNCTION_DEFAULT	0x00	Logs sys exceptior and error Custom I texts are also included.
-	Int32	LOG_FUNCTION_ENTRY	0x01	Logs the entries to PCAN-Ba API functions
	Int32	LOG_FUNCTION_PARAMETERS	0x02	Logs the paramete passed to the PCAI Basic AP functions
=	Int32	LOG_FUNCTION_LEAVE	0x04	Logs the exits fron the PCAI Basic AP functions
=	Int32	LOG_FUNCTION_WRITE	0x08	Logs the CAN message passed to the

			CAN_Wr function.
Int32	LOG_FUNCTION_READ	0x10	Logs the CAN message received within the <u>CAN_Re</u> function.
Int32	LOG_FUNCTION_ALL	0xFFFF	Logs all possible information within the PCAN-Ba API functions

*These values can be combined using the bitwise inclusive OR operator

Trace-Configuration values*:

	Туре	Constant	Value	Description
a	Int32	TRACE_FILE_SINGLE	0x00	Stores messages in a single file until the configured file size is reached.
-	Int32	TRACE_FILE_SEGMENTED	0x01	Stores messages distributed in several files.
=				

Int32	TRACE_FILE_DATE	0x02	Includes date information into the name of the trace file.
Int32	TRACE_FILE_TIME	0x04	Includes the time information into the name of the trace file.
Int32	TRACE_FILE_OVERWRITE	0x08	Forces the overwriting of available traces (same name).

*These values can be combined using the bitwise inclusive OR operator

Feature values*:

	Туре	Constant	Value	Description
	Int32	FEATURE_FD_CAPABLE	1	The PCAN Channel represents a device that supports flexible data rate (CAN- FD).
н	Int32	FEATURE_DELAY_CAPABLE	2	The PCAN Channel represents a

				device that supports the configuration of a delay between sending frames (FPGA devices only)
--	--	--	--	--

*These values can be combined using the bitwise inclusive OR operator

Service-Status values:

	Туре	Constant	Value	Description
Ξ	Int32	SERIVCE_STATUS_STOPPED	1	The service is not running.
Ξ	Int32	SERVICE_STATUS_RUNNING	4	The service is running.

Remarks

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

Python:

The definitions of these values have the following form: <u>TPCANParameter</u>(*value*) where *value* is the value contained in the column with the same name. e.g. *LOG_FUNCTION_ENTRY* is defined as <u>TPCANParameter</u>(0x01).

■ See Also

PCAN Handle Definitions

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send $\underline{\text{feedback}}$ to this documentation
🔒 Home

< ^ >

FD Bit rate Parameter Definitions

Defines the different configuration parameters used to create a Flexible Data rate string for FD capable PCAN-Channels initialization. These values are used as parameter with <u>CAN_InitializeFD</u> (**class-method**: <u>InitializeFD</u>).

Clock Frequency parameters:

	Туре	Constant	Value	Descriptio
	String	PCAN_BR_CLOCK	"f_clock"	Clock frequency Hertz (80000000 60000000 40000000 30000000 24000000 20000000
Ξ	String	PCAN_BR_CLOCK_MHZ	"f_clock_mhz"	Clock frequency Megahertz (80, 60, 40 30, 24, 20)

Nominal Bit rate parameters:

Туре	Constant	Value	Descriptior
 String	PCAN_BR_NOM_BRP	"nom_brp"	Clock presc for nominal quantum (11024).

	String	PCAN_BR_NOM_TSEG1	"nom_tseg1"	TSEG1 segment for nominal bit in time quar (1256).
-	String	PCAN_BR_NOM_TSEG2	"nom_tseg2"	TSEG2 segment for nominal bit in time quar (1128).
	String	PCAN_BR_NOM_SJW	"nom_sjw"	Synchroniza Jump Width nominal bit in time quar (1128).

