8.11.2021

PEAK-System Technik GmbH

PCAN - Parameters

PCAN-Basic Parameters Description

Index

Introduction

The number of configurable parameters within the PCAN-Basic has been growing recently. It is sometimes difficult to figure out when you need to use a specific parameter or how it works. Additionally, there are some parameters that support a pre-initialized behavior. What is the intention of those parameters? We will try to answer questions like this, and more, in this documentation.

Take into consideration that this documentation is based on the PCAN-Basic API, version **4.6.0**. Please check your API version and, if necessary, update it.

Please Note: Not all parameters mentioned in this documentation are applicable to all Peak-Devices that can be used with PCAN-Basic.

Due to the universal nature of the API some parameters are only usable on certain items of our product-line. Please refer to the user-manual of your device to see if the feature the parameter refers to is supported.

The changes history that the API has experienced since its first release can be found in our website at http://www.peak-system.com/PCAN-Basic.126.0.html.

If you want to easily keep informed about our products, for example new releases of our free API PCAN-Basic, you can subscribe to our RSS-Feed or you can visit our support website at http://www.peak-system.com/Support.55.0.html.

Supported PCAN-Parameters

PCAN-Basic currently supports 28 parameters that can be read/configured using the functions CAN_GetValue/CAN_SetValue. Not all parameters can be configured because some of them are **read-only** parameters. Following you will find a list with the parameters and their associated value:

Parameters Groups

In order to delimit the purpose of the different parameters, they are arranged in 5 groups as:

Parameters for “Hardware Identification”:

- PCAN_CHANNEL_CONDITION
- PCAN_DEVICE_ID
- PCAN_HARDWARE_NAME
- PCAN_CONTROLLER_NUMBER
- PCAN_CHANNEL_IDENTIFYING
- PCAN_IP_ADDRESS
- PCAN_AVAILABLE_CHANNELS
- PCAN_DEVICE_PART_NUMBER

Parameters for “Informational” purposes:

- PCAN_API_VERSION
- PCAN_CHANNEL_VERSION
- PCAN_CHANNEL_FEATURES
- PCAN_BITRATE_INFO
- PCAN_BITRATE_INFO_FD
- PCAN_BUSSPEED_NOMINAL
- PCAN_BUSSPEED_DATA
- PCAN_LAN_SERVICE_STATUS
- PCAN_FIRMWARE_VERSION
- PCAN_AVAILABLE_CHANNELS_COUNT

Parameters for “Influencing Behavior”:

- PCAN_5VOLTS_POWER
- PCAN_BUSOFF_AUTORESET
- PCAN_LISTEN_ONLY
- PCAN_BITRATE_ADAPTING
- PCAN_INTERFRAME_DELAY

Parameters for “Data Reading and Flow Control”:

- PCAN_RECEIVE_EVENT

- PCAN_MESSAGE_FILTER
- PCAN_RECEIVE_STATUS
- PCAN_ALLOW_STATUS_FRAMES
- PCAN_ALLOW_RTR_FRAMES
- PCAN_ALLOW_ERROR_FRAMES
- PCAN_ALLOW_ECHO_FRAMES
- PCAN_ACCEPTANCE_FILTER_11BIT
- PCAN_ACCEPTANCE_FILTER_29BIT

Parameters for “Logging and Debugging”:

- PCAN_LOG_LOCATION
- PCAN_LOG_STATUS
- PCAN_LOG_CONFIGURE
- PCAN_LOG_TEXT

Parameters for “CAN Data Recording (Tracing)”:

- PCAN_TRACE_LOCATION
- PCAN_TRACE_STATUS
- PCAN_TRACE_SIZE
- PCAN_TRACE_CONFIGURE

Parameters for “electronic circuits (I/O pins)”:

- PCAN_IO_DIGITAL_CONFIGURATION
- PCAN_IO_DIGITAL_VALUE
- PCAN_IO_DIGITAL_SET
- PCAN_IO_DIGITAL_CLEAR
- PCAN_IO_ANALOG_VALUE

Pre-Initialized Parameters

The parameter configuration within the PCAN-Basic API, except of the parameters grouped as “Logging and Debugging” (these are not tied to a channel in particular), is allowed ***after*** a channel is successfully initialized. Nevertheless, there are some cases in which it is needed to do some configuration even before a channel is initialized. The following parameters can be configured on a channel ***before*** it is initialized:

- PCAN_RECEIVE_STATUS
- PCAN_LISTEN_ONLY
- PCAN_BITRATE_ADAPTING

Identifying a Hardware

First, consider that the first identification takes place when selecting the PCAN-Channel to be used. The channel's name already identifies the bus to use.

The name above tells the API the PCAN hardware to connect to, which kind of bus it uses (***USB***), and that it is the *first* (***1***) hardware ***registered*** in a system. PCAN-Basic allows connecting following interfaces:

- USB: Universal Serial Bus. Up to 16 channels.
- PCI: Peripheral Component Interconnect (including ExpressCard hardware). Up to 16 channels.
- PCC: PC-Card (PCMCIA), Personal Computer Memory Card. Up to 2 channels.
- LAN: Virtual PCAN-Gateway connections. Up to 16 channels.
- DNG: Parallel port Dongle. Up to 1 channel.
- ISA: Industry Standard Architecture. Up to 8 channels.

Note that the way of how hardware is registered in a system depends on its controller driver and on the system itself. When several devices of the same kind are installed on a system (USB for example), by default it is not guaranteed that connecting to PCAN_USBBUS1 after a system restart will still connect to the same hardware.

Therefore, parameters are used to help on the detection of the right hardware. The following parameters are used to identify the physical hardware to connect, for example when several devices are available for connection.

PCAN_CHANNEL_CONDITION

This parameter is used to identify the state of use of a PCAN-Channel by returning a flag value. For example, a connection is only possible when a PCAN-Channel is available, which means:

- It is valid: The PCAN-Channel is one of the listed in the section "Supported by" below.
- It is connectable: The PCAN-Channel is not initialized, or it is currently used by a PCAN-View application.

Availability

Available since version 1.0.0. Nevertheless, usability improved significantly since version 1.0.4, due to bugfixes. The behavior of this parameter was modified with the version 4.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The condition of a PCAN-Channel can be one of the following defined values:

Defined Value	Description
PCAN_CHANNEL_UNAVAILABLE	The channel is not attached/accessible.
PCAN_CHANNEL_AVAILABLE	The channel can be used.
PCAN_CHANNEL_OCCUPIED	The channel was already initialized.
PCAN_CHANNEL_PCANVIEW	The channel is being used by a PCAN-View, but it can be initialized.

Note that the last value was introduced with the PCAN-Basic version 4.0.0. This value is an OR-Operation between PCAN_CHANNEL_AVAILABLE and PCAN_CHANNEL_OCCUPIED. For this reason, all software checking only for availability (result equal to PCAN_CHANNEL_AVAILABLE) will miss to recognize channels that are being connected by PCAN-View applications.

Default Value

Does not apply.

Initialization Status

Not relevant since this parameter is used to ask the status of a PCAN-Channel.

When to Use

It can be used when the availability status of a channel registered in a system at a given time must be known.

Application – Example of Use

Imagine you want to create a Test-Application that connects to a PCAN-PCI device. In order to allow the user to decide which PCAN-Channel should be used for data transmission, you have to list all available PCAN-PCI Channels. Using this parameter, you can filter out the channels that are occupied or unavailable (it is assumed that the PC has 4 PCI channels):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_PCIBUS1, PCAN_PCIBUS2, PCAN_PCIBUS3, PCAN_PCIBUS4 };
DWORD condition;

for (int i = 0; i < 4; i++)
{
    if (CAN_GetValue(channelsToCheck[i], PCAN_CHANNEL_CONDITION, &condition, 4) == PCAN_ERROR_OK)
        if ((condition & PCAN_CHANNEL_AVAILABLE) == PCAN_CHANNEL_AVAILABLE)
            printf("The channel-handle 0x%X is AVAILABLE\n", channelsToCheck[i]);
}
```

Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_PCIBUS1, PCANBasic.PCAN_PCIBUS2, PCANBasic.PCAN_PCIBUS3,
PCANBasic.PCAN_PCIBUS4 };
uint condition;

for (int i=0; i < 4; i++)
{
    if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_CHANNEL_CONDITION, out condition, 4) ==
TPCANStatus.PCAN_ERROR_OK)
```

```
        if ((condition & PCANBasic.PCAN_CHANNEL_AVAILABLE) == PCANBasic.PCAN_CHANNEL_AVAILABLE)
            Console.WriteLine("The channel-handle 0x{0:X} is AVAILABLE", channelsToCheck[i]);
}
```

PCAN_CHANNEL_IDENTIFYING

This parameter is used to physically identify an USB-based PCAN-Channel being used. The identification is done using the status LED of the USB devices. At the moment PEAK-System offers USB devices of three different generations:

- First Generation: PCAN-USB, PCAN-Hub.
- Second Generation: PCAN-USB Pro, PCAN-USB2
- Third Generation: PCAN-USB Classic, PCAN-USB FD, PCAN-USB Pro FD,

According with the hardware used, the blinking of the LED is different in color and blink rate:

- First Generation: Blink color is RED, and the blink rate is about 300 milliseconds.
- Second Generation: Blink color is ORANGE, and the blink rate is about 250 milliseconds.
- Third Generation: Blink color is ORANGE, and the blink rate is about 250 milliseconds.

Availability

It is available since version 1.3.0.

Supported By

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter represents a procedure used for identification that can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The identifying procedure is set to OFF.
PCAN_PARAMETER_ON	The identifying procedure is set to ON.

Note that only one channel can be activated at a time. In order to switch on the identifying procedure in another channel, the previous one must be first switched off.

Default Value

The default state of this identification procedure is off (PCAN_PARAMETER_OFF). After switching it on, the LED of an USB device stays blinking until it is expressly turned off.

Initialization Status

This parameter can be used with both, initialized and uninitialized PCAN-Channels. **Note** that the activation of this identification procedure doesn't affect any communication that can occur on the device while it is being identified.

When to Use

It can be used when an application can connect to several USB devices and it is not clear which (physical) channel must be used in a determined time, for example, before establishing a connection to a channel. It is also useful in application that communicate with several USB devices at the same time and for long periods of time (or applications used for several people), to check with channels are being used in a determined time.

Application – Example of Use

Let's say you have an application communicating with several USB devices. This application is running on a computer on which the order of the devices representing each PCAN-Channel can vary (the computer reboots automatically within a given period, the physical CAN networks are eventually swapped, etc.). Now you're using the application and you need to physically disconnect a device, but you don't know which PCAN-Channel is associated to it, and you don't want to disturb the other channels. You can write a small application that just turns the identifying procedure on a given channel on, so that you can see which device is the one you are looking for (it is assumed that the PC has 3 USB channels):

Native (C++)

```
TPCANHandle channelsToIdentify[] = { PCAN_USBBUS1, PCAN_USBBUS2, PCAN_USBBUS3 };
DWORD activate;

for (int i = 0; i < 3; i++)
{
    activate = PCAN_PARAMETER_ON;
    if (CAN_SetValue(channelsToIdentify[i], PCAN_CHANNEL_IDENTIFYING, &activate, 4) == PCAN_ERROR_OK)
    {
        printf("The channel with handle 0x%X is now BLINKING. ", channelsToIdentify[i]);
        system("PAUSE");
        activate = PCAN_PARAMETER_OFF;
        CAN_SetValue(channelsToIdentify[i], PCAN_CHANNEL_IDENTIFYING, &activate, 4);
    }
}
```

Managed (C#)

```
ushort[] channelsToIdentify = { PCANBasic.PCAN_USBBUS1, PCANBasic.PCAN_USBBUS2, PCANBasic.PCAN_USBBUS3 };
uint activate;

for (int i = 0; i < 3; i++)
{
    activate = PCANBasic.PCAN_PARAMETER_ON;
    if (PCANBasic.SetValue(channelsToIdentify[i], TPCANParameter.PCAN_CHANNEL_IDENTIFYING, ref activate, 4)
== TPCANStatus.PCAN_ERROR_OK)
    {
        Console.Write("The channel with handle 0x{0:X} is now BLINKING. ", channelsToIdentify[i]);
        Console.WriteLine("Press any Key to continue . . .");
        Console.ReadKey();
        activate = PCANBasic.PCAN_PARAMETER_OFF;
        PCANBasic.SetValue(channelsToIdentify[i], TPCANParameter.PCAN_CHANNEL_IDENTIFYING, ref activate, 4);
    }
}
```

PCAN_DEVICE_ID

This parameter is used to distinguish between 2 or more devices of the same kind connected to a computer simultaneously. A device identifier is a persistent value stored in the flash memory of each device, i.e., the value is not lost after disconnecting the hardware.

Note that the devices can have the same identifier. It is up to the user to guarantee that the devices being used are configured with different identifiers, so that a differentiation through this value can work.

This parameter was previously called PCAN_DEVICE_NUMBER. It was renamed to PCAN_DEVICE_ID starting with PCAN-Basic version 4.4.0. PCAN_DEVICE_NUMBER is still present for backward compatibility reasons, but it is marked as **deprecated**. Users should use PCAN_DEVICE_ID instead.

Availability

It is available since version 1.0.0. as PCAN_DEVICE_NUMBER.
It can be read without initialization since version 4.4.0.

Supported By

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Notes:
PCAN-PCI: Only FPGA based devices. Requires a device driver version equal to or greater than 4.2.0.
PCAN-LAN: Only devices with a firmware version equal to or greater than 2.8.2. Requires a device driver version equal to or greater than 4.2.0.

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

According with the firmware version of the PCAN-USB device, this value can have a resolution of a byte (range [0...255]) or a double-word (range [0...4294967295]).

Default Value

If this parameter was never set before, the value is the maximum value possible for the used resolution which is 255 (FFh), or 429496729 (FFFFFFFFh).

Initialization Status

Get: It can be read on initialized or uninitialized PCAN-Channels.
Set: It can be set on initialized PCAN-Channels only.

When to Use

It can be used when it is needed to differentiate between PCAN-USB devices connected to the same system at a given time.

Application – Example of Use

Let's say you want to write an application that reads data from one CAN-BUS and replies to a second CAN-BUS (a.k.a. Gateway application). For this you could have one PCAN-USB device connected to each CAN-BUS. You could set the device number of both PCAN-USBs so that you know which bus is used for writing (for example, **number 1** for the "to write to" bus), and which bus is used for reading (for example, **number 2** for the "to read from" bus). Using this parameter, you would be able to know if both channels are available and which device is used for sending and which one for writing (it is assumed that the PC has 2 USB devices connected):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_USBBUS1, PCAN_USBBUS2 };
DWORD deviceId;
TPCANHandle readChannel, writeChannel;

readChannel = writeChannel = PCAN_NONEBUS;
for (int i = 0; i < 2; i++)
{
    if (CAN_GetValue(channelsToCheck[i], PCAN_DEVICE_ID, &deviceId, 4) == PCAN_ERROR_OK)
    {
        if (deviceId == 1)
        {
            writeChannel = channelsToCheck[i];
            printf("The channel for writing (handle 0x%X) was found.\n", channelsToCheck[i]);
        }
        if (deviceId == 2)
        {
            readChannel = channelsToCheck[i];
            printf("The channel for reading (handle 0x%X) was found.\n", channelsToCheck[i]);
        }
    }
}

if ((readChannel != PCAN_NONEBUS) && (writeChannel != PCAN_NONEBUS))
{
    printf("Both channels were found. Starting to work . . .");
    // Do work . . .
}
else
    printf("Error! Not all needed channels were found. Terminating . . .");
```

Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_USBBUS1, PCANBasic.PCAN_USBBUS2 };
uint deviceId;
ushort readChannel, writeChannel;

readChannel = writeChannel = PCANBasic.PCAN_NONEBUS;
for (int i = 0; i < 2; i++)
{
    if (PCANBasic.GetValue(channelsToCheck[i],  TPCANParameter.PCAN_DEVICE_ID, out deviceId, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        if (deviceId == 1)
        {
            writeChannel = channelsToCheck[i];
            Console.WriteLine("The channel for writing (handle 0x{0:X}) was found.", channelsToCheck[i]);
        }
        if (deviceId == 2)
        {
            readChannel = channelsToCheck[i];
            Console.WriteLine("The channel for reading (handle 0x{0:X}) was found.", channelsToCheck[i]);
        }
    }
}

if ((readChannel != PCANBasic.PCAN_NONEBUS) && (writeChannel != PCANBasic.PCAN_NONEBUS))
{
    Console.WriteLine("Both channels were found. Starting to work . . .");
    // Do work . . .
}
else
    Console.WriteLine("Error! Not all needed channels were found. Terminating . . .");
```

PCAN_HARDWARE_NAME

This parameter is used to retrieve a description text from the hardware represented by a PCAN channel. This text allows the recognition of device's models that use the same interface, for example USB. A normal PCAN USB adaptor would return "PCAN-USB" while the new dual CAN/LIN FD channel adaptor would return "PCAN-USB Pro FD".

Availability

It is available since version 1.0.6.

It can be read without initialization since version 4.4.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The value is a null-terminated string which contains the name of the hardware specified by the given PCAN channel. This string has a maximum length of 32 bytes (null-termination character included).

According with the hardware model represented by the current PCAN-Channel, the following text can be returned:

Hardware Name Value	Interface	Hardware Description
PEAK ISA-CAN	PCAN-ISA	PCAN-ISA, PCAN-PC/104
PEAK ISA-CAN SJA	PCAN-ISA	PCAN-ISA, PCAN-PC/104 with a SJA1000
PEAK Dongle-CAN	PCAN-DNG	PCAN-Dongle with an 82C200
PEAK Dongle-CAN EPP	PCAN-DNG	PCAN-Dongle with an 82C200, using EPP mode
PEAK Dongle-CAN SJA	PCAN-DNG	PCAN-Dongle with a SJA1000
PEAK Dongle-CAN SJA EPP	PCAN-DNG	PCAN-Dongle with a SJA1000, using EPP mode
PEAK Dongle-Pro	PCAN-DNG	PCAN-Dongle Pro
PEAK Dongle-Pro EPP	PCAN-DNG	PCAN-Dongle Pro in EPP mode
PCAN-PCI	PCAN-PCI	CAN Interface for PCI
PCAN-PCI Express	PCAN-PCI	CAN Interface for PCI Express
PCAN-PCI Express FD	PCAN-PCI	CAN and CAN FD Interface for PCI Express
PCAN-cPCI	PCAN-PCI	CAN Interface for CompactPCI
PCAN-MiniPCI	PCAN-PCI	CAN Interface for Mini PCI
PCAN-miniPCIe	PCAN-PCI	CAN Interface for PCI Express Mini (PCIe)
PCAN-miniPCIe FD	PCAN-PCI	CAN and CAN FD Interface for PCI Express Mini (PCIe)
PCAN-M.2	PCAN-PCI	CAN and CAN FD Interface for M.2 (PCIe)
PCAN-Chip PCIe FD	PCAN-PCI	Chip Solutions for the CAN FD Connection to PCI Express
PCAN-PCI/104-Plus	PCAN-PCI	CAN Interface for PC/104-Plus
PCAN-PCI/104-Plus Quad	PCAN-PCI	Four-Channel CAN Interface for PC/104-Plus
PCAN-PCI/104-Express	PCAN-PCI	CAN Interface for PCI/104-Express
PCAN-PC/104-Express FD	PCAN-PCI	CAN and CAN FD Interface for PCI/104-Express
PCAN-ExpressCard	PCAN-PCI	PCAN-ExpressCard
PCAN-ExpressCard 34	PCAN-PCI	PCAN-ExpressCard 34
PCAN-USB	PCAN-USB	PCAN-USB Adapter, PCAN-USB Hub
PCAN-USB FD	PCAN-USB	PCAN-USB FD Adapter, PCAN-USB
PCAN-USB Pro	PCAN-USB	PCAN-USB Pro dual CAN/LIN
PCAN-USB Pro FD	PCAN-USB	PCAN-USB Pro FD dual CAN/LIN FD
PCAN-USB Hub	PCAN-USB	All-in-one USB Adapter for communication through USB, CAN and RS-232

PCAN-USB X6	PCAN-USB	6-Channel CAN and CAN FD Interface for High-Speed USB 2.0
PCAN-Chip USB	PCAN-USB	Stamp Module for the Implementation of CAN FD to USB Connections
PCAN-PCCARD-CAN	PCAN-PCC	PCAN-PC Card
PCAN-Ethernet Gateway DR	PCAN-LAN	PCAN-Gateway wired for mounting on a DIN rail
PCAN-Wireless Gateway DR	PCAN-LAN	PCAN-Gateway wireless for mounting on a DIN rail
PCAN-Wireless Gateway	PCAN-LAN	PCAN-Gateway wireless with D-Sub connector
PCAN-Wireless Automotive Gateway	PCAN-LAN	PCAN-Gateway wireless with automotive connector

Default Value

Does not apply.