Data Bit rate parameters:

	Туре	Constant	Value	Descriptio
Ξ	String	PCAN_BR_DATA_BRP	"data_brp"	Clock prese for fast data time quantu (11024).
=	String	PCAN_BR_DATA_TSEG1	"data_tseg1"	TSEG1 segment fo fast data bi rate in time quanta (1
н	String	PCAN_BR_DATA_TSEG2	"data_tseg2"	TSEG2 segment fo fast data bi rate in time quanta (1

=	String	PCAN_BR_DATA_SJW	"data_sjw"	Synchroniz Jump Widtł fast data bir rate in time quanta (1
---	--------	------------------	------------	--

🗉 Remarks

These definitions are constants 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 (PCAN_BR_DATA_* and PCAN_BR_NOM_*, or only PCAN_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	132
data_tseg2	116
data_sjw	116

Python:

The definitions of these values have the following form: <u>TPCANBitrateFD</u>(*string*) where *string* is the value contained in the "value" column. e.g. *PCAN_BR_CLOCK* is defined as <u>TPCANBitrateFD</u>("f_clock").

∃ See Also

CAN_InitializeFD (class-method: InitializeFD)

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

🔒 Home

$\langle \rangle$

Additional Information

PCAN is the platform for PCAN-Basic. In the following topics there is an overview of PCAN and the fundamental practice with the interface DLL CanApi4 (PCAN-API)

In this Chapter

Topics	Description
<u>PCAN</u> <u>Fundamentals</u>	This section contains an introduction to PCAN.
PCAN-Light	This section contains information about the previous version of the PCAN-Basic.
PCAN-API	This section compares the PCAN-Basic and CanApi4 interfaces, with a function description of CanApi4.
Error Frames	This section contains information about CAN Error frames.
Log File Generation	This section contains information about logging debug data within PCAN-Basic.

Copyright $\ensuremath{\mathbb C}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

Home

PCAN Fundamentals

PCAN is a synonym for PEAK CAN APPLICATIONS and is a flexible system for planning, developing and using Controller Area Networks (CAN). It is a powerful product for both the developer and the enduser.

The PCAN system consists of a collection of Windows Device Drivers. These allow the Real-time connection of Windows applications to all CAN busses that are physically connected to the PC via a PCAN hardware. The interface to the user and the manager of a CANequipped installation are the so-called PCAN clients. With their help process factors are controlled and visualized. The drivers permit the connection of several clients, which are then able to communicate via the CAN busses. Furthermore, several hardware components are supported, which are based on the CAN controller Philips SJA1000.

So-called Nets are available. These specify virtual CAN busses that are extended into the PC. Several clients can connect to a virtual CAN bus. The connection to the outside world (a physical CAN bus) is possible with a hardware interface, e.g. the PCAN-Dongle or the PCAN-ISA card. The following figures give an overview of possible configurations.



Following rules apply to PCAN clients, nets and hardware:

- A PCAN client can be connected to more than one net.
- A net supplies several PCAN clients.
- A hardware component belongs to not more than one Net.
- A Net can have no hardware.
- If a Client sends a message, it will be transferred to every other Client and to the external CAN bus via the hardware.

- If a message is received over the hardware, it is received by every client. Every client receives only those messages, which pass its acceptance filter.
- Definition of the installed hardware and of the nets. Per hardware several nets may be defined. But only one net can be active.
- Clients connect to a net using the net name.
- Every PCAN client has a transmission queue, where CAN messages to be transmitted are waiting until the individual transmission time. At occurrence of the transmission time they are written into the transmission queue of the PCAN hardware.
- Every hardware contains a receive queue for buffering received CAN messages.

■ See Also

Understanding PCAN-Basic

Copyright © 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation

PCAN-Basic Documentation

🕇 Home

< ^ >

PCAN-Light

PCAN-Light is the previous version of the <u>PCAN-Basic</u>, the small API variant to <u>PCAN-API</u>. It makes a fast and comprehensive working possible with the CAN bus system. The connection between the application program and the PCAN hardware is made by the appropriate device driver. The following illustration gives a short overview.







Following rules apply to PCAN-Light client (application software) and PCAN hardware:

• A client is in each case assigned to a hardware at each time. A

connection with one client to different hardware types is not possible at the same time.

- If a multi-channel hardware is present, each channel is treated like a separate hardware.
- If a Client sends a message, it will be transferred to every other Client and to the external CAN bus via the hardware.
- If a PCAN hardware is connected to a client, then no further client can access the same hardware.
- If a message is received over the hardware, it is received by every client. Every client receives only those messages, which pass its message (acceptance) filter.
- Every PCAN-Light client has a transmission queue, where CAN messages to be transmitted are waiting until the individual transmission time. At occurrence of the transmission time they are written into the transmission queue of the PCAN hardware.
- Every hardware contains a transmission queue for buffering CAN messages to be transmitted.