Initialization Status

Get: It can be read on initialized or uninitialized PCAN-Channels.

When to Use

It can be used when it is needed to differentiate between several hardware models using the same interface (e.g., PCAN-PCI, PCAN-ExpressCard)

Application – Example of Use

Consider the following scenario: You want to develop a Diagnostic-Application using a normal PCAN-USB device for data transmission. The program should run on computers that have per default a PCAN-USB Pro attached, intended to be used from other programs (for ECU controlling, Gateway configuration purpose, etc.), and therefore shouldn't be occupied. This means that the system will have 3 PCAN channels registered (PCAN_USBBUS1 to PCAN_USBBUS3). Since the diagnostic network will be always plugged-in to your PCAN-USB, your application must be sure to connect the single channel and not one of the PCAN-USB Pro channels. Using this parameter, you would be able to identify which PCAN-Channel represents a PCAN-USB and which one a PCAN-USB Pro (it is assumed that the PC has 3 USB channels):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_USBBUS1, PCAN_USBBUS2, PCAN_USBBUS3 };
char hardwareName[MAX_LENGTH_HARDWARE_NAME] = { 0 };
TPCANHandle debugBus = PCAN_NONEBUS;

for (int i = 0; i < 3; i++)
{
    if (CAN_GetValue(channelsToCheck[i], PCAN_HARDWARE_NAME, hardwareName, MAX_LENGTH_HARDWARE_NAME) ==
PCAN_ERROR_OK)
    {
        if (strcmp(hardwareName, "PCAN-USB") == 0)
        {
            debugBus = channelsToCheck[i];
            break;
        }
    }
}

if (debugBus != PCAN_NONEBUS)
{
    printf("Single PCAN-USB for debugging (handle 0x%X) found. Starting to work . . .", debugBus);
    // Do work . . .
}
else
    printf("Error! Single PCAN-USB Channel was not found. Terminating . . .");
```

Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_USBBUS1, PCANBasic.PCAN_USBBUS2, PCANBasic.PCAN_USBBUS3 };
StringBuilder hardwareName = new StringBuilder(PCANBasic.MAX_LENGTH_HARDWARE_NAME);
TPCANHandle debugBus = PCANBasic.PCAN_NONEBUS;

for (int i = 0; i < 3; i++)
{
    if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_HARDWARE_NAME, hardwareName,
PCANBasic.MAX_LENGTH_HARDWARE_NAME) == TPCANStatus.PCAN_ERROR_OK)
    {
        if (hardwareName.ToString().Equals("PCAN-USB"))
        {
            debugBus = channelsToCheck[i];
            break;
        }
    }
}

if (debugBus != PCANBasic.PCAN_NONEBUS)
{
    Console.WriteLine("Single PCAN-USB for debugging (0x{0:X}) found. Starting to work . . .", debugBus);
    // Do work . . .
}
else
    Console.WriteLine("Error! Single PCAN-USB Channel was not found. Terminating . . .");
```


PCAN_CONTROLLER_NUMBER

This parameter is used to identify the physical CAN channel index of a multichannel CAN hardware (PCAN-PCI, PCAN-USB Pro, PCAN-LAN, etc.). This index is zero-based, so that the first channel on a device is 0, the second 1, and so on.

Availability

It is available since version 1.2.0.

It can be read without initialization since version 4.4.0.

Supported By

PCAN-DNG (Channel PCAN_DNGBUS1).

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).

PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

A number in the range [0…**n-1**], where **n** is the number of physical channels on the device being used. The correspondence between an index number and the CAN channel description on the hardware etiquette is:

Channel Index	Channel Label
0	CAN 1
1	CAN 2
n-1	CAN n

Default Value

Does not apply.

Initialization Status

Get: It can be read on initialized or uninitialized PCAN-Channels.

Set: It can be set on initialized PCAN-Channels only.

When to Use

It can be used to determine which physical channel of a multichannel PCAN device has to be connected.

Application – Example of Use

The easy case: let's say you want to write an application that should work only with the second channel of any PCAN-USB device. You could just ask for the PCAN_CONTROLLER_NUMBER on each available USB channel until you find a channel with a controller number equals to "1" (it is assumed that the PC has 4 USB channels):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_USBBUS1, PCAN_USBBUS2, PCAN_USBBUS3, PCAN_USBBUS4 };
DWORD controllerNumber;
TPCANHandle canChannel2 = PCAN_NONEBUS;

for (int i = 0; i < 4; i++)
{
    if (CAN_GetValue(channelsToCheck[i], PCAN_CONTROLLER_NUMBER, &controllerNumber, 4) == PCAN_ERROR_OK)
    {
        if (controllerNumber == 1)
        {
            canChannel2 = channelsToCheck[i];
            break;
        }
    }
}

if (canChannel2 != PCAN_NONEBUS)
{
    printf("Second USB CAN-controller found (handle 0x%X). Starting to work . . .", canChannel2);
    // Do work . . .
}
else
    printf("Error! Second USB CAN-controller was not found. Terminating . . .");
```

Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_USBBUS1, PCANBasic.PCAN_USBBUS2, PCANBasic.PCAN_USBBUS3,
PCANBasic.PCAN_USBBUS4 };
uint controllerNumber;
ushort canChannel2 = PCANBasic.PCAN_NONEBUS;

for (int i = 0; i < 4; i++)
{
    if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_CONTROLLER_NUMBER, out controllerNumber,
4) == TPCANStatus.PCAN_ERROR_OK)
    {
        if (controllerNumber == 1)
        {
            canChannel2 = channelsToCheck[i];
            break;
        }
    }
}

if (canChannel2 != PCANBasic.PCAN_NONEBUS)
{
    Console.WriteLine("Second USB CAN-controller found (handle 0x{0:X}). Starting to work . . .",
canChannel2);
    // Do work . . .
}
else
    Console.WriteLine("Error! Second USB CAN-controller was not found. Terminating . . .");
```

The complicated case: you want to use the second channel of a specific PCAN-USB Pro FD hardware, device number 7 for example, and there exists the possibility to have several multi-channels devices attached to the computer at a time. Using the parameter

PCAN_HARDWARE_NAME lets you find any PCAN-USB Pro connected. Using the parameter PCAN_DEVICE_ID lets you find the right Device (number 7). Finally, using the PCAN_CONTROLLER_NUMBER lets you find the right CAN channel to use (it is assumed that the PC has 4 USB channels):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_USBBUS1, PCAN_USBBUS2, PCAN_USBBUS3, PCAN_USBBUS4 };
char hardwareName[MAX_LENGTH_HARDWARE_NAME] = { 0 };
DWORD deviceId;
DWORD controllerNumber;
TPCANHandle canChannel2 = PCAN_NONEBUS;

for (int i = 0; i < 4; i++)
{
    if (CAN_GetValue(channelsToCheck[i], PCAN_HARDWARE_NAME, hardwareName, MAX_LENGTH_HARDWARE_NAME) ==
PCAN_ERROR_OK)
    {
        if (strcmp(hardwareName, "PCAN-USB Pro FD") != 0)
            continue;

        if (CAN_GetValue(channelsToCheck[i], PCAN_DEVICE_ID, &deviceId, 4) == PCAN_ERROR_OK)
        {
            if (deviceId != 7)
                continue;

            if (CAN_GetValue(channelsToCheck[i], PCAN_CONTROLLER_NUMBER, &controllerNumber, 4) ==
PCAN_ERROR_OK)
            {
                if (controllerNumber == 1)
                {
                    canChannel2 = channelsToCheck[i];
                    break;
                }
            }
        }
    }
}

if (canChannel2 != PCAN_NONEBUS)
{
    printf("Second USB CAN-controller found (handle 0x%X). Starting to work . . .", canChannel2);
    // Do work . . .
}
else
    printf("Error! Second USB CAN-controller was not found. Terminating . . .");
```

Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_USBBUS1, PCANBasic.PCAN_USBBUS2, PCANBasic.PCAN_USBBUS3,
PCANBasic.PCAN_USBBUS4 };
uint controllerNumber;
uint deviceId;
StringBuilder hardwareName = new StringBuilder(PCANBasic.MAX_LENGTH_HARDWARE_NAME);
ushort canChannel2 = PCANBasic.PCAN_NONEBUS;

for (int i = 0; i < 4; i++)
{
    if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_HARDWARE_NAME, hardwareName,
PCANBasic.MAX_LENGTH_HARDWARE_NAME) == TPCANStatus.PCAN_ERROR_OK)
    {
        if(hardwareName.ToString().CompareTo("PCAN-USB Pro FD") != 0)
            continue;

        if (PCANBasic.GetValue(channelsToCheck[i],  TPCANParameter.PCAN_DEVICE_ID, out deviceId, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            if (deviceId != 7)
                continue;

            if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_CONTROLLER_NUMBER, out
controllerNumber, 4) == TPCANStatus.PCAN_ERROR_OK)
            {
                if (controllerNumber == 1)
                {
                    canChannel2 = channelsToCheck[i];
                    break;
                }
            }
        }
    }
}
```

```
if (canChannel2 != PCANBasic.PCAN_NONEBUS)
{
    Console.WriteLine("Second USB CAN-controller found (handle 0x{0:X}). Starting to work . . .",
canChannel2);
    // Do work . . .
}
else
    Console.WriteLine("Error! Second USB CAN-controller was not found. Terminating . . .");
```

PCAN_IP_ADDRESS

This parameter applies ONLY to hardware of type PCAN-LAN. It is used to distinguish between 2 or more hardware of this kind connected to a computer simultaneously. An IP address is the configured network address on a PCAN-Gateway device, i.e., the address used to communicate with a PCAN-Gateway device through the network (LAN/WAN).

The IP address identifies a device effectively because it is not allowed to have the same IP address twice within a network, at the same time (address conflict).

Availability

Available since version 4.0.0.

Supported By

PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

Since the format used for the IP address is IPv4, possible values are string representing 4 number sections separated by '.' which are in the range [0…255]. Example of an IP address is: "192.168.0.1".

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when it is needed to differentiate between PCAN-LAN devices connected to the same system at a given time, or just to use the IP address to get more information about a remote PCAN-Gateway device.

Application – Example of Use

Let's say you have several PCAN-LAN channels available to connect to, and each of them represents a different PCAN-Gateway device. You want to observe the CAN data on the remote address 192.168.1.95. Asking PCAN-Basic for channel availability will return only a list of channels like "PCAN_LANBUS1, PCAN_LANBUS2, PCAN_LANBUS3, …,". Asking the IP address on each channel will help you finding the desired device (it is assumed that there are 3 LAN channels available, working with normal CAN protocol):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_LANBUS1, PCAN_LANBUS2, PCAN_LANBUS3 };
char ipBuffer[20] = { 0 };
DWORD adaptToBitrate = PCAN_PARAMETER_ON;
TPCANHandle lanToWatch = PCAN_NONEBUS;

for (int i = 0; i < 3; i++)
{
    // Bitrate cannot be set for LAN channels, it must be adopted
    if (CAN_SetValue(channelsToCheck[i], PCAN_BITRATE_ADAPTING, &adaptToBitrate, 4) == PCAN_ERROR_OK)
    {
        // LAN channels need to be initialized first, before asking their values
        if (CAN_Initialize(channelsToCheck[i], 0) == PCAN_ERROR_CAUTION)
        {
            if (CAN_GetValue(channelsToCheck[i], PCAN_IP_ADDRESS, ipBuffer, 20) == PCAN_ERROR_OK)
            {
                if (strcmp(ipBuffer, "192.168.1.95") == 0)
                {
                    lanToWatch = channelsToCheck[i];
                    break;
                }
            }
            CAN_Uninitialize(channelsToCheck[i]);
        }
    }
}

if (lanToWatch != PCAN_NONEBUS)
{
    printf("LAN channel found (handle 0x%X). Connected, and ready to work . . .", lanToWatch);
    // Do work . . .
}
else
    printf("Error! LAN channel with required IP is not available. Terminating . . .");
```


Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_LANBUS1, PCANBasic.PCAN_LANBUS2, PCANBasic.PCAN_LANBUS3 };
StringBuilder ipBuffer = new StringBuilder(20);
uint adaptToBitrate = PCANBasic.PCAN_PARAMETER_ON;
ushort lanToWatch = PCANBasic.PCAN_NONEBUS;

for (int i = 0; i < 3; i++)
{
    // Bitrate cannot be set for LAN channels, it must be adopted
    if (PCANBasic.SetValue(channelsToCheck[i], TPCANParameter.PCAN_BITRATE_ADAPTING, ref adaptToBitrate, 4)
== TPCANStatus.PCAN_ERROR_OK)
    {
        // LAN channels need to be initialized first, before asking their values
        if (PCANBasic.Initialize(channelsToCheck[i], 0) == TPCANStatus.PCAN_ERROR_CAUTION)
        {
            if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_IP_ADDRESS, ipBuffer, 20) ==
TPCANStatus.PCAN_ERROR_OK)
            {
                if (ipBuffer.ToString().CompareTo("10.1.12.214") == 0)
                {
                    lanToWatch = channelsToCheck[i];
                    break;
                }
            }
            PCANBasic.Uninitialize(channelsToCheck[i]);
        }
    }
}

if (lanToWatch != PCANBasic.PCAN_NONEBUS)
{
    Console.WriteLine("LAN channel found (handle 0x{0:X}). Connected, and ready to work . . .", lanToWatch);
    // Do work . . .
}
else
    Console.WriteLine("Error! LAN channel with required IP is not available. Terminating . . .");
```

PCAN_ATTACHED_CHANNELS

This parameter is used to get information about all existing PCAN channels on a system in a single call, regardless of their current availability (See CHANNEL_CONDITION).

This parameter is closely tied to another, PCAN_ATTACHED_CHANNELS_COUNT. It returns the number of existing channels, which is important for the size calculation of the buffer,

that must be passed to the function CAN_GetValue, when using the parameter PCAN_ATTACHED_CHANNELS.

The size in bytes of this buffer is calculated using the result of PCAN_ATTACHED_CHANNELS_COUNT multiplied by the size of the structure TPCANChannelInformation.

If Python is used, then it is not necessary to calculate the size of the buffer. Since the call to PCANBasic.GetValue in Python returns a tuple as result, the function internally defines a buffer big enough for storing and returning the information of the channels.

Following information is delivered for each available channel:

- *channel_handle*: Contains the PCAN-Channel identification handle, used for API calls (e.g., PCAN_USBBUS1, PCAN_PCIBUS2, etc.).
- *device_type*: Denotes the type of device to which the PCAN-Channel belongs (e.g., PCAN_USB, PCAN_PCI, etc.).
- *controller_number*: Indicates the physical CAN channel index (zero-based) associated to the PCAN channel. This value is the same returned when calling CAN_GetValue with the parameter PCAN_CONTROLLER_NUMBER.
- *device_features*: Contains information about special properties associated to the PCAN-Channel. This value is the same returned when calling CAN_GetValue with the parameter PCAN_CHANNEL_FEATURES.
- *device_name*: Contains the description text from the device to which the PCAN-Channel belongs. This value is the same returned when calling CAN_GetValue with the parameter PCAN_HARDWARE_NAME.
- *device_id*: Represents an identification value stored in the flash memory of the device to which the PCAN-Channel belongs. This value is the same returned when calling CAN_GetValue with the parameter PCAN_DEVICE_ID (previously called PCAN_DEVICE_NUMBER).
- *channel_condition*: Represents the state of use of the PCAN-Channel. This value is the same returned when calling CAN_GetValue with the parameter PCAN_CHANNEL_CONDITION.

Availability

It is available since version 4.4.0.

Supported By

PCAN-NONEBUS: The number of available channels is not tied to any channel, i.e. no specific channel can be used for this query.

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The returned value is an array of TPCANChannelInformation elements. This array contains as many elements as the value returned when calling CAN_GetValue with the parameter PCAN_ATTACHED_CHANNELS_COUNT.

Default Value

Does not apply.

Initialization Status

Not relevant since this parameter is not channel dependent.

When to Use

It can be used to enumerate all existing PCAN channels in a PC at a given time, in only one function call.

Application – Example of Use

Commonly, an application first shows the connection possibilities it has, before starting with a specific work. This implies searching the system for connectable channels, their names, capabilities, and other characteristics that may help choosing a device to work with. The parameter PCAN_ATTACHED_CHANNELS is used to get all this information with only one function call.

Native (C++)

```
DWORD channelsCount;

if (CAN_GetValue(PCAN_NONEBUS, PCAN_ATTACHED_CHANNELS_COUNT, &channelsCount, 4) == PCAN_ERROR_OK)
{
    printf("Total of %d channels were found:\n", channelsCount);
    if (channelsCount > 0)
    {
        TPCANChannelInformation* channels = new TPCANChannelInformation[channelsCount];
        if (CAN_GetValue(PCAN_NONEBUS, PCAN_ATTACHED_CHANNELS, channels, channelsCount *
sizeof(TPCANChannelInformation)) == PCAN_ERROR_OK)
        {
            for (int i = 0; i < (int)channelsCount; i++)
            {
                printf("%d) ---------------------------\n", i + 1);
                printf("      Name: %s\n", channels[i].device_name);
                printf("    Handle: 0x%X\n", channels[i].channel_handle);
                printf("Controller: %d\n", channels[i].controller_number);
                printf(" Condition: %d\n", channels[i].channel_condition);
                printf("  . . . . . \n");
            }
        }
        delete[] channels;
    }
}
```

Managed (C#)

```
uint channelsCount;

if (PCANBasic.GetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_ATTACHED_CHANNELS_COUNT, out
channelsCount, 4) == TPCANStatus.PCAN_ERROR_OK)
{
    Console.WriteLine("Total of {0} channels were found:", channelsCount);
    if (channelsCount > 0)
    {
        TPCANChannelInformation[] channels = new TPCANChannelInformation[channelsCount];
        if (PCANBasic.GetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_ATTACHED_CHANNELS, channels) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            for (int i = 0; i < channelsCount; i++)
            {
                Console.WriteLine("{0}) ---------------------------\n", i + 1);
                Console.WriteLine("      Name: {0}", channels[i].device_name);
                Console.WriteLine("    Handle: 0x{0:X}", channels[i].channel_handle);
                Console.WriteLine("Controller: {0}", channels[i].controller_number);
                Console.WriteLine(" Condition: {0}", channels[i].channel_condition);
                Console.WriteLine("  . . . . .");
            }
        }
    }
}
```

PCAN_DEVICE_PART_NUMBER

This parameter is used to get the part number (IPEH-number) associated with a PCAN device. This is a text that allows the recognition and differentiation of device models using the same interface, for example USB. A PCAN-USB FD adaptor would return “IPEH-004022” while a classic PCAN-USB would return “IPEH-002021/002022”.

Availability

It is available since version 4.6.0.

Supported By

PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The part number is represented as a string of the form “IPEH-xxxxxx”, where xxxxxx represents a six-digit number. If a device has several revisions so, that it is represented by more than one IPEH number, then all numbers are contained in the same text, separated by a “/” character. For example, a classic PCAN-USB reports the value “IPEH-002021/002022”.
The returned value is a null terminated string with a length of minimum 12 bytes. It is recommended to use a buffer big enough to guarantee success if a special case as described above occur.

Default Value

Does not apply.

Initialization Status

It can be read on initialized or uninitialized PCAN-Channels.

When to Use

It can be used when it is wanted to differentiate between several hardware models using the same interface.

Application – Example of Use

Let’s say that you have a device that you want to acquire again but the label on it is not readable anymore. You could use this parameter to get the IPEH-number, so that you can contact your distributor and ask for that specific device:

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
char partNumber[100] = { 0 };

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
    if (CAN_GetValue(channelUsed, PCAN_DEVICE_PART_NUMBER, partNumber, 100) == PCAN_ERROR_OK)
        printf("Part number: %s", partNumber);
    else
        printf("Error! Could not retrieve the part number of the device.");
```

```
else
    printf("Error! Could not initialize the channel 0x%X", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
StringBuilder partNumber = new StringBuilder(100);

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
    if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_DEVICE_PART_NUMBER, partNumber, 100) ==
TPCANStatus.PCAN_ERROR_OK)
        Console.WriteLine("Part number: {0}", partNumber);
    else
        Console.WriteLine("Error! Could not retrieve the part number of the device.");
else
    Console.WriteLine("Error! Could not initialize the channel 0x{0:X}", channelUsed);
```

Using Informational Parameters

These parameters are intended to give versioning information about the API itself, as well as about the Hardware (e.g., device driver version). This is important since different features can or cannot be available according with the versions being used.

To be sure that a PCAN-Basic software works properly with a specific hardware, it is a good idea to check version parameters at the beginning (after connection). In this way, you can ensure that the software will work for users as it was working for you at development.

Note that when dependences between a PCAN-Parameter and the API and/or driver/firmware Version appear, they will be notified and remarked in the Online-Help of the PCAN-Basic, as well as in our website (e.g., Forum).

PCAN_API_VERSION

This parameter is used to get the API implementation version.

Availability

Available since version 1.0.0.

Supported By

All channels: Due to the API structure, a channel value is needed to get a PCAN-Parameter when using the function CAN_GetValue. But since the API version doesn't depend on a specific channel, any defined channel value can be used, including PCAN_NONEBUS.

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The API version value is represented as a string of the form "a.b.c.d", where:

a: represents the major version number.
b: represents the minor version number.
c: represents the release version number.
d: represents the build number.

All four values have a maximum size of 16 bits that allows a value of 65535 per each. The returned value is a null terminated string with a maximum length of 24 bytes. It is recommended to use a buffer that large to guaranty success in any case.

Default Value

Does not apply.

Initialization Status

Not relevant since this parameter is not channel dependent.

When to Use

It can be used to determine if a feature to be used is available or not, or just as informative output in an application.

Application – Example of Use

Let's say that you want to show from your application a list of the APIs and libraries being used with their versions, so that if any problem appears then a user can get back to you with versioning information.

Native (C++)

```
char apiVersion[MAX_LENGTH_VERSION_STRING] = { 0 };

if (CAN_GetValue(PCAN_NONEBUS, PCAN_API_VERSION, apiVersion, MAX_LENGTH_VERSION_STRING) == PCAN_ERROR_OK)
{
    printf("The PCAN-Basic version used is: %s\n", apiVersion);
}
```

Managed (C#)

```
StringBuilder apiVersion = new StringBuilder(PCANBasic.MAX_LENGTH_VERSION_STRING);

if (PCANBasic.GetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_API_VERSION, apiVersion,
PCANBasic.MAX_LENGTH_VERSION_STRING) == TPCANStatus.PCAN_ERROR_OK)
{
    Console.WriteLine("The PCAN-Basic version used is: {0}", apiVersion);
}
```


PCAN_CHANNEL_VERSION

This parameter is used to obtain information about the underlying device driver of a PCAN device being used as well as to obtain copyright information.

Availability

It is available since version 1.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The information about driver version and copyright is represented as a multiline string (4 lines) offering the following information in each line:

1) Device driver name and driver version.
2) Architecture implemented on the driver and targeted platform.
3) Year of Copyright.
4) Company name and city where its head office is located.

Note that this format is available beginning with the device driver version 3.x. The returned value is a null terminated string with a maximum length of 256 bytes (null termination included). It is recommended to use a buffer that large to guaranty success in any case.

Default Value

Does not apply.

Initialization Status

Not relevant, since this parameter refers to device driver used for a given channel. Device drivers are loaded on Windows start and unloaded again on Windows shutdown.

When to Use

It can be used as informative output about the used driver in an application.

Application – Example of Use

Let's say that your application is distributed without hardware, so that there is the possibility that a user can have a device with a version you have not tested. Using this parameter avoids losing time by looking for an error that may not be caused by your software but using a wrong or old driver (it is assumed, a USB channel is connected, and an error is raised).

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    //"Somewhere a problem occurred and is catched here";

    char channelVersion[MAX_LENGTH_VERSION_STRING] = { 0 };
    printf("Error! An exception has occurred while working with channel 0x%X\n\n", channelUsed);
    if (CAN_GetValue(channelUsed, PCAN_CHANNEL_VERSION, channelVersion, MAX_LENGTH_VERSION_STRING) ==
PCAN_ERROR_OK)
    {
        printf("Please contact us and share following information:\n");
        printf("%s\n", channelVersion);
    }
    else
        printf("It was not possible to get informaiton about the channel");
}
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // "Somewhere a problem occurred and is catched here");

    StringBuilder channelVersion = new StringBuilder(PCANBasic.MAX_LENGTH_VERSION_STRING);
    Console.WriteLine("Error! An exception has occurred while working with channel 0x{0:X}\n", channelUsed);
    if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_CHANNEL_VERSION, channelVersion,
PCANBasic.MAX_LENGTH_VERSION_STRING) == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Please contact us and share following information:");
        Console.WriteLine(channelVersion);
    }
    else
        Console.WriteLine("It was not possible to get informaiton about the channel");

}
```

PCAN_CHANNEL_FEATURES

This parameter is used to obtain information about the special properties of the device being used.

Availability

It is available since version 4.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The information about special features is returned as a "flag" value. At the moment this documentation was written only the following flags were defined:

1) FD_CAPABLE: Indicates that the channel supports Flexible Data rate communication.

Note: To communicate using the new CAN-FD specification, a channel must be FD capable and must be initialized with the function CAN_InitializeFD. After a successful initialization, the CAN communication is carried out by the functions CAN_ReadFD and CAN_WriteFD. Note that FD capable channels and the FD functions can be used for non-FD communication too, i.e., CAN data as specified in the norm ISO 11898 (CAN 2.0 A/B).

2) DELAY_CAPABLE: Indicates that the channel supports the configuration of a delay, in microsecond resolution, between sending frames.

Note: Only FPGA based devices with a firmware version equal to or greater than 2.4.0 support this feature. At the moment this documentation was written only the FPGA based USB devices were able to support delay configuration.

4) **IO_CAPABLE:** Indicates that the hardware represented by a channel is equipped with I/O pins and that those can be configured.

Note: Currently, only PCAN-Chip USB devices support using I/O parameters.

Default Value

Does not apply.

Initialization Status

This parameter can be used with both, initialized and uninitialized PCAN-Channels.

When to Use

It can be used to decide the initialization mode of a PCAN channel, according with its capabilities.

Application – Example of Use

Let's say that your application was updated to support using USB FD hardware. This means, now your application needs to inform the user whether an attached USB hardware is FD capable, to be able to initialize it as FD. You could use this parameter to show a list of FD capable hardware to the user (it is assumed that there are 4 USB channels available):

Native (C++)

```
TPCANHandle channelsToCheck[] = { PCAN_USBBUS1, PCAN_USBBUS2, PCAN_USBBUS3, PCAN_USBBUS4 };
DWORD features;

printf("CAN-FD capable channels:\n");
for (int i = 0; i < 4; i++)
{
    if (CAN_GetValue(channelsToCheck[i], PCAN_CHANNEL_FEATURES, &features, 4) == PCAN_ERROR_OK)
    {
        if ((features & FEATURE_FD_CAPABLE) == FEATURE_FD_CAPABLE)
        {
            printf("Channel 0x%X\n", channelsToCheck[i]);
        }
    }
}
```

Managed (C#)

```
ushort[] channelsToCheck = { PCANBasic.PCAN_USBBUS1, PCANBasic.PCAN_USBBUS2, PCANBasic.PCAN_USBBUS3,
PCANBasic.PCAN_USBBUS4 };
uint features;

Console.WriteLine("CAN-FD capable channels:");
for (int i = 0; i < 4; i++)
{
    if (PCANBasic.GetValue(channelsToCheck[i], TPCANParameter.PCAN_CHANNEL_FEATURES, out features, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        if ((features & PCANBasic.FEATURE_FD_CAPABLE) == PCANBasic.FEATURE_FD_CAPABLE)
        {
            Console.WriteLine("Channel 0x{0:X}", channelsToCheck[i]);
        }
    }
}
```


PCAN_BITRATE_INFO

This parameter is used to obtain information about the bit rate being used, when a channel was initialized using the function CAN_Initialize.

Availability

It is available since version 4.0.0.

It can be read without initialization since version 4.4.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).

PCAN-DNG (Channel PCAN_DNGBUS1).

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).

PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

This value has a resolution of Word (range [0... 65535]), which represents bit rate registers (BTR0-BTR1), for a CAN controller SJA1000.

Default Value

Does not apply.

Initialization Status

Get: It can be read on initialized or uninitialized PCAN-Channels.

When to Use

It can be used to obtain the BTR0BTR1 value representing the bit rate being used.

Application – Example of Use

Let's say that you have connected a channel (PCAN_USBBUS1), using the parameter PCAN_BITRATE_ADAPTING. After connecting you realize that the bit rate being used is different from the given one. This parameter lets you know the bit rate used, so you can

inform the user about the actual bit rate value used for communication (it is assumed, a USB channel is connected at a bitrate other than 500 kBit/s):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD adaptToBitrate = PCAN_PARAMETER_ON;
DWORD bitrateInfo;
TPCANStatus result;

if (CAN_SetValue(channelUsed, PCAN_BITRATE_ADAPTING, &adaptToBitrate, 4) == PCAN_ERROR_OK)
{
    result = CAN_Initialize(channelUsed, PCAN_BAUD_500K);
    if (result == PCAN_ERROR_OK)
    {
        printf("Channel successfully initialized with BTR0BTR1 0x%X", PCAN_BAUD_500K);
    }
    else
        if (result == PCAN_ERROR_CAUTION)
        {
            if(CAN_GetValue(channelUsed, PCAN_BITRATE_INFO, &bitrateInfo, 4) == PCAN_ERROR_OK)
                printf("Channel successfully connected. Bitrate adapted to BTR0BTR1 0x%X", bitrateInfo);
            else
                printf("Error! Could not get the BITRATE_INFO value.");
        }
        else
            printf("Error! Channel could not be initialized.");
}
else
    printf("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint adaptToBitrate = PCANBasic.PCAN_PARAMETER_ON;
uint bitrateInfo;
TPCANStatus result;

if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_BITRATE_ADAPTING, ref adaptToBitrate, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
    result = PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K);
    if (result == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Channel successfully initialized with BTR0BTR1 0x%X",
TPCANBaudrate.PCAN_BAUD_500K);
    }
    else
        if (result == TPCANStatus.PCAN_ERROR_CAUTION)
        {
            if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_BITRATE_INFO, out bitrateInfo, 4) ==
TPCANStatus.PCAN_ERROR_OK)
                Console.WriteLine("Channel successfully connected. Bitrate adapted to BTR0BTR1 0x{0:X}",
bitrateInfo);
            else
                Console.WriteLine("Error! Could not get the BITRATE_INFO value.");
        }
        else
            Console.WriteLine("Error! Channel could not be initialized.");
}
else
    Console.WriteLine("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```

PCAN_BITRATE_INFO_FD

This parameter is used to obtain information about the bit rate being used when a channel was initialized using the function CAN_InitializeFD.

Availability

It is available since version 4.0.0.

It can be read without initialization since version 4.4.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).

PCAN-DNG (Channel PCAN_DNGBUS1).

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

Possible values are strings representing the nominal and data bit rate (see TPCANBitrateFD chapter in the online help of PCAN-Basic) used by a FD capable hardware.

Default Value

Does not apply.

Initialization Status

Get: It can be read on initialized or uninitialized PCAN-Channels.

When to Use

It can be used to obtain the TPCANBitrateFD value representing the bit rate being used.

Application – Example of Use

Let's say that you have connected a channel (PCAN_USBBUS1), using the parameter PCAN_BITRATE_ADAPTING. After connecting you realize the bit rate being used is different from the given one. Asking this parameter lets you know the bit rate used, so you can inform the user about the actual bit rate value used for communication (it is assumed, a USB channel is connected at a bitrate other than 500 kBit/s / 2 MB/s):

Native (C++)

```
#define PCAN_BITRATE_SAE_J2284_4
(LPSTR)"f_clock=80000000,nom_brp=2,nom_tseg1=63,nom_tseg2=16,nom_sjw=16,data_brp=2,data_tseg1=15,data_tseg2=
4,data_sjw=4"

TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD adaptToBitrate = PCAN_PARAMETER_ON;
char bitrateInfoFD[256] = { 0 };
TPCANStatus result;

if (CAN_SetValue(channelUsed, PCAN_BITRATE_ADAPTING, &adaptToBitrate, 4) == PCAN_ERROR_OK)
{
    result = CAN_InitializeFD(channelUsed, PCAN_BITRATE_SAE_J2284_4);
    if (result == PCAN_ERROR_OK)
    {
        printf("Channel successfully initialized with FD-Bitrate %s", PCAN_BITRATE_SAE_J2284_4);
    }
    else
    {
        if (result == PCAN_ERROR_CAUTION)
        {
            if (CAN_GetValue(channelUsed, PCAN_BITRATE_INFO_FD, bitrateInfoFD, 256) == PCAN_ERROR_OK)
                printf("Channel successfully connected. Bitrate adapted to FD-Bitrate %s", bitrateInfoFD);
            else
                printf("Error! Could not get the BITRATE_INFO_FD value.");
        }
        else
            printf("Error! Channel could not be initialized.");
    }
}
else
    printf("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```

Managed (C#)

```
const string PCAN_BITRATE_SAE_J2284_4 =
"f_clock=80000000,nom_brp=2,nom_tseg1=63,nom_tseg2=16,nom_sjw=16,data_brp=2,data_tseg1=15,data_tseg2=4,data_
```

```
sjw=4";

ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint adaptToBitrate = PCANBasic.PCAN_PARAMETER_ON;
StringBuilder bitrateInfoFD = new StringBuilder(256);
TPCANStatus result;

if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_BITRATE_ADAPTING, ref adaptToBitrate, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
    result = PCANBasic.InitializeFD(channelUsed, PCAN_BITRATE_SAE_J2284_4);
    if (result == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Channel successfully initialized with FD-Bitrate {0}", PCAN_BITRATE_SAE_J2284_4);
    }
    else
    {
        if (result == TPCANStatus.PCAN_ERROR_CAUTION)
        {
            if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_BITRATE_INFO_FD, bitrateInfoFD, 256) ==
TPCANStatus.PCAN_ERROR_OK)
                Console.WriteLine("Channel successfully connected. Bitrate adapted to FD-Bitrate {0}",
bitrateInfoFD);
            else
                Console.WriteLine("Error! Could not get the BITRATE_INFO_FD value.");
        }
        else
            Console.WriteLine("Error! Channel could not be initialized.");
    }
}
else
    Console.WriteLine("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```


PCAN_BUSSPEED_NOMINAL

This parameter is used to obtain information about the currently used nominal CAN Bus speed, in **bits per second**.

Availability

It is available since version 4.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).

PCAN-DNG (Channel PCAN_DNGBUS1).

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

This value has a resolution of a Double-Word (range [0... 4294967295]).

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to show a friendly bit rate value, which can be understood well and fast by any user.

Application – Example of Use

Let's say that you have connected a channel (PCAN_USBBUS1), using the parameter PCAN_BITRATE_ADAPTING. After connecting you realize the bit rate being used is different from the given one. Since the configured bit rate could be based on unknown BTR0-BTR1 values, maybe you will not be able to decode this by yourself. This parameter lets you just ask this "decoded" value, so you can be able to show the bit rate used in bits/s, Kbits/s, Mbit/s, etc., instead of its coded bit rate values (like the bit rate registers), which are not intuitive (it is assumed, a USB channel is connected at a bitrate other than 500 kBit/s):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD adaptToBitrate = PCAN_PARAMETER_ON;
DWORD bitrateSpeedNominal;
TPCANStatus result;

if (CAN_SetValue(channelUsed, PCAN_BITRATE_ADAPTING, &adaptToBitrate, 4) == PCAN_ERROR_OK)
{
    result = CAN_Initialize(channelUsed, PCAN_BAUD_500K);
    if (result == PCAN_ERROR_OK)
    {
        printf("Channel successfully initialized at 500 kBit/s");
    }
    else
    {
        if (result == PCAN_ERROR_CAUTION)
        {
            if (CAN_GetValue(channelUsed, PCAN_BUSSPEED_NOMINAL, & bitrateSpeedNominal, 4) == PCAN_ERROR_OK)
                printf("Channel successfully connected. Bitrate adapted to %g kBit/s", bitrateSpeedNominal /
1000.0);
            else
                printf("Error! Could not get the PCAN_BUSSPEED_NOMINAL value.");
        }
        else
            printf("Error! Channel could not be initialized.");
    }
}
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint adaptToBitrate = PCANBasic.PCAN_PARAMETER_ON;
uint bitrateSpeedNominal;
TPCANStatus result;

if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_BITRATE_ADAPTING, ref adaptToBitrate, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
    result = PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K);
    if (result == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Channel successfully initialized at 500 kBit/s");
    }
    else
    {
        if (result == TPCANStatus.PCAN_ERROR_CAUTION)
        {
            if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_BUSSPEED_NOMINAL, out
bitrateSpeedNominal, 4) == TPCANStatus.PCAN_ERROR_OK)
                Console.WriteLine("Channel successfully connected. Bitrate adapted to {0} kBit/s",
bitrateSpeedNominal / 1000.0);
            else
                Console.WriteLine("Error! Could not get the PCAN_BUSSPEED_NOMINAL value.");
         }
         else
             Console.WriteLine("Error! Channel could not be initialized.");
    }
}
else
    Console.WriteLine("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```

PCAN_BUSSPEED_DATA

This parameter is used to obtain information about the currently used CAN data speed (Bit rate Switch), in **bits per second**.

Availability

It is available since version 4.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

This value has a resolution of a Double-Word (range [0... 4294967295]).

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to show a friendly bit rate value, which can be understood well and fast by any user.

Application – Example of Use

Let's say that you have connected a channel (PCAN_USBBUS1), using the parameter PCAN_BITRATE_ADAPTING. After connecting you realize the bit rate being used is different from the given one. Since the configured bit rate could be based on unknown bit rate values, maybe you will not be able to decode this by yourself. This parameter lets you just ask this "decoded" value, so you can be able to show the bit rate used in bits/s, Kbits/s, Mbit/s, etc., instead of its coded bit rate values (clock frequency, sample jump with, etc.), which are not intuitive (it is assumed, a USB channel is connected at a bitrate other than 500 kBit/s / 2 MB/s):

Native (C++)

```
#define PCAN_BITRATE_SAE_J2284_4
(LPSTR)"f_clock=80000000,nom_brp=2,nom_tseg1=63,nom_tseg2=16,nom_sjw=16,data_brp=2,data_tseg1=15,data_tseg2=
4,data_sjw=4"

TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD adaptToBitrate = PCAN_PARAMETER_ON;
DWORD bitrateSpeedNominal, bitrateSpeedData;
TPCANStatus result;

if (CAN_SetValue(channelUsed, PCAN_BITRATE_ADAPTING, &adaptToBitrate, 4) == PCAN_ERROR_OK)
{
    result = CAN_InitializeFD(channelUsed, PCAN_BITRATE_SAE_J2284_4);
    if (result == PCAN_ERROR_OK)
    {
        printf("Channel successfully initialized at FD-Bitrate 500 kBit/s / 2 MB/s");
    }
    else
    {
        if (result == PCAN_ERROR_CAUTION)
        {
            if ((CAN_GetValue(channelUsed, PCAN_BUSSPEED_NOMINAL, &bitrateSpeedNominal, 4) == PCAN_ERROR_OK)
```

```
&&
                (CAN_GetValue(channelUsed, PCAN_BUSSPEED_DATA, &bitrateSpeedData, 4) == PCAN_ERROR_OK))
                printf("Channel successfully connected. Bitrate adapted to %g kBit/s / %g MB/s",
                    bitrateSpeedNominal / 1000.0, bitrateSpeedData / 1000000.0);
            else
                printf("Error! Could not get the BUS SPEED information.");
        }
        else
            printf("Error! Channel could not be initialized.");
    }
}
else
    printf("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```

Managed (C#)

```
const string PCAN_BITRATE_SAE_J2284_4 =
"f_clock=80000000,nom_brp=2,nom_tseg1=63,nom_tseg2=16,nom_sjw=16,data_brp=2,data_tseg1=15,data_tseg2=4,data_
sjw=4";

ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint adaptToBitrate = PCANBasic.PCAN_PARAMETER_ON;
uint bitrateSpeedNominal, bitrateSpeedData;
TPCANStatus result;

if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_BITRATE_ADAPTING, ref adaptToBitrate, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
    result = PCANBasic.InitializeFD(channelUsed, PCAN_BITRATE_SAE_J2284_4);
    if (result == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Channel successfully initialized at 500 kBit/s / 2 MB/s");
    }
    else
    {
        if (result == TPCANStatus.PCAN_ERROR_CAUTION)
        {
            if ((PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_BUSSPEED_NOMINAL, out
bitrateSpeedNominal, 4) == TPCANStatus.PCAN_ERROR_OK) &&
                (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_BUSSPEED_DATA, out bitrateSpeedData, 4)
==  TPCANStatus.PCAN_ERROR_OK))
                Console.WriteLine("Channel successfully connected. Bitrate adapted to {0} kBit/s / {1}
MB/s",
                bitrateSpeedNominal / 1000.0, bitrateSpeedData / 1000000.0);
            else
                Console.WriteLine("Error! Could not get the BUS SPEED information.");
        }
        else
            Console.WriteLine("Error! Channel could not be initialized.");
    }
}
else
    Console.WriteLine("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```


PCAN_LAN_SERVICE_STATUS

This parameter is used to obtain the running status of the System service that is part of the Virtual PCAN-Gateway solution. This service works together with the device driver PCAN-LAN. Both make the interaction with PCAN-LAN hardware possible (PCAN-Gateway Ethernet/Wireless) by using the PCAN environment in a Windows system.