Schematic process flow of an PCAN-Light application software



- 1. Start: Call the Init function to iniciate the hardware.
- 2. Operate: After a successful start, the message filter could be adapted at your own conception. Furthermore CAN messages can be read and written. The driver of a type of hardware defined

the range of functions and is structured in basic and additional functions.

3. Finish: Call the Close function. This process disconnect the application software from PCAN hardware.

■ See Also

PCAN Fundamentals

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

PCAN-Basic Documentation

Home



Also called CanApi4 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 one 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:

Register and remove functions for nets and hardware.

Function	Description
CloseAll	Disconnects all hardware, nets, and clients.
RegisterHardware	Registers a Not-Plug-and-Play CAN hardware.
RegisterHardwarePCI	Registers a PCI CAN hardware.
RegisterNet	Defines of a PCAN net.
RemoveHardware	Removes and deactivates CAN hardware.

RemoveNet	Removes a PCAN net.
-----------	---------------------

Configuration

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, eg. the size of the receive or transmit buffer.
SetHwParam	Configures a hardware parameter, eg. - the PEAK serial number, - and additional parameters for the PCAN- USB hardware.
SetNetParam	Configures net parameter.

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, eg. - self-receive mode of transmitted messages. - improve the accuracy of the reception filter.

Communication

Functions for the data interchange over the CAN bus.

Function	Description
Read	Reads a number of CAN_*-records from the client's receive queue. Records are CAN messages, error events, and other information.
Write	Writes a number of CAN messages or other commands into the transmit queue of a client.

Information

Functions for the information about clients, nets, drivers, and hardware.

Function	Description
GetClientParam	Retrieves client parameter, eg. - 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.
GetDiagnosticText	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.
GetHardwareStatus	Detects the current status of a CAN hardware.
GetVersionInfo	Reads version and copyright information from the driver.

Copyright $\textcircled{\mbox{\scriptsize c}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send feedback to this documentation

PCAN-Basic Documentation

🕇 Home



Error Frames

Error Frames can be received if:

- the hardware represented by the connected PCAN-Channel supports error frames generation,
- the connected PCAN-Channel configured the parameter PCAN_ALLOW_ERROR_FRAMES using the function <u>CAN_SetValue</u> (class-method: <u>SetValue</u>) to activate the error frames.

The error frame data will be placed into a <u>TPCANMsg</u> or <u>TPCANMsgFD</u> structure, according to the kind of initialization used, and its MSGTYPE field will have the PCAN_MESSAGE_ERRFRAME bit set.

The following list shows the elements of an error frame:

Field	Information from	Contents
ID	ECC register SJA1000 (Error Code Capture), bits 6 and 7 (ERRC0, ERRC1)	 0: The message only transports updated Error Counter values 1: Bit error 2: Form error 4: Stuff error 8: Other type of error
DATA [0]	ECC register, bit 5 (DIR)	0: Error has occured during transmission 1: Error has occured during reception
DATA [1]	ECC register, bits 0 to 4 (SEG0 to SEG4)	Current position of the bit stream processor: 2: ID.28 to ID.21 3: Start of frame 4: Bit SRTR 5: Bit IDE 6: ID.20 to ID.18

		 7: ID.17 to ID.13 8: CRC Sequence 9: Reserved Bit 0 10: Data Field 11: Data Length Code 12: Bit RTR 13: Reserved Bit 1 14: ID.4 to ID.0 15: ID.12 to ID.5 17: Active Error Flag 18: Intermission 19: Tolerate Dominant Bits 23: Error Delimiter 24: CRC Delimiter 25: Acknowledge Slot 26: End of Frame 27: Acknowledge Limiter 28: Overload Flag
DATA	RX Error Counter	Current value of the Receive
[2]	Register (RXERR)	Error counter
DATA	TX Error Counter	Current value of the Transmit
[3]	Register (TXERR)	Error counter

Remarks

For further information about this, see Philips Data Sheet "SJA1000 Stand-alone CAN controller".