Availability

It is available since version 4.1.0.

Supported By

PCAN_NONEBUS: The status of the service is not tied to any channel connection, i.e. no specific channel can be used for this query.

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The status of the Virtual PCAN-Gateway service can be one of the following defined values:

Defined Value	Description
SERVICE_STATUS_STOPPED	The service is not running, i.e. stopped or in a state different than 'running'.
SERVICE_STATUS_RUNNING	The service is running.

Default Value

Does not apply.

Initialization Status

Not relevant since this parameter is not channel dependent.

When to Use

It can be used to ensure that the Virtual PCAN-Gateway communication is working.

Application – Example of Use

Let's say that you have written an application that connects always automatically to the first PCAN-LAN channel within a timeout of 20 seconds, and that your application is automatically launched when Windows starts. Now let's say that, for some reason, the service starts with a delay of 30 seconds. In this case, your application would be never able to connect the channel, because the timeout would be reached before a LAN channel can be initialized. To avoid this, you could check in your application first if the service is running before trying to connect the channel:

Native (C++)

```
DWORD serviceState;
TPCANStatus result;

do
{
    // Check for the status of the service with an interval of 1 second
    printf("Checking status of PCAN-LAN Service...\n");
    Sleep(1000);
    result = CAN_GetValue(PCAN_NONEBUS, PCAN_LAN_SERVICE_STATUS, &serviceState, 4);

} while ((result == PCAN_ERROR_OK) && (serviceState != SERVICE_STATUS_RUNNING));

if (result == PCAN_ERROR_OK)
{
    printf("The PCAN-LAN service is running. Proceed to establish a connection...");
}
else
    printf("Error! The status of the PCAN-LAN service could not be retrieved");
```

Managed (C#)

```
uint serviceState;
TPCANStatus result;

do
{
    // Check for the status of the service with an interval of 1 second
    Console.WriteLine("Checking status of PCAN-LAN Service...");
    System.Threading.Thread.Sleep(1000);
    result = PCANBasic.GetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LAN_SERVICE_STATUS, out
serviceState, 4);

} while ((result == TPCANStatus.PCAN_ERROR_OK) && (serviceState != PCANBasic.SERVICE_STATUS_RUNNING));

if (result == TPCANStatus.PCAN_ERROR_OK)
{
    Console.WriteLine("The PCAN-LAN service is running. Proceed to establish a connection...");
}
else
    Console.WriteLine("Error! The status of the PCAN-LAN service could not be retrieved");
```

PCAN_FIRMWARE_VERSION

This parameter is used to get the firmware version of the PCAN device associated with a PCAN channel.

Availability

It is available since version 4.4.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

The API version value is represented as a string of the form “a.b.c”, where:

a: represents the major version number.
b: represents the minor version number.
c: represents the release version number.

All three values have a maximum size of 16 bits that allows a value of 65535 per each. The returned value is a null terminated string with a maximum length of 18 bytes. It is recommended to use a buffer that large to guaranty success in any case.

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to determine if a device is up-to-date, or just as informative output in an application.

Application – Example of Use

Let’s say that you want to show information about the PCAN hardware being used in your application, so that a user can get back to you with versioning information if physical problems arise. This would allow you to check, if his/her problem(s) could be caused by an outdated firmware (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
char version[MAX_LENGTH_VERSION_STRING] = { 0 };

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_GetValue(channelUsed, PCAN_FIRMWARE_VERSION, version, MAX_LENGTH_VERSION_STRING) ==
PCAN_ERROR_OK)
    {
        printf("Firmware version: %s", version);
    }
```

```
    else
        printf("Error! Could not retrieve the firmware version.");
}
else
    printf("Error! Could not initialize the channel 0x%X", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
StringBuilder version = new StringBuilder(PCANBasic.MAX_LENGTH_VERSION_STRING);

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_FIRMWARE_VERSION, version,
PCANBasic.MAX_LENGTH_VERSION_STRING) == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Firmware version: {0}", version);
    }
    else
        Console.WriteLine("Error! Could not retrieve the firmware version.");
}
else
    Console.WriteLine("Error! Could not initialize the channel 0x{0:X}.", channelUsed);
```


PCAN_ATTACHED_CHANNELS_COUNT

This parameter is used to get information about all existing PCAN channels on a system in a single call, regardless of their current availability (see CHANNEL_CONDITION).

This parameter is very tied to another, PCAN_ATTACHED_CHANNELS. It returns a buffer of structures of type “TPCANChannelInformation”, containing channels data. The size in bytes of this buffer is calculated using the result of PCAN_ATTACHED_CHANNELS_COUNT multiplied by the size of the structure TPCANChannelInformation.

Availability

It is available since version 4.4.0.

Supported By

PCAN-NONEBUS: The number of available channels is not tied to any channel, i.e., no specific channel can be used for this query.

Access Mode

This parameter can only be read. It cannot be modified.

Possible Values

A number in the range [0…**n**], where **n** is the sum of the maximum supported channels per device. At the time of writing this documentation a maximum of **59** channels can be handled simultaneously: 1 PCAN-Dongle, 2 PCAN-PCC, 8 PCAN-ISA, 16 PCAN-PCI, 16 PCAN-USB, and 16 PCAN-LAN devices.

Default Value

Does not apply.

Initialization Status

Not relevant since this parameter does not depend on a specific channel.

When to Use

It can be used to determine if any channels are currently present on the system or to calculate the size of a buffer for getting information about those channels, if needed.

Application – Example of Use

Commonly, an application first shows the connection possibilities it has, before starting with a specific work. This implies searching the system for connectable channels, their names, capabilities, and other characteristics that may help choosing a device to work with. According to the programming language used, it is needed to generate a buffer big enough to store information about those existing channels. This parameter is then used to calculate the size of that buffer.

Native (C++)

```
DWORD channelsCount;

if (CAN_GetValue(PCAN_NONEBUS, PCAN_ATTACHED_CHANNELS_COUNT, &channelsCount, 4) == PCAN_ERROR_OK)
{
    printf("Total of %d channels were found:\n", channelsCount);
    if (channelsCount > 0)
    {
        TPCANChannelInformation* channels = new TPCANChannelInformation[channelsCount];
        if (CAN_GetValue(PCAN_NONEBUS, PCAN_ATTACHED_CHANNELS, channels, channelsCount *
sizeof(TPCANChannelInformation)) == PCAN_ERROR_OK)
        {
            for (int i = 0; i < (int)channelsCount; i++)
            {
                printf("%d) ----------------------------\n", i + 1);
                printf("      Name: %s\n", channels[i].device_name);
                printf("    Handle: 0x%X\n", channels[i].channel_handle);
                printf("Controller: %d\n", channels[i].controller_number);
                printf(" Condition: %d\n", channels[i].channel_condition);
                printf("  . . . . . \n");
            }
        }
        delete[] channels;
    }
}
```

Managed (C#)

```
uint channelsCount;

if (PCANBasic.GetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_ATTACHED_CHANNELS_COUNT, out
channelsCount, 4) == TPCANStatus.PCAN_ERROR_OK)
{
    Console.WriteLine("Total of {0} channels were found:", channelsCount);
    if (channelsCount > 0)
    {
        TPCANChannelInformation[] channels = new TPCANChannelInformation[channelsCount];
        if (PCANBasic.GetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_ATTACHED_CHANNELS, channels) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            for (int i = 0; i < channelsCount; i++)
            {
                Console.WriteLine("{0}) ----------------------------\n", i + 1);
                Console.WriteLine("      Name: {0}", channels[i].device_name);
                Console.WriteLine("    Handle: 0x{0:X}", channels[i].channel_handle);
                Console.WriteLine("Controller: {0}", channels[i].controller_number);
                Console.WriteLine(" Condition: {0}", channels[i].channel_condition);
                Console.WriteLine("  . . . . .");
            }
        }
    }
}
```

Using Special Behaviors

These parameters are intended to activate some modes on the devices being used that cause those devices to react or work in an exceptional way.

Note that not all modes are supported by all kind of devices.

PCAN_5VOLTS_POWER

This parameter is used for switching the external 5V on the D-Sub connector of a PCAN-Device. This is useful when connecting external bus converter modules to the card (AU5790 / TJA1054)).

Availability

Available since version 1.0.0.
It can be read without initialization since version 4.4.0.

Supported By

PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Notes:
PCAN-USB: only the devices of type “PCAN-USB Hub” can support this parameter.

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter represents an extra voltage that can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The external 5V on the D-Sub connector is inactive.
PCAN_PARAMETER_ON	The external 5V on the D-Sub connector is active.

Default Value

The default state of extra voltage is inactive (PCAN_PARAMETER_OFF). After activating it, the extra 5V stays on the D-Sub until it is expressly deactivated, or the device is reinitialized (plugged-out and plugged-in again, or PC-reboot).

Initialization Status

Get: It can be read on initialized or uninitialized PCAN-Channels.
Set: It can be set on initialized PCAN-Channels only.

When to Use

It can be used when connecting external bus converter modules to a device, so that it is also supplied with power.

Application – Example of Use

Let's say that your application is connected to a Single-Wired CAN network using a PC-Card Channel. A Bus-Converter (e.g., High-speed to Single-Wire CAN) is also connected to the channel used. It will be used only in special cases when you want to transfer software or diagnostic data. You will need to use the PCAN_5VOLTS_POWER to allow the adapter to work (it is assumed a PCAN-PC Card is used):

Native (C++)

```
TPCANHandle channelUsed = PCAN_PCCBUS1;
DWORD powerState = PCAN_PARAMETER_ON;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_SetValue(channelUsed, PCAN_5VOLTS_POWER, &powerState, 4) == PCAN_ERROR_OK)
    {
        printf("The 5V power on channel 0x%X is now active\n", channelUsed);
        printf("Start working...\n");
        // Do needed work
        printf("Work finished!\n");
        powerState = PCAN_PARAMETER_OFF;
        if (CAN_SetValue(channelUsed, PCAN_5VOLTS_POWER, &powerState, 4) == PCAN_ERROR_OK)
        {
            printf("The 5V power on channel 0x%X is now deactivated", channelUsed);
        }
        else
        {
            printf("Error! The 5V power could not be disabled.\n");
            printf("....Risk of damage if short circuit....");
        }
    }
    else
        printf("Error! The 5V power could not be enabled.");
}
else
    printf("Error! Channel could not be initialized.");
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_PCCBUS1;
uint powerState = PCANBasic.PCAN_PARAMETER_ON;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_5VOLTS_POWER, ref powerState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("The 5V power on channel 0x{0:X} is now active", channelUsed);
        Console.WriteLine("Start working...");
        // Do needed work
        Console.WriteLine("Work finished!");
        powerState = PCANBasic.PCAN_PARAMETER_OFF;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_5VOLTS_POWER, ref powerState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("The 5V power on channel 0x{0:X} is now deactivated", channelUsed);
        }
        else
        {
            Console.WriteLine("Error! The 5V power could not be disabled.");
            Console.WriteLine("....Risk of damage if short circuit....");
        }
    }
    else
        Console.WriteLine("Error! The 5V power could not be enabled.");
}
else
    Console.WriteLine("Error! Channel could not be initialized.");
```

PCAN_BUSOFF_AUTORESET

This parameter instructs the PCAN driver to automatically reset the CAN controller of a PCAN Channel when a bus-off state is detected.

Availability

It is available since version 1.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The automatic Hardware reset is OFF.
PCAN_PARAMETER_ON	The automatic Hardware reset is ON.

Default Value

The default state of the automatic reset on bus-off is inactive (PCAN_PARAMETER_OFF). After activating it, the automatic reset stays active until it is expressly deactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to avoid resetting a device manually, after it has reached a bus-off state, for instance, when an application should work unattended, and it is known that bus-off error may occur.

Application – Example of Use

Let's say that your application is running some diagnostics on an Electronic Control unit (ECU) of a car, and this ECU is battery powered (car switch on and off). Having an application communicating to the same CAN Network and having the ECU switching on and off can cause the PCAN-Channel (hardware, CAN Controller) to reach the OFF status. No communication can be achieved until the OFF status disappears. To avoid the need to manually reset the application/PCAN-Channel each time the car is switch on or off, you can use this parameter to do this automatically for you (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD autoResetState = PCAN_PARAMETER_ON;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_SetValue(channelUsed, PCAN_BUSOFF_AUTORESET, &autoResetState, 4) == PCAN_ERROR_OK)
    {
        printf("The channel 0x%X will be reset automatically on bus-off state.\n", channelUsed);
        printf("Start working...\n");
        // Do needed work
```

```
        printf("Work finished!\n");
    }
    else
        printf("Error! The automatic bus-off reset could not be activated");
}
else
    printf("Error! Channel could not be initialized.");
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint autoResetState = PCANBasic.PCAN_PARAMETER_ON;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_BUSOFF_AUTORESET, ref autoResetState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("The channel 0x{0:X} will be reset automatically on bus-off state.", channelUsed);
        Console.WriteLine("Start working...");
        // Do needed work
        Console.WriteLine("Work finished!");
    }
    else
        Console.WriteLine("Error! The automatic bus-off reset could not be activated");
}
else
    Console.WriteLine("Error! Channel could not be initialized.");
```


PCAN_LISTEN_ONLY

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

This parameter is a so called “**pre-initialized**” parameter, which means that it can be set before a PCAN-Channel is initialized to activate/deactivate the parameter as fast as possible, in this way avoiding problems that can appear within sensitive operations.

Availability

It is available since version 1.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.
Note: On PCAN-LAN devices, the Listen-Only mode cannot be set over this parameter. It can only be changed directly on the device, by using its configuration interface.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The Listen-only mode is OFF.
PCAN_PARAMETER_ON	The Listen-Only mode is ON.

Default Value

The default state of the Listen-Only mode is deactivated (PCAN_PARAMETER_OFF). After activating it, the Listen-Only mode stays active until it is expressly deactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used in initialized or uninitialized channels.

When to Use

It can be used when an application wants to passively inspect the data being transferred within a CAN network, without causing any perturbation on it.

Application – Example of Use

Let's say that your application must work in an environment where only 4 different bit rates are used. Since the 4-bit rates are known you want to offer the possibility to auto detect the bit rate that is currently configured in a CAN network at connection time. You could use this parameter to passively connect to a network using different bit rates without causing errors when connecting with a wrong bit rate. In this way your application can recognize the bit rate being used, and the communication is not affected while this procedure is done (it is assumed, a USB channel is connected and there is communication on the bus within 1 second using one of the tested bitrates):

Native (C++)

```
TPCANBaudrate baudrates[] = { PCAN_BAUD_125K, PCAN_BAUD_250K, PCAN_BAUD_500K, PCAN_BAUD_1M };
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD listenOnlyState = PCAN_PARAMETER_ON;
TPCANMsg msg;

printf("Checking the current bitrate on channel 0x%X\n", channelUsed);
for (int i = 0; i < 4; i++)
{
    if (CAN_SetValue(channelUsed, PCAN_LISTEN_ONLY, &listenOnlyState, 4) == PCAN_ERROR_OK)
    {
        if (CAN_Initialize(channelUsed, baudrates[i]) == PCAN_ERROR_OK)
        {
            printf(".....");
            // Wait 1 second before trying to get a message
            Sleep(1000);
            // If a message is received, then bitrate was found
            if (CAN_Read(channelUsed, &msg, NULL) == PCAN_ERROR_OK)
            {
                printf("\nBitrate (BTR0BTR1) used is 0x%X", baudrates[i]);
                break;
            }
            CAN_Uninitialize(channelUsed);
        }
        else
        {
            printf("Error! Channel could not be initialized.");
            break;
        }
    }
    else
    {
        printf("Error! The listen-only feature could not be activated");
        break;
    }
}
```

Managed (C#)

```
TPCANBaudrate[] baudrates = { TPCANBaudrate.PCAN_BAUD_125K, TPCANBaudrate.PCAN_BAUD_250K,
                              TPCANBaudrate.PCAN_BAUD_500K, TPCANBaudrate.PCAN_BAUD_1M };
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint listenOnlyState = PCANBasic.PCAN_PARAMETER_ON;
TPCANMsg msg;

Console.WriteLine("Checking the current bitrate on channel 0x{0:X}", channelUsed);
for (int i = 0; i < 4; i++)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_LISTEN_ONLY, ref listenOnlyState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        if (PCANBasic.Initialize(channelUsed, baudrates[i]) == TPCANStatus.PCAN_ERROR_OK)
        {
            Console.Write(".....");
            // Wait 1 second before trying to get a message
            System.Threading.Thread.Sleep(1000);
            // If a message is received, then bitrate was found
            if (PCANBasic.Read(channelUsed, out msg) == TPCANStatus.PCAN_ERROR_OK)
            {
                Console.WriteLine("\nBitrate (BTR0BTR1) used is 0x{0:X}", baudrates[i]);
                break;
            }
            PCANBasic.Uninitialize(channelUsed);
        }
        else
        {
            Console.WriteLine("Error! Channel could not be initialized.");
            break;
        }
    }
    else
    {
        Console.WriteLine("Error! The listen-only feature could not be activated");
        break;
    }
}
```


PCAN_BITRATE_ADAPTING

This parameter allows the user to connect to an active PCAN-Channel when the bit rate used is unknown. When this mode is set, PCAN-Basic will try first to use the bit rate given as parameter in the initialization process; if the channel has a different bit rate configured, then the new connection will use the configured bit rate and the initialization function will return a warning value, indicating that the used bit rate differs from the given one.

This parameter is a so called "**pre-initialized only**" parameter, which means that it can be only set before a PCAN-Channel is initialized.

Availability

It is available since version 4.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The Bitrate-Adapting feature is OFF.
PCAN_PARAMETER_ON	The Bitrate-Adapting feature is ON.

Default Value

The default state of the Bitrate-Adapting mode is deactivated (PCAN_PARAMETER_OFF). This parameter has effect only at initialize time. It cannot be set after activating a channel. The parameter returns to its default value after calling the initialize/InitializeFD function.

Initialization Status

This parameter can be used only on uninitialized channels.

When to Use

It can be used when an application wants to connect to a channel, regardless of whether the channel is being used (PCAN-View) with a different or unknown bit rate.

Application – Example of Use

Let's say that your application works with remote LAN channels (PCAN-Gateway virtual channels) and you don't know the configured bit rate in one, some, or all of them. Since LAN channel bit rates cannot be changed using the PCAN-Basic API, the initialization will fail if you use a wrong bit rate. Having this parameter activated before calling initialize allows the application to test the bit rate passed, and to ignore it if it doesn't match. In this way the initialization will always succeeds (it is assumed, a LAN channel is available):

Native (C++)

```
TPCANHandle channelUsed = PCAN_LANBUS1;
DWORD adaptToBitrate = PCAN_PARAMETER_ON;
DWORD bitrateSpeedNominal;
TPCANStatus result;

if (CAN_SetValue(channelUsed, PCAN_BITRATE_ADAPTING, &adaptToBitrate, 4) == PCAN_ERROR_OK)
{
    result = CAN_Initialize(channelUsed, PCAN_BAUD_500K);
    if (result == PCAN_ERROR_OK)
    {
        printf("LAN channel successfully initialized at 500 kBit/s");
    }
    else
    {
        if (result == PCAN_ERROR_CAUTION)
        {
            if (CAN_GetValue(channelUsed, PCAN_BUSSPEED_NOMINAL, &bitrateSpeedNominal, 4) == PCAN_ERROR_OK)
                printf("LAN channel successfully connected. Bitrate adapted to %g kBit/s",
bitrateSpeedNominal / 1000.0);
            else
                printf("Error! Could not get the PCAN_BUSSPEED_NOMINAL value.");
        }
        else
            printf("Error! Channel could not be initialized.");
    }
}
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_LANBUS1;
uint adaptToBitrate = PCANBasic.PCAN_PARAMETER_ON;
uint bitrateSpeedNominal;
TPCANStatus result;

if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_BITRATE_ADAPTING, ref adaptToBitrate, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
```

```
    result = PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K);
    if (result == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("LAN channel successfully initialized at 500 kBit/s");
    }
    else
    {
        if (result == TPCANStatus.PCAN_ERROR_CAUTION)
        {
            if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_BUSSPEED_NOMINAL, out
bitrateSpeedNominal, 4) == TPCANStatus.PCAN_ERROR_OK)
                Console.WriteLine("LAN channel successfully connected. Bitrate adapted to {0} kBit/s",
bitrateSpeedNominal / 1000.0);
            else
                Console.WriteLine("Error! Could not get the PCAN_BUSSPEED_NOMINAL value.");
        }
        else
            Console.WriteLine("Error! Channel could not be initialized.");
    }
}
else
    Console.WriteLine("Error! Channel not activate the feature PCAN_BITRATE_ADAPTING.");
```


PCAN_INTERFRAME_DELAY

This parameter helps the user to configure a pause/delay, with a microsecond resolution, between CAN frames being sent within a PCAN-Channel. Other applications working with the same PCAN-Hardware (for instance, a PCAN-View) are not influenced when configuring a delay.

Note: This feature is only supported by FPGA based devices with a firmware version equal to or greater than 2.4.0. At the moment of writing this documentation, only the FPGA based PCAN-USB Devices (PCAN-USB FD, PCAN-USB Pro FD, PCAN-Chip USB) support an inter frame delay.

Availability

It is available since version 4.2.0.

Supported By

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This value must be in the range of [0...1023]) microseconds. If the value to be set is bigger than the resolution supported by the firmware, then the value is truncated.

Default Value

The default value of the inter frame delay is 0, which is mean that the delay is deactivated. After configuring a value bigger than 0, the inter frame delay will be used until it is expressly deactivated (set to 0), or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used only on initialized channels.

When to Use

It can be used on applications that want to increment the separation time of consecutive transmitted CAN frames.

Application – Example of Use

Let's say you have an application that use a FPGA based device like PCAN-USB Pro FD for flashing some ECUs. Your ECUs are distributed and connected using gateways, so that small transmission delays can occur. Since FPGA device can support up to 100% bus load it is possible that your application sends data too fast and that the flashing protocol used can experience problems if it relays on a client/server model like ISO-TP or UDS. You could configure a small delay between packages, so that the maximum bus load will not be reached, and so your processes work without failures (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD delay = 10;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_SetValue(channelUsed, PCAN_INTERFRAME_DELAY, &delay, 4) == PCAN_ERROR_OK)
    {
        printf("Interframe delay set to %d on channel 0x%X.", delay, channelUsed);
        printf("Start working...");
        // Do needed work
        printf("Work finished!");
    }
    else
        printf("Error! Interframe delay is not supported or could not be set");
}
else
    printf("Error! Channel could not be initialized.");
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint delay = 10;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_INTERFRAME_DELAY, ref delay, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Interframe delay set to {0} on channel 0x{1:X}.", delay, channelUsed);
        Console.WriteLine("Start working...");
        // Do needed work
        Console.WriteLine("Work finished!");
    }
    else
        Console.WriteLine("Error! Interframe delay is not supported or could not be set");
}
else
    Console.WriteLine("Error! Channel could not be initialized.");
```

Controlling the Data Flow

These parameters are intended to control the data being received through a PCAN-Channel, how it is received, and even how/when an application should check for new incoming data. According with the amount of information being transmitted within a CAN network it will reasonable to delimit the data being accepted by an application in order to facilitate the work with it.

Receiving a lot of data while only having to process just a part of it can cause the unnecessary use of memory and CPU processing, thus slowing the system down. In the same way, the reaction time for reading incoming data is also the key for successful processing of incoming information.

PCAN_RECEIVE_EVENT

This parameter passes an event handle (Windows Event Objects) to the underlying API. This event will be triggered (its state is set to "signaled") when CAN data is placed into the receive queue of a PCAN-Channel.

Events are normally used when an application separates processing in different execution threads. In a thread, that waiting for an event to occur doesn't affect the normal execution of an application.

Note that the event is not triggered each time a message is included into the queue, but only when it states was "not signaled" and data is received. When an event is signaled, then you must read the queue until emptiness and eventually reset the event (if you are using a manual reset event).

Availability

It is available since version 1.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be enabled or disabled.

Status	Value needed
ENABLED	Valid event object handle, returned by the Windows function CreateEvent.
DISABLED	0, or NULL, or IntPtr.Zero (managed environments).

Default Value

The default state is disabled (0). After enabling this parameter (by configuring an event handle), the PCAN-Basic API will try to signal the handle until it is disabled (by setting as handle a value of 0), or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Note that when you need to reinitialize a PCAN-Channel, you will need to set the event again each time after initializing the channel, since the event will have again its default value of 0 after initialization. **Note** too that it is strongly recommended to close the handle (using CloseHandle) ***after*** a PCAN-Channel has been uninitialized, since the API could try to set an invalid handle and this can cause undesired behavior.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to avoid timeouts: when an application wants to react and process information as fast as possible. It can be used to avoid unnecessary data polling: when an application should check for specific messages that are seldom received and/or it is unknown when they can arrive.

Application – Example of Use

Let's say you have written a diagnostic application used for data updates on a device (e.g., Electronic Control Unit). The application must wait until the device is initialized and then must send a message to set the device in maintenance mode. The device has to response within the first 50 milliseconds after receiving the maintenance message, otherwise means it cannot enter the desired mode. For this, you would start a thread that send the request and wait for a response (it is assumed, a USB channel is connected, that a message with ID=0 and first data byte = 7 causes a device to enter the diagnostic mode, and that a message with ID=0 and first data byte = 0 causes a device to leave the diagnostic mode):

Native (C++)

```
HANDLE readEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
DWORD eventValue = (DWORD)readEvent;
TPCANHandle channelUsed = PCAN_USBBUS1;
TPCANMsg triggerMsg;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (readEvent != 0)
    {
        if (CAN_SetValue(channelUsed, PCAN_RECEIVE_EVENT, &eventValue, 4) == PCAN_ERROR_OK)
        {
            triggerMsg.ID = 0;
            triggerMsg.MSGTYPE = PCAN_MESSAGE_STANDARD;
            triggerMsg.LEN = 1;
            triggerMsg.DATA[0] = 7;

            if (CAN_Write(channelUsed, &triggerMsg) == PCAN_ERROR_OK)
            {
                if (WaitForSingleObject(readEvent, 50) == WAIT_OBJECT_0)
                {
                    printf("Device entered the diagnostig mode. Starting to work...\n");
                    printf("Work finished!\n");
                    triggerMsg.DATA[0] = 0;
                    if (CAN_Write(channelUsed, &triggerMsg) == PCAN_ERROR_OK)
                    {
                        printf("Device back to normal state!\n");
```

```
                    }
                    else
                        printf("Error! Trigger message for normal mode could not be sent.\n");
                }
                else
                    printf("Error - Timeout! Device could not be set in diagnostic mode.\n");
            }
            else
                printf("Error! Trigger message for diagnostic mode could not be sent.\n");
        }
        else
            printf("Error! Read-event could not be configured.\n");
    }
    else
        printf("Error! Read-event could not be created.\n");
}
else
    printf("Error! Channel 0x%X could not be initialized.\n", channelUsed);
```


Managed (C#)

```
System.Threading.AutoResetEvent readEvent = new System.Threading.AutoResetEvent(false);
uint eventValue = (uint)readEvent.SafeWaitHandle.DangerousGetHandle().ToInt32();
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
TPCANMsg triggerMsg = new TPCANMsg();

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_RECEIVE_EVENT, ref eventValue, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        triggerMsg.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDARD;
        triggerMsg.DATA = new byte[8];
        triggerMsg.ID = 0;
        triggerMsg.LEN = 1;
        triggerMsg.DATA[0] = 7;