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

Send $\underline{\mathsf{feedback}}$ to this documentation

Home
 Hom

Log File Generation

In order to support debugging of problems, that can arise during CAN communication, PCAN-Basic can generate a log file, containing a protocol of all API function calls. There are two different ways to configure and activate this logging functionality:

- Using the <u>API</u>.
- Using the <u>Windows Registry</u>.

Log configuration using API

In order to configure the Log functionality, PCAN-Basic API provides 3 parameters that can be configured with the function <u>CAN_SetValue</u> (class-method: <u>SetValue</u>). These parameters are:

- <u>PCAN_LOG_LOCATION</u>, to set the location of the log file.
- <u>PCAN_LOG_CONFIGURE</u>, to configure the content of the log file.
- <u>PCAN_LOG_STATUS</u>, to enable/disable logging.

Example:

Within the following example, the PCAN-Basic is configured to log all possible information (LOG_FUNCTION_ALL = 0xFFFF), as well as to store the log file on the desktop of an user called "admin".

C++:

```
TPCANStatus result;
char strMsg[MAX_PATH];
// Configures the data in the log file.
//
iBuffer = LOG_FUNCTION_ALL;
result = CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_CONFI
if(result != PCAN_ERROR_OK)
```

< ^ >

```
ł
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
// Configures the path of the log file.
11
char *buffer = "C:\\users\\admin\\desktop";
result = <u>CAN SetValue</u>(PCAN_NONEBUS, PCAN_LOG_LOCAT
if(result != PCAN_ERROR_OK)
{
    CAN_GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
// Configures the status of the log file
11
int iBuffer = PCAN_PARAMETER_ON;
result = <u>CAN SetValue</u>(PCAN_NONEBUS, PCAN_LOG_STATU
if(result != PCAN ERROR OK)
{
    CAN GetErrorText(result, 0, strMsg);
    MessageBox(strMsg);
}
```

Log configuration using Windows Registry

In order to enable the log file generation, the following registry key must be created:

```
HKEY_CURRENT_USER\SOFTWARE\PEAK-System\PCAN-Basic\
```

The existence of this key is analogous to use the function <u>CAN_SetValue</u> (class-method: <u>SetValue</u>) to set the parameter <u>PCAN_LOG_STATUS</u> to "on". If this key is not present, then no log file is generated.

Configuration:

If no further configuration is made, then the default values for <u>PCAN_LOG_LOCATION</u>, and <u>PCAN_LOG_CONFIGURE</u> are used. In order to configure the location and content of the log file, two registry values are used:

Flags: This is a DWORD value, that represents a logical **OR** operation between the values <u>LOG_FUNCTION_*</u> that are wanted to be included within the logging data. The value <u>LOG_FUNCTION_ALL</u> causes logging all possible information.

Path: This is a String value, that represents the path to a folder in the computer, where the log file will be created.

Example:

Within the following example, the PCAN-Basic is configured to log function entries (LOG_FUNCTION_ENTRY = 1), function parameters (LOG_FUNCTION_PARAMETERS = 2), and function outs (LOG_FUNCTION_LEAVE = 4), as well as to store the log file on the desktop of an user called "admin".

```
[HKEY_CURRENT_USER\SOFTWARE\PEAK-System\PCAN-Basic
"Flags"=dword:00000007
"Path"="C:\\Users\\admin\\desktop"
```

Remarks:

The registry key should be deleted (or renamed) after a debug session is done. If the key is leaved, all PCAN-Basic applications running under the same user account will remain writing data to their log files, generating in this way huge text files that consume hard-disk space unnecessarily.

Send feedback to this documentation

Copyright $\ensuremath{\mathbb{C}}$ 2017. PEAK-System Technik GmbH. All rights reserved.

PCAN-Basic Documentation

🕇 Home

 $\langle \rangle$

PCAN-Basic Documentation



Welcome to the documentation of PCAN-Basic, the new small Version of the PCAN-API from PEAK-System.

Introduction - DLL API Reference - Additional Information

Last Update: 31.07.2017

Copyright @ 2017. PEAK-System Technik GmbH. All rights reserved. Send <u>feedback</u> to this documentation