        if (PCANBasic.Write(channelUsed, ref triggerMsg) == TPCANStatus.PCAN_ERROR_OK)
        {
            if (readEvent.WaitOne(50))
            {
                Console.WriteLine("Device entered the diagnostig mode. Starting to work...");
                Console.WriteLine("Work finished!");
                triggerMsg.DATA[0] = 0;
                if (PCANBasic.Write(channelUsed, ref triggerMsg) == TPCANStatus.PCAN_ERROR_OK)
                {
                    Console.WriteLine("Device back to normal state!");
                }
                else
                    Console.WriteLine("Error! Trigger message for normal mode could not be sent.");
            }
            else
                Console.WriteLine("Error - Timeout! Device could not be set in diagnostic mode.");
        }
        else
            Console.WriteLine("Error! Trigger message for diagnostic mode could not be sent.");
    }
    else
        Console.WriteLine("Error! Read-event could not be configured.");
}
else
    Console.WriteLine("Error! Channel 0x{0:X} could not be initialized.", channelUsed);
```

PCAN_MESSAGE_FILTER

This parameter instructs a PCAN-Channel to receive or not to receive messages by modifying the acceptance mask and acceptance code of its CAN chip.

Note that an internal hardware reset is done when the acceptance mask and code must be modified. If other application is using the same device, its communication could be affected in some scenarios.

Availability

It is available since version 1.0.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

In a setting operation, this parameter can be opened or closed.

Defined Value	Description
PCAN_FILTER_OPEN	The CAN filter allows all messages to pass.
PCAN_FILTER_CLOSE	The CAN filter discards all messages.

In a getting operation, a third value can be received.

Defined Value	Description
PCAN_FILTER_CUSTOM	The CAN filter allows a custom range of messages to pass.

Default Value

The default state of the filter is to receive all messages (PCAN_FILTER_OPEN). **Note** that a PCAN-Channel starts receiving any message being transmitted with a CAN network immediately after the channel is initialized. **Note** also that using the function CAN_FilterMessages will cause the filter to be closed automatically before registering the desired message range, if the filter state before calling the function was PCAN_FILTER_OPEN.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used for switching the acceptance of messages temporarily, for example to avoid receiving unwanted messages during a defined amount of time.

Application – Example of Use

Let's say you have an application reading and interpreting a considerable amount of information from a CAN network and showing it in some visual controls. Because the data fluctuates too fast you would be required to check the general status of the data at some time, but you don't have the possibility to freeze the information being sent within the network. You could close the CAN filter for a while, so that the last received information stays on the visual controls, giving you enough time to check it (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD filterState;
```

```
if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, close the filter...
    filterState = PCAN_FILTER_CLOSE;
    if (CAN_SetValue(channelUsed, PCAN_MESSAGE_FILTER, &filterState, 4) == PCAN_ERROR_OK)
    {
        printf("Filter of channel 0x%X is now closed.\n", channelUsed);
        // do needed work/checks
        printf("Checks finished. Enabling communication again...\n");
        filterState = PCAN_FILTER_OPEN;
        if (CAN_SetValue(channelUsed, PCAN_MESSAGE_FILTER, &filterState, 4) == PCAN_ERROR_OK)
        {
            printf("Filter of channel 0x%X is open again.\n", channelUsed);
        }
        else
            printf("Filter of channel 0x%X could not be restablished\n", channelUsed);
    }
    else
        printf("Filter of channel 0x%X could not be closed\n", channelUsed);
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```


Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint filterState;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, close the filter...
    filterState = PCANBasic.PCAN_FILTER_CLOSE;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_MESSAGE_FILTER, ref filterState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Filter of channel 0x{0:X} is now closed.", channelUsed);
        // do needed work/checks
        Console.WriteLine("Checks finished. Enabling communication again...");
        filterState = PCANBasic.PCAN_FILTER_OPEN;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_MESSAGE_FILTER, ref filterState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Filter of channel 0x{0:X} is open again.", channelUsed);
        }
        else
            Console.WriteLine("Filter of channel 0x{0:X} could not be restablished", channelUsed);
    }
    else
        Console.WriteLine("Filter of channel 0x{0:X} could not be closed", channelUsed);
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_RECEIVE_STATUS

This parameter helps the user to allow / disallow the reception of messages (Data, Status, and Error frames) within a PCAN-Channel, regardless of the value of its reception filter. The acceptance filter of the PCAN-Channel remains unchanged (other applications working with the same PCAN-Hardware will not be disturbed).

This parameter is a so called "**pre-initialized**" parameter, which means that it can be set before a PCAN-Channel is initialized to activate/deactivate the parameter as fast as possible, avoiding in this way problems that can appears within sensitive operations.

Availability

It is available since version 1.1.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The message receiving status is OFF.
PCAN_PARAMETER_ON	The message receiving status is ON.

Default Value

The default value of the receive status is activated (PCAN_PARAMETER_ON). After deactivating it, the receiving status stays inactive until it is expressly reactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used in initialized or uninitialized channels.

When to Use

It can be used on applications that want to discard messages for a while, without having to take modifications on the message filter, avoiding disturbances within the device being used.

Application – Example of Use

Suppose you have an application that uses a complicated filter, for example twelve different message ranges. At some point, you need to stop receiving messages for a while and you want to avoid reconfiguring the filter and resetting the CAN controller, which happens when the filter needs to be reconfigured (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD receptionState;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of messages is turned off...
    receptionState = PCAN_PARAMETER_OFF;
    if (CAN_SetValue(channelUsed, PCAN_RECEIVE_STATUS, &receptionState, 4) == PCAN_ERROR_OK)
    {
        printf("Message reception on channel 0x%X is now disabled.\n", channelUsed);
        // do needed work...
        printf("Operation finished. Enabling communication again...\n");
        receptionState = PCAN_PARAMETER_ON;
        if (CAN_SetValue(channelUsed, PCAN_RECEIVE_STATUS, &receptionState, 4) == PCAN_ERROR_OK)
        {
            printf("Normal opertaion on channel 0x%X restablished.", channelUsed);
        }
        else
            printf("Message reception of channel 0x%X could not be restablished", channelUsed);
    }
    else
        printf("Message reception of channel 0x%X could not be changed", channelUsed);
}
else
    printf("Channel 0x%X could not be initialized", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint receptionState;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of messages is turned off...
    receptionState = PCANBasic.PCAN_PARAMETER_OFF;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_RECEIVE_STATUS, ref receptionState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Message reception on channel 0x{0:X} is now disabled.", channelUsed);
        // do needed work...
        Console.WriteLine("Operation finished. Enabling communication again...");
        receptionState = PCANBasic.PCAN_PARAMETER_ON;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_RECEIVE_STATUS, ref receptionState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Normal opertaion on channel 0x{0:X} restablished.", channelUsed);
        }
        else
            Console.WriteLine("Message reception of channel 0x{0:X} could not be restablished",
channelUsed);
    }
    else
        Console.WriteLine("Message reception of channel 0x{0:X} could not be changed", channelUsed);
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```


PCAN_ALLOW_ STATUS_FRAMES

This parameter helps the user to allow / disallow the reception of Status frames within a PCAN-Channel. This parameter doesn't affect the acceptance filter of the PCAN-Channel. Furthermore, other applications working with the same PCAN-Hardware can individually configure the reception of Status frames.

Note that disabling the PCAN_RECEIVE_STATUS parameter also suppresses the reception of Status frames.

Availability

It is available since version 4.2.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The reception of Status frames is OFF.
PCAN_PARAMETER_ON	The reception of Status frames is ON.

Default Value

The default value of the Status frames reception is activated (PCAN_PARAMETER_ON). After deactivating it, the Status frames reception stays inactive until it is expressly reactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used only on initialized channels.

When to Use

It can be used on applications that want to allow/discard Status frames, since this is not possible using the acceptance filter.

Application – Example of Use

Let's say you have an application that needs to wake up a device by sending a message. It is possible that sending the wake-up message generates some disturbance in the bus since the device is in sleep mode. You can deactivate the reception of Status frames for a while, until the device is awake and running (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD statusFrames;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of status frames is turned off...
    statusFrames = PCAN_PARAMETER_OFF;
    if (CAN_SetValue(channelUsed, PCAN_ALLOW_STATUS_FRAMES, &statusFrames, 4) == PCAN_ERROR_OK)
    {
        printf("Reception of status frames on channel 0x%X is now disabled.\n", channelUsed);
        // do needed work...
        printf("Operation finished. Enabling reception of status frames again...\n");
        statusFrames = PCAN_PARAMETER_ON;
        if (CAN_SetValue(channelUsed, PCAN_ALLOW_STATUS_FRAMES, &statusFrames, 4) == PCAN_ERROR_OK)
        {
            printf("Reception of status frames on channel 0x%X restablished.\n", channelUsed);
        }
        else
            printf("Reception of status frames for channel 0x%X could not be restablished\n", channelUsed);
    }
    else
        printf("Reception of status frames for channel 0x%X could not be changed\n", channelUsed);
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint statusFrames;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of status frames is turned off...
    statusFrames = PCANBasic.PCAN_PARAMETER_OFF;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_STATUS_FRAMES, ref statusFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Reception of status frames on channel 0x{0:X} is now disabled.", channelUsed);
        // do needed work...
        Console.WriteLine("Operation finished. Enabling reception of status frames again...");
        statusFrames = PCANBasic.PCAN_PARAMETER_ON;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_STATUS_FRAMES, ref statusFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Reception of status frames on channel 0x{0:X} restablished.", channelUsed);
        }
        else
            Console.WriteLine("Reception of status frames for channel 0x{0:X} could not be restablished",
channelUsed);
    }
    else
        Console.WriteLine("Reception of status frames for channel 0x{0:X} could not be changed",
```

```
channelUsed);
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_ALLOW_ RTR_FRAMES

This parameter helps the user to allow / disallow the reception of RTR frames within a PCAN-Channel. This parameter doesn't affect the acceptance filter of the PCAN-Channel. Furthermore, other applications working with the same PCAN-Hardware can individually configure the reception of RTR frames.

Note that disabling the PCAN_RECEIVE_STATUS parameter also suppresses the reception of RTR frames.

Availability

It is available since version 4.2.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The reception of RTR frames is OFF.
PCAN_PARAMETER_ON	The reception of RTR frames is ON.

Default Value

The default value of the RTR frames reception is activated (PCAN_PARAMETER_ON). After deactivating it, the RTR frames reception stays inactive until it is expressly reactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used only on initialized channels.

When to Use

It can be used on applications that want to allow/discard RTR frames for a while, without having to take modifications on the message filter, avoiding disturbances within the device being used.

Application – Example of Use

Let's say you have an application that responses to RTR frames with information that can vary, e.g., it can be set by the user. You can deactivate the reception of RTR messages (and their processing) while a user is updating this information, without having to stop or disable the code handling RTRs (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD rtrFrames;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of RTR frames is turned off...
    rtrFrames = PCAN_PARAMETER_OFF;
    if (CAN_SetValue(channelUsed, PCAN_ALLOW_RTR_FRAMES, &rtrFrames, 4) == PCAN_ERROR_OK)
    {
        printf("Reception of RTR frames on channel 0x%X is now disabled.\n", channelUsed);
        // do needed work...
        printf("Operation finished. Enabling reception of RTR frames again...\n");
        rtrFrames = PCAN_PARAMETER_ON;
        if (CAN_SetValue(channelUsed, PCAN_ALLOW_RTR_FRAMES, &rtrFrames, 4) == PCAN_ERROR_OK)
        {
            printf("Reception of RTR frames on channel 0x%X restablished.\n", channelUsed);
        }
        else
            printf("Reception of RTR frames for channel 0x%X could not be restablished\n", channelUsed);
    }
    else
        printf("Reception of RTR frames for channel 0x%X could not be changed\n", channelUsed);
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint rtrFrames;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of RTR frames is turned off...
    rtrFrames = PCANBasic.PCAN_PARAMETER_OFF;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_RTR_FRAMES, ref rtrFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Reception of RTR frames on channel 0x{0:X} is now disabled.", channelUsed);
        // do needed work...
        Console.WriteLine("Operation finished. Enabling reception of RTR frames again...");
        rtrFrames = PCANBasic.PCAN_PARAMETER_ON;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_RTR_FRAMES, ref rtrFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Reception of RTR frames on channel 0x{0:X} restablished.", channelUsed);
        }
        else
            Console.WriteLine("Reception of RTR frames for channel 0x{0:X} could not be restablished",
channelUsed);
    }
    else
        Console.WriteLine("Reception of RTR frames for channel 0x{0:X} could not be changed", channelUsed);
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_ALLOW_ ERROR_FRAMES

This parameter helps the user to allow / disallow the reception of CAN Error frames within a PCAN-Channel. This parameter doesn't affect the acceptance filter of the PCAN-Channel. Furthermore, other applications working with the same PCAN-Hardware can individually configure the reception of Error frames.

Note that disabling the PCAN_RECEIVE_STATUS parameter also suppresses the reception of Error frames.

Availability

It is available since version 4.2.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The reception of CAN Error frames is OFF.
PCAN_PARAMETER_ON	The reception of CAN Error frames is ON.

Default Value

The default value of the CAN Error frames reception is deactivated (PCAN_PARAMETER_OFF). After activating it, the CAN Error frames reception stays active until it is expressly deactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used only on initialized channels.

When to Use

It can be used in applications that want to allow/discard CAN Error frames, since this is not possible using the acceptance filter.

Application – Example of Use

Let's say you have an application that is not showing the expected behavior regarding CAN communication. You could activate the Error frames to see if the CAN bus is disturbed and to get more information about possible causes for it (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD errorFrames;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of Error frames is turned on...
    errorFrames = PCAN_PARAMETER_ON;
    if (CAN_SetValue(channelUsed, PCAN_ALLOW_ERROR_FRAMES, &errorFrames, 4) == PCAN_ERROR_OK)
    {
        printf("Reception of Error frames on channel 0x%X is now enabled.\n", channelUsed);
        // do needed work...
        printf("Operation finished. Disabling reception of Error frames again...\n");
        errorFrames = PCAN_PARAMETER_OFF;
        if (CAN_SetValue(channelUsed, PCAN_ALLOW_ERROR_FRAMES, &errorFrames, 4) == PCAN_ERROR_OK)
        {
```

```
            printf("Reception of Error frames on channel 0x%X disabled.\n", channelUsed);
        }
        else
            printf("Reception of Error frames for channel 0x%X could not be disabled\n", channelUsed);
    }
    else
        printf("Reception of Error frames for channel 0x%X could not be changed\n", channelUsed);
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint errorFrames;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Doing work. At some point, when needed, the reception of Error frames is turned on...
    errorFrames = PCANBasic.PCAN_PARAMETER_ON;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_ERROR_FRAMES, ref errorFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Reception of Error frames on channel 0x{0:X} is now enabled.", channelUsed);
        // do needed work...
        Console.WriteLine("Operation finished. Disabling reception of Error frames again...");
        errorFrames = PCANBasic.PCAN_PARAMETER_OFF;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_ERROR_FRAMES, ref errorFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Reception of Error frames on channel 0x{0:X} disabled.", channelUsed);
        }
        else
            Console.WriteLine("Reception of Error frames for channel 0x{0:X} could not be disabled",
channelUsed);
    }
    else
        Console.WriteLine("Reception of Error frames for channel 0x{0:X} could not be changed",
channelUsed);
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```


PCAN_ALLOW_ ECHO_FRAMES

This parameter helps the user to allow / disallow the reception of echo messages within a PCAN-Channel, this is, the reception of frames sent by itself. This parameter doesn't affect the acceptance filter of the PCAN-Channel. Furthermore, other applications working with the same PCAN-Hardware can individually configure the reception of echo frames.

Note that disabling the PCAN_RECEIVE_STATUS parameter also suppresses the reception of echo frames.

Availability

It is available since version 4.6.0.

Supported By

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Notes:

Requires a device driver version equal to or greater than 4.3.0.

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The reception of Echo frames is OFF.
PCAN_PARAMETER_ON	The reception of Echo frames is ON.

Default Value

The default value of the Echo frames reception is deactivated (PCAN_PARAMETER_OFF). After activating it, the Echo frames reception stays active until it is expressly deactivated, or the channel is disconnected (e.g., using the function CAN_Uninitialize).

Initialization Status

This parameter can be used only on initialized channels.

When to Use

It can be used in applications that want to allow/discard Echo CAN frames, since this is not possible using the acceptance filter.

Application – Example of Use

Let's say you have an application that need to assure, that some messages are physically sent before sending additional data or proceeding to do other tasks. You could activate the Echo frames to assert that all needed messages were placed in the CAN network:

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    DWORD echoFrames = PCAN_PARAMETER_ON;
    if (CAN_SetValue(channelUsed, PCAN_ALLOW_ECHO_FRAMES, &echoFrames, 4) == PCAN_ERROR_OK)
    {
        printf("Reception of Echo frames on channel 0x%X is now enabled.\n", channelUsed);
        // Required data is sent
        TPCANMsg toSend;
        toSend.ID = 1;
        toSend.LEN = 1;
        toSend.DATA[0] = 1;
        toSend.MSGTYPE = PCAN_MESSAGE_STANDARD;

        if (CAN_Write(channelUsed, &toSend) == PCAN_ERROR_OK)
        {
            // Gives time for the message to be sent
            Sleep(10);
            // Check for Echo frame
            TPCANMsg received;
            bool echoReceived = false;
            while (CAN_Read(channelUsed, &received, NULL) != PCAN_ERROR_QRCVEMPTY)
                if (received.MSGTYPE & PCAN_MESSAGE_ECHO)
                {
                    echoReceived = true;
                    break;
                }

            if(echoReceived)
                printf("Echo frame received on channel 0x%X.\n", channelUsed);
            else
                printf("Error! Echo frame never received.\n");
        }
        else
            printf("Error while trying to send a CAN message on channel 0x%X\n", channelUsed);
    }
    else
        printf("Reception of Echo frames for channel 0x%X could not be changed\n", channelUsed);
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    uint echoFrames = PCANBasic.PCAN_PARAMETER_ON;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ALLOW_ECHO_FRAMES, ref echoFrames, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Reception of Echo frames on channel 0x{0:X} is now enabled.", channelUsed);
        // Required data is sent
        TPCANMsg toSend = new TPCANMsg();
        toSend.DATA = new byte[8];
        toSend.ID = 1;
        toSend.LEN = 1;
        toSend.DATA[0] = 1;
        toSend.MSGTYPE = TPCANMessageType.PCAN_MESSAGE_STANDARD;

        if(PCANBasic.Write(channelUsed, ref toSend) == TPCANStatus.PCAN_ERROR_OK)
        {
            // Gives time for the message to be sent
            System.Threading.Thread.Sleep(10);
            // Check for Echo frame
            TPCANMsg received;
            bool echoReceived = false;
            while(PCANBasic.Read(channelUsed, out received) == TPCANStatus.PCAN_ERROR_OK)
                if((received.MSGTYPE & TPCANMessageType.PCAN_MESSAGE_ECHO) ==
TPCANMessageType.PCAN_MESSAGE_ECHO)
                {
                    echoReceived = true;
                    break;
                }

            if(echoReceived)
                Console.WriteLine("Echo frame received on channel 0x{0:X}.", channelUsed);
            else
                Console.WriteLine("Error! Echo frame never received.");
        }
        else
            Console.WriteLine("Error while trying to send a CAN message on channel 0x{0:X}", channelUsed);
    }
    else
        Console.WriteLine("Reception of Echo frames for channel 0x{0:X} could not be changed", channelUsed);
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```


PCAN_ACCEPTANCE_FILTER_11BIT

This parameter helps the user 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.

This parameter allows the configuration of complex filter patterns, and it is intended for users with extended CAN knowledge. Note that the calculation of mask and code patterns is not a trivial matter. For most applications the use of the function CAN_FilterMessages for setting message reception ranges is more adequate. A simple example on code and mask calculation can be seen in the Appendix D.

Notes:

- The acceptance code and mask are coded together in a 64-bit value, each of them using 4 bytes. The acceptance **code** is stored at the **most significant** bytes. Bitwise and shifting operations are needed to code and decode the values into and from a 64-bit unsigned integer variable.
- To set an acceptance code and mask denoting 29-bit CAN IDs, the parameter PCAN_ACCEPTANCE_FILTER_29BIT must be used instead.

- The SJA1000 CAN controller has only one acceptance filter for both, standard (11-bit) and extended (64-bit) IDs. When doing settings for 11-bit IDs, the acceptance mask and code are internally shifted to the left as adaptation measure, which also causes possible reception of unwanted messages. For this reason, is also not advisable to mix 11-bits and 29-bits filters.
- An internal hardware reset is done each time the acceptance filter is modified. If other application is using the same device, its communication could be affected in some scenarios.

Availability

It is available since version 4.2.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter has a quad-word resolution. Since it contains two double-word values representing the acceptance **code** and **mask**, the maximum value range accepted for this parameter is given by the limits of their internal values, which is a range between [0...16838]. This means, the maximum value of this parameter as 64-bit value is 70364449226751, that is, hexadecimal **00003FFF00003FFF**h.

Default Value

The default state of the reception filter is to receive all messages (PCAN_FILTER_OPEN). This represents a default acceptance **code** of **0**h and an acceptance **mask** of **7FF**h (**00000000000007FF**h). **Note** that an automatic filter reset is done before registering the desired code and mask, if the filter state before using this parameter was PCAN_FILTER_OPEN.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when it is necessary to allow or block the reception of specific CAN messages whose identifiers follow a concrete pattern, and when those patterns are difficult to represent as a simple range of messages.

Application – Example of Use

Let's say you want to write an application that read data from an ECU for diagnostic purposes. The ECU sends a lot of information periodically and you are interested only in 3 messages, 100h, 400h, and 500h. Using the function CAN_FilterMessage would imply to do three calls, one for each ID, which in turn cause 3 hardware resets. With only one call to CAN_SetValue using the parameter PCAN_ACCEPTANCE_FILTER_11BIT and the value

0000000000000500h you cause the same effect, the acceptance filter will only let the reception of those 3 IDs, but you save two function calls and two unnecessary hardware resets (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
uint64_t acceptanceFilter11 = 0x500;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_SetValue(channelUsed, PCAN_ACCEPTANCE_FILTER_11BIT, &acceptanceFilter11, 8) == PCAN_ERROR_OK)
    {
        printf("The filter was configured to accept the standard IDs 0x100, 0x400, and 0x500\n");
    }
    else
        printf("Error! The 11-bit acceptance filter could not be set\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```


Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
ulong acceptanceFilter11 = 0x500;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if(PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ACCEPTANCE_FILTER_11BIT, ref acceptanceFilter11,
8) == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("The filter was configured to accept the standard IDs 0x100, 0x400, and 0x500");
    }
    else
        Console.WriteLine("Error! The 11-bit acceptance filter could not be set");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_ACCEPTANCE_FILTER_29BIT

This parameter helps the user 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.

This parameter allows the configuration of complex filter patterns, and it is intended for users with extended CAN knowledge. Note that the calculation of mask and code patterns is not a trivial matter. For most applications the use of the function CAN_FilterMessages for setting message reception ranges is more adequate. A simple example on code and mask calculation can be seen in the Appendix D.

Notes:

- The acceptance code and mask are coded together in a 64-bit value, each of them using 4 bytes. The acceptance **code** is stored at the **most significant** bytes. Bitwise and shifting operations are needed to code and decode the values into and from a 64-bit unsigned integer variable.
- To set an acceptance code and mask denoting 11-bit CAN IDs, the parameter PCAN_ACCEPTANCE_FILTER_11BIT must be used instead.
- The SJA1000 CAN controller has only one acceptance filter for both, standard (11-bit) and extended (64-bit) IDs. When doing settings for 11-bit IDs, the acceptance mask and code are internally shifted to the left as adaptation measure, which also causes possible reception of unwanted messages. For this reason, is also not advisable to mix 11-bits and 29-bits filters.

- An internal hardware reset is done each time the acceptance filter is modified. If other application is using the same device, its communication could be affected in some scenarios.

Availability

It is available since version 4.2.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter has a quad-word resolution. Since it contains two double-word values representing the acceptance code and mask, the maximum value range accepted for this parameter is given by the limits of their internal values, which is a range between [0... 4294967295]. This means, the maximum value of this parameter as 64-bit value is 18446744073709551615, that is, hexadecimal **FFFFFFFFFFFFFFFF**h.

Default Value

The default state of the reception filter is to receive all messages (PCAN_FILTER_OPEN). This represents a default acceptance **code** of **0**h and an acceptance **mask** of **1FFFFFFF**h (**00000000**1FFFFFFFh). **Note** that an automatic filter reset is done before registering the desired code and mask, if the filter state before using this parameter was PCAN_FILTER_OPEN.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when it is necessary to allow or block the reception of particular CAN messages whose identifiers follow a concrete pattern, and when those patterns are difficult to represent as a simple range of messages.

Application – Example of Use

Let's say you want to write an application that read data from an ECU for diagnostic purposes. The ECU sends a lot of information periodically and you are interested only in 3 messages, 1100h, 1400h, and 1500h. Using the function CAN_FilterMessage would imply to do three calls, one for each ID, which in turn cause 3 hardware resets. With only one call to CAN_SetValue using the parameter PCAN_ACCEPTANCE_FILTER_29BIT and the value **00001000**00000500**h** you achieve the same effect; the acceptance filter will only let the reception of those 3 IDs, but you save two function calls and two unnecessary hardware resets (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
uint64_t acceptanceFilter29 = 0x0000100000000500;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_SetValue(channelUsed, PCAN_ACCEPTANCE_FILTER_29BIT, &acceptanceFilter29, 8) == PCAN_ERROR_OK)
    {
        printf("The filter was configured to accept the extended IDs 0x1100, 0x1400, and 0x1500\n");
    }
    else
        printf("Error! The 29-bit acceptance filter could not be set\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
ulong acceptanceFilter29 = 0x0000100000000500;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_ACCEPTANCE_FILTER_29BIT, ref acceptanceFilter29,
8) == TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("The filter was configured to accept the extended IDs 0x1100, 0x1400, and
0x1500");
    }
    else
        Console.WriteLine("Error! The 29-bit acceptance filter could not be set");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

Using Logging Parameters

These parameters are intended to support the developing phase of a PCAN-Basic project by helping with debug operations. Using the logging system can help finding logic problems within the use of the API, detecting problems with the data being sent or received, checking parameter data, commands order, etc.

It is also possible to activate / deactivate and configure the logging functionality without having to change the code of an application, which allows later debugging session after an application is already released. More information about this can be found in the online forum, Activate debug-logging over Windows Registry, or in Appendix A.

The logging functionality is not tied to a PCAN-Channel but to the use of the PCAN-Basic library itself. This implies three important points:

- The PCAN-Channel handle to use in any CAN_GetValue / CAN_SetValue must be PCAN_NONEBUS, if any PCAN_LOG_* parameter is used. Any other value will cause the function to fail.
- The data logged corresponds to the API calls issued by the process that has loaded the PCAN-Basic dll.
- You cannot start a debug session for different threads of the same application.

PCAN_LOG_LOCATION

This value is used to set the folder on a computer in where the Log-File will be stored, within a debug session.

Note that setting this value starts recording debug information automatically. You could include calls to this parameter in any part of your code that normally shouldn't has to be executed, so you will be notified through the log file if this point was reached (as a kind of assert).

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** too that the name of the log file cannot be specified. The name of the log file is always **PCANBasic.log**.

Availability

It is available since version 1.0.0.

Supported By

PCAN_NONEBUS: Logging parameters are used globally, i.e., they are not tied to a specific PCAN-Channel, but to a specific process.

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This value is a string containing a fully qualified and valid path to an existing directory on the executing computer. To use the default path (calling process path) an empty string must be set.

Kind of Path	Value needed
CUSTOM Path	A valid directory string (Files and Paths).
DEFAULT Path	Empty string (calling process folder).

Default Value

The default value is the path to the calling process folder.

Initialization Status

Does not apply. It is not necessary to have any PCAN-Channel initialized to use this parameter.

When to Use

It can be used when you want to differentiate on debug or logging session by assigning different paths and creating several PCANBasic.log files.

Application – Example of Use

Let's say you have started several instances of the same program and you want to debug all of them at the same time. Additionally, you want to separate the log files per application. You could create a folder for each and configure the path on each application, so that each of them can create its own log file:

Native (C++)

```
int processId = GetProcessId(GetCurrentProcess());
char buffer[MAX_PATH], fullPath[MAX_PATH];

GetTempPathA(MAX_PATH, buffer);
sprintf_s(fullPath, MAX_PATH, "%sPCAN-Basic_%d", buffer, processId);

if (CreateDirectoryA(fullPath, NULL))
{
    if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_LOCATION, fullPath, MAX_PATH) == PCAN_ERROR_OK)
    {
        printf("Logging is active. The log file is located at:\n%s", fullPath);
    }
    else
        printf("Error! Log location could not be configured.\n");
}
else
    printf("Folder for log file could not be created at:\n%s", fullPath);
```

Managed (C#)

```
int processId = GetProcessId(GetCurrentProcess());
char logsFolder[MAX_PATH];
char fullPath[MAX_PATH];

GetTempPathA(MAX_PATH, logsFolder);
sprintf_s(fullPath, MAX_PATH, "%sPCAN-Basic_%d", logsFolder, processId);

if (CreateDirectoryA(fullPath, NULL))
{
    if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_LOCATION, fullPath, MAX_PATH) == PCAN_ERROR_OK)
    {
        printf("Logging is active. The log file is located at:\n%s", fullPath);
    }
    else
        printf("Error! Log location could not be configured.\n");
}
else
    printf("Folder for log file could not be created at:\n%s", fullPath);
```

PCAN_LOG_STATUS

This parameter helps the user to control the activity status of a debug session within the PCAN-Basic API.

Note that if the logging status is set to ON without having configured a destination path for the log file or without having configured the information to be logged, then the session process will start with the default values, which equates to the log file being placed in the folder where the calling process is located and only exceptions will be logged.

Availability

It is available since version 1.0.0.

Supported By

PCAN_NONEBUS: Logging parameters are used globally, i.e., they are not tied to a specific PCAN-Channel, but to a specific process.

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The Logging is OFF.
PCAN_PARAMETER_ON	The Logging is ON.

Default Value

The default value of the Logging mode is deactivated (PCAN_PARAMETER_OFF). After activating it, the logging functionality stays active until it is expressly deactivated.

Initialization Status

Does not apply. It is not necessary to have any PCAN-Channel initialized to use this parameter.

When to Use

It can be used to interrupt debug sessions (start, stop, restart, etc.).

Application – Example of Use

Let's say you want to debug your application. You already noted that you have an intermittent problem. To get only logged data that potentially contains information about the issue being investigated, you could activate the debug session only in those moments in which the anomaly takes place:

Native (C++)

```
DWORD logState = PCAN_PARAMETER_ON;

//... when needed, the logging functionality is activated
if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_STATUS, &logState, 4) == PCAN_ERROR_OK)
{
    printf("Logging is enabled.\n");
    //... Log as needed
    logState = PCAN_PARAMETER_OFF;
    if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_STATUS, &logState, 4) == PCAN_ERROR_OK)
    {
```

```
        printf("Logging is disabled.\n");
    }
    else
        printf("Error! Logging could not be disabled.\n");
}
else
    printf("Error! Logging could not be enabled.\n");
```

Managed (C#)

```
uint logState = PCANBasic.PCAN_PARAMETER_ON;

//... when needed, the logging functionality is activated
if (PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LOG_STATUS, ref logState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
    Console.WriteLine("Logging is enabled.");
    //... Log as needed
    logState = PCANBasic.PCAN_PARAMETER_OFF;
    if (PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LOG_STATUS, ref logState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Logging is disabled.");
    }
    else
        Console.WriteLine("Error! Logging could not be disabled.");
}
else
    Console.WriteLine("Error! Logging could not be enabled.");
```


PCAN_LOG_CONFIGURE

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.

Availability

It is available since version 1.0.0.

Supported By

PCAN_NONEBUS: Logging parameters are used globally, i.e. they are not tied to a specific PCAN-Channel.

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be configured with one of the following values or a combination of those:

Defined Value	Description
LOG_FUNCTION_DEFAULT	This value is always active.
LOG_FUNCTION_ENTRY	Logs when a function is entered.
LOG_FUNCTION_PARAMETERS	Logs the parameters passed to a function.
LOG_FUNCTION_LEAVE	Logs when a function is leaved and its return value.
LOG_FUNCTION_WRITE	Logs the parameters and CAN data passed to the CAN_Write function.
LOG_FUNCTION_READ	Logs the parameters and CAN data received through the CAN_Read function.

Default Value

The default value of this parameter is to log only internal exceptions (LOG_FUNCTION_DEFAULT). **Note** that having only this default value can cause to log no data

at all, since the appearance of exceptions are very rare (we do our best to maintain this API bugs free ☺).

Initialization Status

Does not apply. It is not necessary to have any PCAN-Channel initialized to use this parameter.

When to Use

It can be used when only specific debug information is desired.

Application – Example of Use

Let's say you have an application that has a problem with the sequence in which some API functions are called, and you want to know which function is being called too early or too late. You could configure the debug session to only log the calling of the functions, so that you can see the order in which those functions are processed:

Native (C++)

```
DWORD configuration = LOG_FUNCTION_ENTRY;
DWORD logState = PCAN_PARAMETER_ON;

if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_CONFIGURE, &configuration, 4) == PCAN_ERROR_OK)
{
    if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_STATUS, &logState, 4) == PCAN_ERROR_OK)
    {
        printf("Debug operation started.\n");
        //... Log as needed
        logState = PCAN_PARAMETER_OFF;
        if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_STATUS, &logState, 4) == PCAN_ERROR_OK)
        {
            printf("Debug operation finished. Please check the log file.\n");
        }
        else
            printf("Error! Logging could not be disabled.\n");
    }
    else
        printf("Error! Logging could not be enabled.\n");
}
else
    printf("Error! Logging could not be configured.\n");
```

Managed (C#)

```
uint configuration = PCANBasic.LOG_FUNCTION_ENTRY;
uint logState = PCANBasic.PCAN_PARAMETER_ON;

if (PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LOG_CONFIGURE, ref configuration, 4) ==
TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LOG_STATUS, ref logState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Debug operation started.");
        //... Log as needed
        logState = PCANBasic.PCAN_PARAMETER_OFF;
        if (PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LOG_STATUS, ref logState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Debug operation finished. Please check the log file.");
        }
        else
            Console.WriteLine("Error! Logging could not be disabled.");
    }
    else
        Console.WriteLine("Error! Logging could not be enabled.");
}
else
    Console.WriteLine("Error! Logging could not be configured.");
```

PCAN_LOG_TEXT

This parameter helps the user to insert custom text into the log file generated in a debug session.

Note that using this parameter starts recording debug information automatically if the logging functionality was inactive. You could include calls to this parameter in parts of your code that normally shouldn't have to be executed, so that any unwanted behavior triggers the start of a debug session (as a kind of watch dog).

Availability

It is available since version 1.0.0.

Supported By

PCAN_NONEBUS: Logging parameters are used globally, i.e., they are not tied to a specific PCAN-Channel.

Access Mode

This parameter can only be written.

Possible Values

This parameter must be a string containing the data to be inserted in the log file. There is no limit for the length of the string, but it is recommended to use a length not bigger than MAX_PATH (255 bytes).

Default Value

Does not apply. This is a value that can only be written.

Initialization Status

Does not apply. It is not necessary to have any PCAN-Channel initialized to use this parameter.

When to Use

It can be used if you want to use the log functionality for your own purposes, i.e., to debug own processes, behavior, to mark executed code places, etc.

Application – Example of Use

Let's say you are writing an application and want to include debug information of other processes being done inside of it, e.g., to log when any access violation occurs, or when the user makes any configuration changes, etc. Instead of implementing your own debug logging, you could use this parameter and so save implementation time, since this logging file works, has been tested already, and it includes already information such as when an entry was done and from which thread it was done (an exception is simulated):

Native (C++)

```
try
{
    // Do some operations and check for exceptions
    throw std::exception("An exception occurred doing certain operations");
}
catch (std::exception &ex)
{
    // Exception occurred. Log personal message
    char buffer[MAX_PATH];
    sprintf_s(buffer, MAX_PATH, "MyAPP-Exception: %s", ex.what());
    if (CAN_SetValue(PCAN_NONEBUS, PCAN_LOG_TEXT, buffer, MAX_PATH) == PCAN_ERROR_OK)
        printf("Exception logged sucessfully\n");
    else
        printf("Error! It was not possible to log own message\n");
}
```

Managed (C#)

```
try
{
    // Do some operations and check for exceptions
    throw new Exception("An exception occurred doing certain operations");
}
catch(Exception ex)
{
    // Exception occurred. Log personal message
    string message = "MyAPP-Exception: " + ex.Message;
    if (PCANBasic.SetValue(PCANBasic.PCAN_NONEBUS, TPCANParameter.PCAN_LOG_TEXT, message,
(uint)message.Length) == TPCANStatus.PCAN_ERROR_OK)
        Console.WriteLine("Exception logged sucessfully");
    else
        Console.WriteLine("Error! It was not possible to log own message");
}
```

Using Tracing Parameters

These parameters are intended to minimize the developing time and cost of CAN applications using the PCAN-Basic API, by allowing the recording and storing of all CAN communication in an ASCII formatted file that can be loaded by any text editor. Thanks to the structured stored data, it can be easily parsed into own applications too (see Appendix B, and Appendix C).

Since the trace formats are officially used by several Peak-System applications, there are already several tools that can load and process those trace files, further minimizing the investment in own software programming. For example, the information recorded can be inspected using PCAN-Explorer and can even be played back for simulation purposes using the PCAN-Trace application.

Consider that the trace functionality is available for each PCAN-Channel. This implies three important points:

- The PCAN-Channel must be first initialized before a trace session can be started.
- You can start as many trace sessions as used/initialized PCAN-Channels within your application, simultaneously.
- The data traced corresponds to the data successfully transmitted through a PCAN-Channel, using the functions CAN_ReadFD and CAN_WriteFD in case of a channel initialized as FD, or using the functions CAN_Read and CAN_Write in case a channel was initialized in normal mode. **Note** that if an application never calls those functions, then no data will be traced.

PCAN_TRACE_LOCATION

This value is used to set the folder on a computer in where the PCAN-Trace file will be stored. If a session is running (a PCAN-Trace file is being written), PCAN_TRACE_LOCATION instructs the API to close the current PCAN-Trace file and to start the process again with the new folder information.

Note that the name of the trace file cannot be freely specified. The base name of the trace file is always the name of the PCAN-Channel being used (**PCAN_USBBUS1.trc**, for example). It is only possible to enhance the name with the date and/or time of creation of the file.

Availability

It is available since version 1.3.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).

PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This value is a string containing a fully qualified and valid path to an existing directory on the executing computer. To use the default path (calling process path) an empty string must be set.

Kind of Path	Value needed
CUSTOM Path	A valid directory string (Files and Paths).
DEFAULT Path	Empty string (calling process folder).

Default Value

The default value is the path to the calling process folder.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when you want to sort trace sessions being done.

Application – Example of Use

Let's say you have an application that operates in different modes (flashing, diagnostic, custom, user, etc.). You could have a folder for each mode, so that trace files are automatically sorted by the application's mode used (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
char tracesFolder[MAX_PATH];
char fullPath[MAX_PATH];
int mode = 2;

GetTempPathA(MAX_PATH, tracesFolder);

switch (mode)
{
    // 1: Flashing
    case 1:
        sprintf_s(fullPath, MAX_PATH, "%sPCAN-Basic_Flashing", tracesFolder);
        break;
    // 2: Diagnostic
    case 2:
        sprintf_s(fullPath, MAX_PATH, "%sPCAN-Basic_Diagnostic", tracesFolder);
        break;
    // 3: Normal
    case 3:
        sprintf_s(fullPath, MAX_PATH, "%sPCAN-Basic_Normal", tracesFolder);
        break;
    // Unknown mode
    default:
        sprintf_s(fullPath, MAX_PATH, "%sPCAN-Basic_UnknownMode", tracesFolder);
        break;
}

if (CreateDirectoryA(fullPath, NULL))
{
    if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
    {
        if (CAN_SetValue(channelUsed, PCAN_TRACE_LOCATION, fullPath, MAX_PATH) == PCAN_ERROR_OK)
        {
            printf("The location for trace files was successfully set to:\n");
            printf(fullPath);
```

```
        }
        else
            printf("Error! Trace location could not be configured.\n");
    }
    else
        printf("Channel 0x%X could not be initialized\n", channelUsed);
}
else
    printf("Folder for log file could not be created at:\n%s", fullPath);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
string tracesFolder = System.IO.Path.GetTempPath();
string fullPath;
int mode = 1;

switch (mode)
{
    // 1: Flashing
    case 1:
        fullPath = System.IO.Path.Combine(tracesFolder, "PCAN-Basic_Flashing");
        break;
    // 2: Diagnostic
    case 2:
        fullPath = System.IO.Path.Combine(tracesFolder, "PCAN-Basic_Diagnostic");
        break;
    // 3: Normal
    case 3:
        fullPath = System.IO.Path.Combine(tracesFolder, "PCAN-Basic_Normal");
        break;
    // Unknwn mode
    default:
        fullPath = System.IO.Path.Combine(tracesFolder, "PCAN-Basic_UnknownMode");
        break;
}

System.IO.Directory.CreateDirectory(fullPath);
if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_LOCATION, fullPath, (uint)fullPath.Length)
== TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("The location for trace files was successfully set to:");
        Console.WriteLine(fullPath);
    }
    else
        Console.WriteLine("Error! Trace location could not be configured.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```


PCAN_TRACE_STATUS

This parameter helps the user to control the activity status of a trace session within the PCAN-Basic API.

Note that if the tracing status is set to ON without having configured a destination path for the trace file or without having configured the tracing mode, then the session process will start with the default values, that is:

- The PCAN-Trace file will be placed in the folder where the calling process is located.
- The file name to use is the name of the used PCAN-Channel (**PCAN_USBBUS1.trc**, for example).
- Existent files will not be overwritten, i.e., starting the trace process will fail.
- The API will create one PCAN-Trace file and will fill it with data until the file reaches a size of **10 megabytes**.

Important Note: For messages to be written in the trace file, the receive queue must be read actively, even if the application is only sending. Transmitted messages are also synchronized over the reception queue.

Availability

It is available since version 1.3.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be activated or deactivated.

Defined Value	Description
PCAN_PARAMETER_OFF	The Tracing is OFF.
PCAN_PARAMETER_ON	The Tracing is ON.

Default Value

The default value of the Tracing mode is deactivated (PCAN_PARAMETER_OFF). After activating it, the tracing functionality stays active until one of these possibilities happens:

- The tracing session is expressly deactivated.
- The used PCAN-Channel is disconnected (e.g., using the function CAN_Uninitialize).
- The configuration of the tracing session instructs the API to stop tracing (e.g., the maximum size for a trace file is reached).

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to control a tracing session (start, stop, restart, etc.).

Application – Example of Use

Let's say you want to allow the user of your application to decide when data should be traced. You could allow this by simply invoking this parameter through a function that a user could trigger using a button click (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD traceState = PCAN_PARAMETER_ON;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Do work, and when needed, activate the trace functionality
    if (CAN_SetValue(channelUsed, PCAN_TRACE_STATUS, &traceState, 4) == PCAN_ERROR_OK)
    {
        printf("Trace session started successfully.\n");
        //... trace as needed, and when finish, deactivate the trace functionality
        traceState = PCAN_PARAMETER_OFF;
            if (CAN_SetValue(channelUsed, PCAN_TRACE_STATUS, &traceState, 4) == PCAN_ERROR_OK)
            {
                printf("Trace session finished.\n");
            }
```

```
            else
                printf("Error! The trace session could not be stopped.\n");
        }
        else
            printf("Error! The trace session could not be started.\n");
    }
    else
        printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint traceState = PCANBasic.PCAN_PARAMETER_ON;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Do work, and when needed, activate the trace functionality
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_STATUS, ref traceState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Trace session started successfully.");
        //... trace as needed, and when finish, deactivate the trace functionality
        traceState = PCANBasic.PCAN_PARAMETER_OFF;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_STATUS, ref traceState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Trace session finished.");
        }
        else
            Console.WriteLine("Error! The trace session could not be stopped.");
    }
    else
        Console.WriteLine("Error! The trace session could not be started.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```


PCAN_TRACE_SIZE

This parameter is used to set the maximum size in megabytes that a single PCAN-Trace file can have. **Note** that trying to set the size for a file will fail if a tracing session is active.

Availability

It is available since version 1.3.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).
PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This value is an integer representing the number of megabytes a file can store. To use the default size (10 megabytes) the value of 0 must be set.

Kind of Size	Valid Value
CUSTOM Size	A value between 1 and 100 megabytes.
DEFAULT Size	A value of 0 (defaults to 10 megabytes).

Default Value

The default size value is 10 Megabytes. This allows to record about 166.000~ CAN messages (Standard frames, with 8 data bytes).

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used to control the amount of data to be stored in a single file. According with the tracing configuration, this parameter can be used to automatically stop a trace session (e.g., to record data until a given limit is reached).

Application – Example of Use

Let's say you want to allow the user of your application to decide how big a trace should be. You could allow this by simply invoking this parameter through a function that a user could trigger using a button-click (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD sizeToSet = 20;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Set the desired size for a trace file, in this sample 20 MB
    if (CAN_SetValue(channelUsed, PCAN_TRACE_SIZE, &sizeToSet, 4) == PCAN_ERROR_OK)
    {
        printf("Trace size set successfully. New size is %d MB\n", sizeToSet);
    }
    else
        printf("Error! The size for the trace could not be set.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint sizeToSet = 20;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Set the desired size for a trace file, in this sample 20 MB
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_SIZE, ref sizeToSet, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Trace size set successfully. New size is {0} MB", sizeToSet);
    }
    else
        Console.WriteLine("Error! The size for the trace could not be set.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_TRACE_CONFIGURE

This parameter is used to configure the trace process and the file generated in a trace session. **Note** that trying to configure the trace process will fail if a tracing session is active.

Availability

It is available since version 1.3.0.

Supported By

PCAN-ISA (Channels PCAN_ISABUS1 to PCAN_ISABUS8).
PCAN-DNG (Channel PCAN_DNGBUS1).

PCAN-PCI (Channels PCAN_PCIBUS1 to PCANPCIBUS16).
PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).
PCAN-PCC (Channels PCAN_PCCBUS1 to PCAN_PCCBUS2).
PCAN-LAN (Channels PCAN_LANBUS1 to PCAN_LANBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter can be configured with one of the following values or a combination of those:

Defined Value	Description
TRACE_FILE_SINGLE	A trace session is stored in a single and stays active until the file reaches the maximum configured file size, or it is deactivated, or the PCAN-Channel used is disconnected.
TRACE_FILE_SEGMENTED	A trace session is stored in several files. A new file is created when a previous file reaches the maximum configured size. The tracing session stays active until it is deactivated, or the PCAN-Channel used is disconnected.
TRACE_FILE_DATE	The name of the trace file also includes the start-date of the tracing session. The date is expressed using 8 digits with the form **YYYYMMDD**, where YYYY are four digits for the year, MM two digits for the month, and DD two digits for the day, e.g., "**20130228**_PCAN_USBBUS1.trc" for the 28th of February 2013. If both, TRACE_FILE_DATE and TRACE_FILE_TIME are configured, the file name starts always with the date: "**20130228**140733_PCAN_USBBUS1_1.trc".
TRACE_FILE_TIME	The name of the trace file also includes the start-time of the tracing session. The time is expressed using 6 digits with the form **HHMMSS**, where HH are two digits for the hour in 24 hours format, MM two digits for the minutes, and SS two digits for the seconds, e.g., "**140733**_PCAN_USBBUS1.trc" for the 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: "20130228**140733**_PCAN_USBBUS1_1.trc".
TRACE_FILE_OVERWRITE	It causes the overwriting of an existence 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

The default value of this parameter is TRACE_FILE_SINGLE, which means a single file is created and filled out until the maximum configured file size is reached.

Note that the name of the file to use is the name of the PCAN-Channel being traced (e.g., PCAN_USBBUS1.trc). If a file with the same name already exists, then the activation of the tracing session will fail.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when the trace behavior desired is something other than the default behavior.

Application – Example of Use

Let's say you want to trace CAN data, but you don't know how many bytes you will trace, or you know that the trace information will be more than the maximum file size allowed (100 megabytes). You could configure the trace process to use several files (segmentation) so that the only limit is the storing unit used. In this way the application stays tracing data in different files until you stop the process or an error on file creation occurs (it is assumed, a USB channel is connected):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD traceState = PCAN_PARAMETER_ON;
DWORD sizeToSet = 20;
DWORD configuration;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Set the desired size for a trace file, in this sample 20 MB
    if (CAN_SetValue(channelUsed, PCAN_TRACE_SIZE, &sizeToSet, 4) == PCAN_ERROR_OK)
    {
        printf("Trace size set to %d MB\n", sizeToSet);
        // Configure the trace to save data in several files and to use the
        // file creation time as part of the file name
        configuration = TRACE_FILE_SEGMENTED | TRACE_FILE_TIME;
        if (CAN_SetValue(channelUsed, PCAN_TRACE_CONFIGURE, &configuration, 4) == PCAN_ERROR_OK)
        {
            printf("Trace configured successfully. Value: 0x%X\n", configuration);
            if (CAN_SetValue(channelUsed, PCAN_TRACE_STATUS, &traceState, 4) == PCAN_ERROR_OK)
            {
                printf("Trace session started on channel 0x%X\n", channelUsed);
            }
            else
                printf("Error! The trace session could not be started.\n");
        }
        else
            printf("Error! The trace could not be configured.\n");
    }
    else
        printf("Error! The size for the trace could not be set.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint traceState = PCANBasic.PCAN_PARAMETER_ON;
uint sizeToSet = 20;
uint configuration;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Set the desired size for a trace file, in this sample 20 MB
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_SIZE, ref sizeToSet, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Trace size set to {0} MB", sizeToSet);
        // Configure the trace to save data in several files and to use the
        // file creation time (e.g., 100712 for 10:07:12) as part of the file name
        configuration = PCANBasic.TRACE_FILE_SEGMENTED | PCANBasic.TRACE_FILE_TIME;
        if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_CONFIGURE, ref configuration, 4) ==
TPCANStatus.PCAN_ERROR_OK)
        {
            Console.WriteLine("Trace configured successfully. Value: 0x{0:X} ", configuration);
            if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_TRACE_STATUS, ref traceState, 4) ==
TPCANStatus.PCAN_ERROR_OK)
            {
                Console.WriteLine("Trace session started on channel 0x{0:X}", channelUsed);
            }
            else
                Console.WriteLine("Error! The trace session could not be started.");
        }
        else
            Console.WriteLine("Error! The trace could not be configured.");
    }
```

```
    else
        Console.WriteLine("Error! The size for the trace could not be set.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

Using Electronic Circuits Parameters

These parameters are intended to condense features. Some CAN devices are equipped with pins for digital and analog signals, that make providing electronic circuits with I/O functionality possible. Instead of offering separate APIs for this, the I/O features are accessible as parameters, over the functions CAN_GetValue/CAN_SetValue.

At the time of writing this documentation, only the PCAN-Chip USB module offers I/O capabilities in form of 5 digital input pins, that also can be configured as digital outputs, and one analog input pin.

PCAN_IO_DIGITAL_CONFIGURATION

This parameter is used to configure the output mode of **all** digital Input / Output pins available on a device. It allows the configuration of up to 32 pins, as a bit mask value.

Note that at the time of writing this documentation only PCAN-Chip USB based devices, with a firmware version equal to or greater than 3.3.0, support the configuration of a maximum of 5 digital pins. For this reason, all other (unused) bits are automatically discarded from the bit mask value passed as parameter. No error is generated by setting a bit for a nonexistent pin.

Availability

It is available since version 4.3.0.

Supported By

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter has a double-word resolution, which allows a value range between [0... 4294967295]. It is a **bit mask,** in which every bit represents a digital input.
Each digital input pin of the device can be set as digital output.

Bit Value	Description
0	The pin works as digital input.
1	The pin works as digital input **and** output.

Bit0, the least significant bit, corresponds to Pin0; Bit1 corresponds to Pin1, and so on until Pin31.

Default Value

The default value for each pin (and so for the whole mask) is 0, meaning that all pins are configured as digital input only (no digital outputs active).

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when you want to provide electronic circuits with digital I/O functionality from a CAN environment.

Application – Example of Use

Let's say you have an electronic unit with some LEDs. You could configure the digital pins as digital outputs so that you can light them up or turn them off from your CAN application (it is assumed a PCAN-Chip USB, connected as PCAN_USBBUS1 channel, is used):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD configuration;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // Flag value for configuring digital outputs from Pin0 to Pin4
    configuration = 0x1F;
    if (CAN_SetValue(channelUsed, PCAN_IO_DIGITAL_CONFIGURATION, &configuration, 4) == PCAN_ERROR_OK)
    {
        printf("All digital Pins (0 to 4) successfully configured as digital outputs.n");
    }
    else
        printf("Error! The Pins could not be configured.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint configuration;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // Flag value for configuring digital outputs from Pin0 to Pin4
    configuration = 0x1F;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_IO_DIGITAL_CONFIGURATION, ref configuration, 4)
== TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("All digital Pins (0 to 4) successfully configured as digital outputs.");
    }
    else
        Console.WriteLine("Error! The Pins could not be configured.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_IO_DIGITAL_VALUE

This parameter is used to set the output values represented by the digital pins available on a device, as a bit mask value. Unlike PCAN_IO_DIGITAL_SET and PCAN_IO_DIGITAL_CLEAR, this operation applies to **all** pins, i.e., each available pin is set to one of the two possible states.

Note that the bit mask allows setting the value of 32 pins, though, at the time of writing this documentation only 5 digital pins are supported. For this reason, all other (not used) bits are automatically discarded from the bit mask value passed as parameter. No error is generated by setting a bit for a nonexistent pin.

Availability

It is available since version 4.3.0.

Supported By

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is read/write. It can be set and read.

Possible Values

This parameter has a double-word resolution, which allows a value range between [0... 4294967295]. It is a **bit mask,** in which every bit represents the value for a digital output pin. The value of each digital pin of the device can be set to “low” or “high”.

Bit Value	Description
0	The digital pin value is “Low”.
1	The digital pin value is “High”.

Bit0, the least significant bit, corresponds to the value of Pin0; Bit1 corresponds to the value of Pin1, and so on until Pin31.

Default Value

The default value for each pin (and for the whole mask) is 0, meaning that the values of all digital output pins are set to “Low”.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when you want to provide electronic circuits with digital I/O functionality from a CAN environment.

Application – Example of Use

Let’s say you have an electronic unit with some LEDs. You could set digital pins to “High”, (for instance, Pin0 and Pin4) so that connected LEDs turn on (it is assumed a PCAN-Chip USB, connected as PCAN_USBBUS1 channel, is used):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD ledActivation;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // The Pin0 and Pin4 are set to high, all other Pins to "low"
    ledActivation = 0x11;
    if (CAN_SetValue(channelUsed, PCAN_IO_DIGITAL_VALUE, &ledActivation, 4) == PCAN_ERROR_OK)
    {
        printf("Pin0 and Pin4 were successfully set to 'high'.\n");
        printf("Pin1, Pin2, and Pin3 were successfully set to 'low'.\n");
    }
    else
        printf("Error! The Pin states could not be changed.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint ledActivation;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
```

```
    // The Pin0 and Pin4 are set to high, all other Pins to "low"
    ledActivation = 0x11;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_IO_DIGITAL_VALUE, ref ledActivation, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Pin0 and Pin4 were successfully set to 'high'.");
        Console.WriteLine("Pin1, Pin2, and Pin3 were successfully set to 'low'. ");
    }
    else
        Console.WriteLine("Error! The Pin states could not be changed.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```


PCAN_IO_DIGITAL_SET

This parameter is used to configure the value of **selected** digital Output pins to “High” within a device, using a bit mask value. Unlike PCAN_IO_DIGITAL_VALUE, only needed pins are set to “High”; unwanted ones are not touched, i.e. they are not re-configured.

Note that at the time of writing this documentation only PCAN-Chip USB based devices, with a firmware version equal to or greater than 3.3.0, support the configuration of a maximum of 5 digital pins. For this reason, all other (not used) bits are automatically discarded from the bit mask value passed as parameter. No error is generated by setting a bit for a nonexistent pin.

Availability

It is available since version 4.3.0.

Supported By

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is write-only. It can only be set.

Possible Values

This parameter has a double-word resolution, which allows a value range between [0... 4294967295]. It is a **bit mask,** in which every bit represents a digital pin.

Bit Value	Description
0	The digital pin at this position is ignored.
1	The value of the digital pin at this position is set to “High”.

Bit0, the least significant bit, corresponds to Pin0; Bit1 corresponds to Pin1, and so on until Pin31.

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when you want to provide electronic circuits with digital I/O functionality from a CAN environment.

Application – Example of Use

Let's say you have an electronic unit with 5 LEDs. You have already turned 2 of them on (Pin0 and Pin4). You could use this parameter to turn on another one (for instance, Pin2), without having to set again the other LEDs (it is assumed a PCAN-Chip USB, connected as PCAN_USBBUS1 channel, is used):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD ledActivation;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // The Pin2 is set to "high". Other Pins remain unchanged
    ledActivation = 0x4;
    if (CAN_SetValue(channelUsed, PCAN_IO_DIGITAL_SET, &ledActivation, 4) == PCAN_ERROR_OK)
    {
        printf("Pin2 was successfully set to 'high'.\n");
    }
    else
        printf("Error! The state of Pin2 could not be changed.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint ledActivation;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    // The Pin2 is set to "high". Other Pins remain unchanged
    ledActivation = 0x4;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_IO_DIGITAL_SET, ref ledActivation, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Pin2 was successfully set to 'high'.");
    }
    else
        Console.WriteLine("Error! The state of Pin2 could not be changed.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_IO_DIGITAL_CLEAR

This parameter is used to configure the value of **selected** digital Output pins to "Low" within a device, using a bit mask value. Unlike PCAN_IO_DIGITAL_VALUE, only needed pins are set to "Low"; unwanted ones are not touched, i.e., they are not re-configured.

Note that at the time of writing this documentation only PCAN-Chip USB based devices, with a firmware version equal to or greater than 3.3.0, support the configuration of a maximum of 5 digital pins. For this reason, all other (not used) bits are automatically discarded from the bit mask value passed as parameter. No error is generated by setting a bit for a nonexistent pin.

Availability

It is available since version 4.3.0.

Supported By

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is write-only. It can only be set.

Possible Values

This parameter has a double-word resolution, which allows a value range between [0... 4294967295]. It is a **bit mask,** in which every bit represents a digital pin.

Bit Value	Description
0	The digital pin at this position is ignored.
1	The value of the digital pin at this position is set to “Low”.

Bit0, the least significant bit, corresponds to Pin0; Bit1 corresponds to Pin1, and so on until Pin31.

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when you want to provide electronic circuits with digital I/O functionality from a CAN environment.

Application – Example of Use

Let’s say you have an electronic unit with 5 LEDs. You have already turned 3 of them on (Pin0, Pin2, and Pin4). You could use this parameter to turn off one of those (for instance Pin0), without having to expressly set the remaining four again, two LEDs on and two LEDs off, respectively (it is assumed a PCAN-Chip USB, connected as PCAN_USBBUS1 channel, is used):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD ledClearing;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    // The Pin0 is set to "low". Other Pins remain unchanged
    ledClearing = 0x1;
    if (CAN_SetValue(channelUsed, PCAN_IO_DIGITAL_CLEAR, &ledClearing, 4) == PCAN_ERROR_OK)
    {
        printf("Pin0 was successfully set to 'low'.\n");
    }
    else
        printf("Error! The state of Pin0 could not be changed.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint ledClearing;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
```

```
    // The Pin0 is set to "low". Other Pins remain unchanged
    ledClearing = 0x1;
    if (PCANBasic.SetValue(channelUsed, TPCANParameter.PCAN_IO_DIGITAL_CLEAR, ref ledClearing, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Pin0 was successfully set to 'low'.");
    }
    else
        Console.WriteLine("Error! The state of Pin0 could not be changed.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

PCAN_IO_ANALOG_VALUE

This parameter is used for reading analog voltages from the analog input pin of a device.

Note that at the time of writing this documentation only PCAN-Chip USB based devices, with a firmware version equal to or greater than 3.3.0, support only 1 analog pin.

Availability

It is available since version 4.3.0.

Supported By

PCAN-USB (Channels PCAN_USBBUS1 to PCAN_USBBUS16).

Access Mode

This parameter is read only. It cannot be written.

Possible Values

This parameter has a double-word resolution, which allows a value range between [0... 4294967295]. The returned value represents the direct value from the A/D converter, which is an unsigned integer value.

The returned value must be converted into a signed value, considering the external wiring.

Default Value

Does not apply.

Initialization Status

The PCAN-Channel must be initialized before using this parameter.

When to Use

It can be used when you want to provide electronic circuits with digital I/O functionality from a CAN environment.

Application – Example of Use

Let's say you have an electronic unit with a potentiometer. You could read the state of it and present the calculated value on your application or take some decisions according on the current value (it is assumed a PCAN-Chip USB, connected as PCAN_USBBUS1 channel, is used):

Native (C++)

```
TPCANHandle channelUsed = PCAN_USBBUS1;
DWORD analogValue;

if (CAN_Initialize(channelUsed, PCAN_BAUD_500K) == PCAN_ERROR_OK)
{
    if (CAN_GetValue(channelUsed, PCAN_IO_ANALOG_VALUE, &analogValue, 4) == PCAN_ERROR_OK)
```

```
    {
        printf("Raw value of analog Pin is %d\n", analogValue);
    }
    else
        printf("Error! The value of the analog Pin could not be read.\n");
}
else
    printf("Channel 0x%X could not be initialized\n", channelUsed);
```

Managed (C#)

```
ushort channelUsed = PCANBasic.PCAN_USBBUS1;
uint analogValue;

if (PCANBasic.Initialize(channelUsed, TPCANBaudrate.PCAN_BAUD_500K) == TPCANStatus.PCAN_ERROR_OK)
{
    if (PCANBasic.GetValue(channelUsed, TPCANParameter.PCAN_IO_ANALOG_VALUE, out analogValue, 4) ==
TPCANStatus.PCAN_ERROR_OK)
    {
        Console.WriteLine("Raw value of analog Pin is {0}", analogValue);
    }
    else
        Console.WriteLine("Error! The value of the analog pin could not be read.");
}
else
    Console.WriteLine("Channel 0x{0:X} could not be initialized", channelUsed);
```

Appendix A: Debug-log over Registry

These steps will guide you activating/deactivating the Logging functionality of PCAN-Basic using the registry of Windows.

Activating a Log Session

1. Stop all applications using the PCAN-Basic.
2. Open the Windows's Registry (e.g. using the Windows Start menu / "Execute…" and typing "**regedit**").
3. Create the following registry key under the [**HKEY_CURRENT_USER**] hive:
   \Software\PEAK-System\PCAN-Basic\Log
4. To specify the data to be logged, add a new **DWORD** value to the key created before, and call it "**Flags**".
5. Sets the value for "**Flags**" according to your needs. This value is the numerical value of any LOG_FUNCTION_* define or a logic-OR combination of them.
6. To specify the directory where the log file should be created, add a new **STRING** value to the key created before, and call it "**Path**".
7. Sets the value for "**Path**" with the full path to the directory you want.

At this point, starting any application that use the PCAN-Basic API will cause the automatic generation of a debug session.

Deactivating a Log Session

1. Stop all applications using the PCAN-Basic.
2. Open the Windows's Registry (e.g. using the Windows Start menu / "Execute…" and typing "**regedit**").
3. Locate the registry hive [**HKEY_CURRENT_USER**].
4. Search for the following registry key:
   \Software\PEAK-System\PCAN-Basic\Log
5. Delete the key and its values.

At this point, starting any application that use the PCAN-Basic will not cause logging operations anymore.

VERY IMPORTANT NOTE

Please don't forget to delete the created key after your debug session is done. If you leave the key, all PCAN-Basic applications running under your Windows account will remain writing data to their log files, generating in this way huge text files that consume hard-disk space unnecessarily.

Appendix B: PCAN-Trace Format 1.1

The PCAN-Basic API uses the PCAN-Trace format 1.1 for channels with normal CAN (non FD), which is used by PCAN-Explorer 3.0.2, PCAN-Explorer 4, PCAN-Trace 1.5, PCAN-View 3, and the Peak-Converter 1. This format is used for channels initialized in “normal mode”, that is channels initialized using the function CAN_Initialize, doing communication over the functions CAN_Read and CAN_Write.

Example

```
;$FILEVERSION=1.1
;$STARTTIME=37704.5364870833
;
;   C:\TraceFile.trc
;
;   Start time: 24.03.2003 12:52:32.484
;   PCAN-Net: TestNet
;
;   Columns description:
;   ~~~~~~~~~~~~~~~~~~~~
;   +-Message Number
;   |          +Time Offset (ms)
;   |          |         +Type
;   |          |         |          +ID (hex)
;   |          |         |          |      +Data Length Code
;   |          |         |          |      |   +Data Bytes (hex) ...
;   |          |         |          |      |   |
;---+--   ----+----  --+--  ----+---  +   -+ -- -- -- -- -- -- --
     1)       1059.9  Rx          0300  7   00 00 00 00 04 00 00
     2)       1283.2  Rx          0300  7   00 00 00 00 04 00 00
     3)       1298.9  Tx          0400  2   00 00
     4)       1323.0  Rx          0300  7   00 00 00 00 06 00 00
     5)       1346.8  Warng   FFFFFFFF  4   00 00 00 04  BUSLIGHT
     6)       1349.2  Error       0008  4   00 19 08 08
```

Description

File Coding:

The Trace file is ASCII coded.

Comment Lines:

Lines prefixed with a Semicolon are “Comments” and are ignored while loading Trace files, except for $-Keywords.

$-Keywords:

These are defined informations that gives different information about the Trace file. They appear as a comment line. Possible keywords are:

- $FILEVERSION: contains the major and minor version of the file format, i.e. “1.1” for this version.
- $STARTIME: contains the absolute start time of the trace file:
  - Format: Floating point, point as decimal separator.
  - Value: the integral part represents the number of days that have passed since 30th December of 1899. The fractional part, the fraction of a 24 hour day that has elapsed, resolution is 1 millisecond.

Columns:

The information contained in a Trace file is accommodated within 5 columns:

- Message Number: Index of a recorded message (ignored while loading the trace file).
- Time Offset (ms): Time offset since start of the trace session. The time has a resolution of 1/10 milliseconds.
    - Format: Floating point, point as decimal separator.
    - Value: the integral part represents the milliseconds offset. The fractional part is 1/10 milliseconds (1 digit).
- Type: Represents the kind of message recorded. Possible message types are:
    - “Rx”: Message was received (in PCAN-Basic, using the function CAN_Read).
    - “Tx”: Message was sent (in PCAN-Basic, using the function CAN_Write).
    - “Warng”: Message represents a received Warning-Frame.
    - “Error”: Message represents an Error-Frame.
- ID (hex): Represents the CAN-ID in hexadecimal notation. Possible values are:
    - 4 digits for 11-bit CAN-IDs (0000-07FF).
    - 8 digits for 29-bit CAN-IDs (00000000-1FFFFFFF).
    - Special case: “FFFFFFFF” for Warning-Frames.
- Data Length Code: It is a number between 0-8 representing the amount of data contained within the message recorded.
- Data Bytes (hex): represents the data of a recorded message. According with the message type, the data can be:
    - If the message represents common CAN data: so many data bytes, in hexadecimal notation, as the Data Length Code indicates.
    - If the message represents a remote request frame: “RTR”
    - If the message represents a Warning-Frame: 4 data bytes expressed in hexadecimal notation, using Motorola format. At the end of this line, the short name of the Warning (ignored while loading the Trace file). Example: “00 00 00 04 BUSLIGHT”.
    - If the message represents an Error-Frame: 4 data bytes expressed in hexadecimal notation.

Appendix C: PCAN-Trace Format 2.0

The PCAN-Basic API uses the PCAN-Trace format 2.0 for channels with FD capabilities (CAN-FD), which is used by PCAN-View 4, PEAK-Converter 2, and PCAN-Explorer 6. This format is used for channels initialized in "FD mode", that is channels initialized using the function CAN_InitializeFD, doing communication via the functions CAN_ReadFD and CAN_WriteFD.

Example

```
;$FILEVERSION=2.0
;$STARTTIME=41736.3716038773
;$COLUMNS=N,O,T,B,I,d,R,l,D
;
;   C:\TraceFile.trc
;   Start time: 07.04.2014 08:55:06.575.0
;   Generated by PCAN-View v4.0.7.340
;-------------------------------------------------------------------------------
;   Bus  Name         Connection             Protocol
;   1    Connection1  TestNet@pcan_usb       CAN
;-------------------------------------------------------------------------------
;   Message   Time    Type     ID     Rx/Tx
;   Number    Offset  |  Bus   [hex]  |  Reserved
;   |         [ms]    |  |     |      |  |  Data Length
;   |         |       |  |     |      |  |  |  Data [hex] ...
;   |         |       |  |     |      |  |  |  |
;---+-- ------+------ +- +- --+----- +- +- +- +- -- -- -- -- -- -- --
      1      1059.900 DT 1      0300 Rx -  7  00 00 00 00 04 00 00
      2      1283.231 DT 1      0300 Rx -  7  00 00 00 00 04 00 00
      3      1298.945 DT 1      0400 Tx -  2  00 00
      4      1323.201 DT 1      0300 Rx -  7  00 00 00 00 06 00 00
      5      1334.416 FD 1      0500 Tx -  12 01 02 03 04 05 06 07 08 09 0A 0B 0C
      6      1334.222 ER 1           Rx -     04 00 02 00 00
      7      1334.224 EV User-defined event text
      8      1334.231 ST 1           Rx -     00 00 00 08
      9      1334.268 ER 1           Rx -     04 00 02 08 00
```

Description

File Coding:

The Trace file is ASCII coded.

Comment Lines:

Lines prefixed with a Semicolon are "Comments" and are ignored while loading Trace files, except for $-Keywords.

$-Keywords:

These are defined informations that gives different information about the Trace file. They appear as a comment line. Possible keywords are:

- $FILEVERSION: contains the major and minor version of the file format, i.e. "**2.0**" for this version.
- $STARTIME: contains the absolute start time of the trace file:
  - Format: Floating point, point as decimal separator.
  - Value: the integral part represents the number of days that have passed since 30th December of 1899. The fractional part, the fraction of a 24 hour day that has elapsed, resolution is 1 millisecond.
- $COLUMNS: represents the columns contains the trace file. The column order cannot be changed. But some columns are optional. The obligatory order is as follow (optional columns are enclosed in square brackets): [N],O,[B],T,I,d,[R],l/L,D.

Columns:

The information contained in a Trace file is accommodated within 10 columns, though some of them are optional:

- N: Message number, index of recorded message. Optional.
- O: Time offset since start of the trace. Resolution: 1 microsecond.
  The value before the decimal separator represents milliseconds. The value behind the decimal separator represents microseconds (3 digits).
- B: Bus (1-16). Optional.
- T: Time of message:
  - DT: CAN or J1939 data frame.
  - FD: CAN FD data frame.
  - FB: CAN FD data frame with BRS bit set (Bit Rate Switch).
  - FE: CAN FD data frame with ESI bit set (Error State Indicator).
  - BI: CAN FD data frame with both bits set, BRS and ESI.
  - RR: Remote Request frame.
  - ST: Hardware status change.
  - ER: Error frame.
  - EV: Event. User-defined text. Begins directly after 2-digit type indicator.
- I: CAN-ID (Hex):
  - 4 digits for 11-bit CAN-IDs (0000-07FF).
  - 8 digits for 29-bit CAN-IDs (00000000-1FFFFFFF).
- d: Direction: Indicates whether the message was received (‘Rx’) or transmitted (‘Tx’).
- R: Reserved. Only used for J1939 protocol. Contains ‘-’ for CAN buses. For J1939 protocol, contains destination address of a transport protocol PDU2- large message. Optional for files that contain only CAN or CAN FD frames.
- l: Data Length (0-1785). This is the real number of data bytes, not the Data Length Code (0..15). Optional. If omitted, the Data Length Code column (‘L’) must be included.
- L: Data Length Code (0-15). Optional. If omitted, the Data Length (‘l’) must be included.
- D: Data. 0-1785 data bytes in hexadecimal notation.

Appendix D: Acceptance Code and Mask Calculation

An acceptance filter is composed of an *acceptance code* and an *acceptance mask*. These values are used to set an 11-bit acceptance filter (using the parameter PCAN_ACCEPTANCE_FILTER_11BIT), or a 29-bit acceptance filter (using the parameter PCAN_ACCEPTANCE_FILTER_29BIT), depending on the needs you may have in your application. The way how the code and mask values are calculated is the same, regardless if the IDs are 11-bit or 29-bit.

Take into account that PCAN Hardware filtering is based on the SJA1000 CAN controller, which uses only one acceptance filter for both, standard (11-bit) and extended IDs (29-bit). Though it is allowed, mixing of 11-bit and 29-bit filters is not advisable.

As example, the acceptance filter for the standard IDs (11-bit) 101h, 401h, and 501h, will calculated:

Code

The acceptance code is a value resulting after applying a logical AND operation between all IDs wanted to be received.

```
      15                   0
101h: 0000 0001 0000 0001 b
401h: 0000 0100 0000 0001 b
501h: 0000 0101 0000 0001 b
--------------------------- AND
      0000 0000 0000 0001 => Code=1h
```

Mask

The acceptance mask is a value resulting after applying a ***kind of*** logical exclusive OR (XOR) between all IDs wanted to be received, meaning, that only one difference between two bits within the wanted IDs is enough to satisfy the XOR condition and to mark that bit as “don’t care bit” (The “don’t care bit” value is ‘**1**’):

```
      15                   0
101h: 0000 0001 0000 0001 b
401h: 0000 0100 0000 0001 b
501h: 0000 0101 0000 0001 b
--------------------------- XOR
      0000 0101 0000 0000 => Mask=500h
```

Note that, even when using an acceptance filter, it is possible to still receive unwanted messages. For instance, in the example above the standard ID 1h could also be received.

More information about SJA1000 acceptance filter can be found in the SJA1000 specifications document.