dSPACE CAN API 2.0

C Reference

For dSPACE CAN API Package 4.0.6

Release 2021-A – May 2021

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication and the contents hereof are subject to change without notice.

Contents

About this Reference

Content

The reference gives you detailed information on the API functions, data structures, and error codes of the dSPACE CAN API 2.0.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
⍰	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
🕮	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the 📄 icon in dSPACE Help. The PDF opens on the first page.

Introduction to the dSPACE CAN API 2.0

Where to go from here

Information in this section

Basics of the dSPACE CAN API 2.0

Introduction

The dSPACE CAN API 2.0 lets you program custom applications for CAN interfaces from dSPACE, Eberspächer GmbH, Kvaser, and Vector Informatik GmbH.

Supported CAN interfaces

Currently, the dSPACE CAN API 2.0 supports the following CAN interfaces:

- dSPACE CAN interfaces:
  - DCI-CAN2
  - DCI-CAN/LIN1
- CAN interfaces from Eberspächer GmbH

  For a detailed list, refer to CAN Interface Names on page 40.
- CAN interfaces from Kvaser

  For a detailed list, refer to CAN Interface Names on page 40.
- CAN interfaces from Vector Informatik GmbH

  For a detailed list, refer to CAN Interface Names on page 40.
- 2 virtual CAN channels for testing purposes.

Support of CAN FD

The dSPACE CAN API 2.0 supports CAN FD.

For basics on CAN FD, refer to Basics on CAN FD on page 16.

Accessing the dSPACE CAN API 2.0

The dSPACE CAN API 2.0 is a functional Windows DLL (32 and 64 bit).

Access via applications written in C, C++, or C# The dSPACE CAN API 2.0 interfaces are provided via export C functions, which you can use with your application written in C or C++.

To use the dSPACE CAN API 2.0 in a C# application, you have to reference the dSPACE CAN API 2.0 .NET DLL `DsCanApi20DotNet.dll`. The `DsCanApi20DotNet.dll` is part of the installation of the dSPACE CAN API 2.0.

Access via Python scripts To use the dSPACE CAN API 2.0 in a Python script, you have to import the dSPACE CAN API 2.0 Python module `dscanapi20lib`. The `dscanapi20lib` Python module is part of the installation of the dSPACE CAN API 2.0.

Multi-application and multiclient support

The dSPACE CAN API 2.0 supports both multi-application and multiclient.

Multi-application support Multi-application support means that several application processes can connect to the same CAN interface channel.

Multiclient support Multiclient support means that several clients can connect to the same CAN interface channel.

The demo for getting started with dSPACE CAN API 2.0 shows an example multiclient scenario. Refer to Steps Shown in the Demo on page 20.

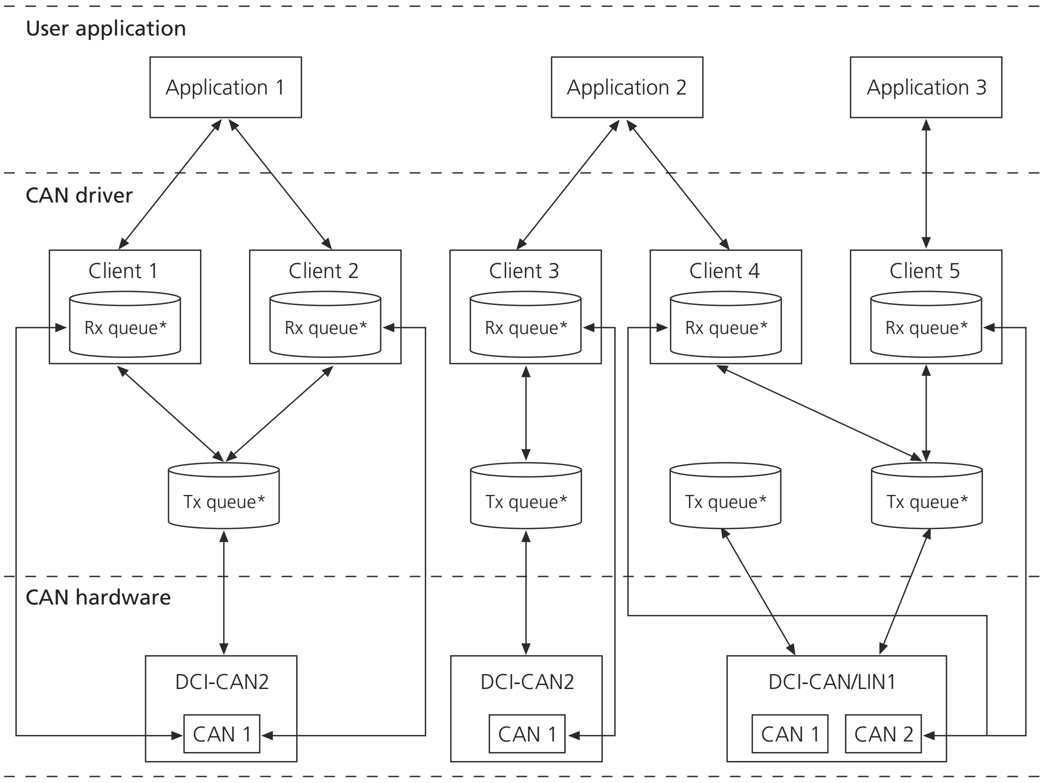
Note

- In a multi-application and/or multiclient scenario, the access to a specific CAN interface channel is based on the collaborative access permission assignment: A client cannot configure a CAN interface channel until the CAN interface channel returned access permission to the client.

  To initialize a CAN channel and get access permission to it, use the `DSCAN_InitChannel` function.

Example scenario The following illustration shows a possible scenario of multi-application and multiclient operation with the dSPACE CAN API 2.0.

- Client 1 and client 2 both connect to the same CAN interface channel (multiclient).
- Client 4 (of application 2) and client 5 (of application 3) both connect to the same CAN interface channel (multiclient and multi-application).



* Each RX/TX queue can store 32767 classic CAN messages. The number of CAN FD messages per queue is lower and depends on the message payload.

dSPACE CAN API 2.0 versus dSPACE CAN API 1.0

dSPACE CAN API 2.0 was introduced with dSPACE Release 2016-B. It is the successor of dSPACE CAN API 1.0, includes all previous features, and additionally supports CAN FD.

As of dSPACE Release 2020-A, dSPACE CAN API 1.0 is no longer supported.

Information on installation and licensing

For information on installing dSPACE software and handling dSPACE licenses, refer to What Do You Want To Do? (Installing dSPACE Software 🕮).

Related topics

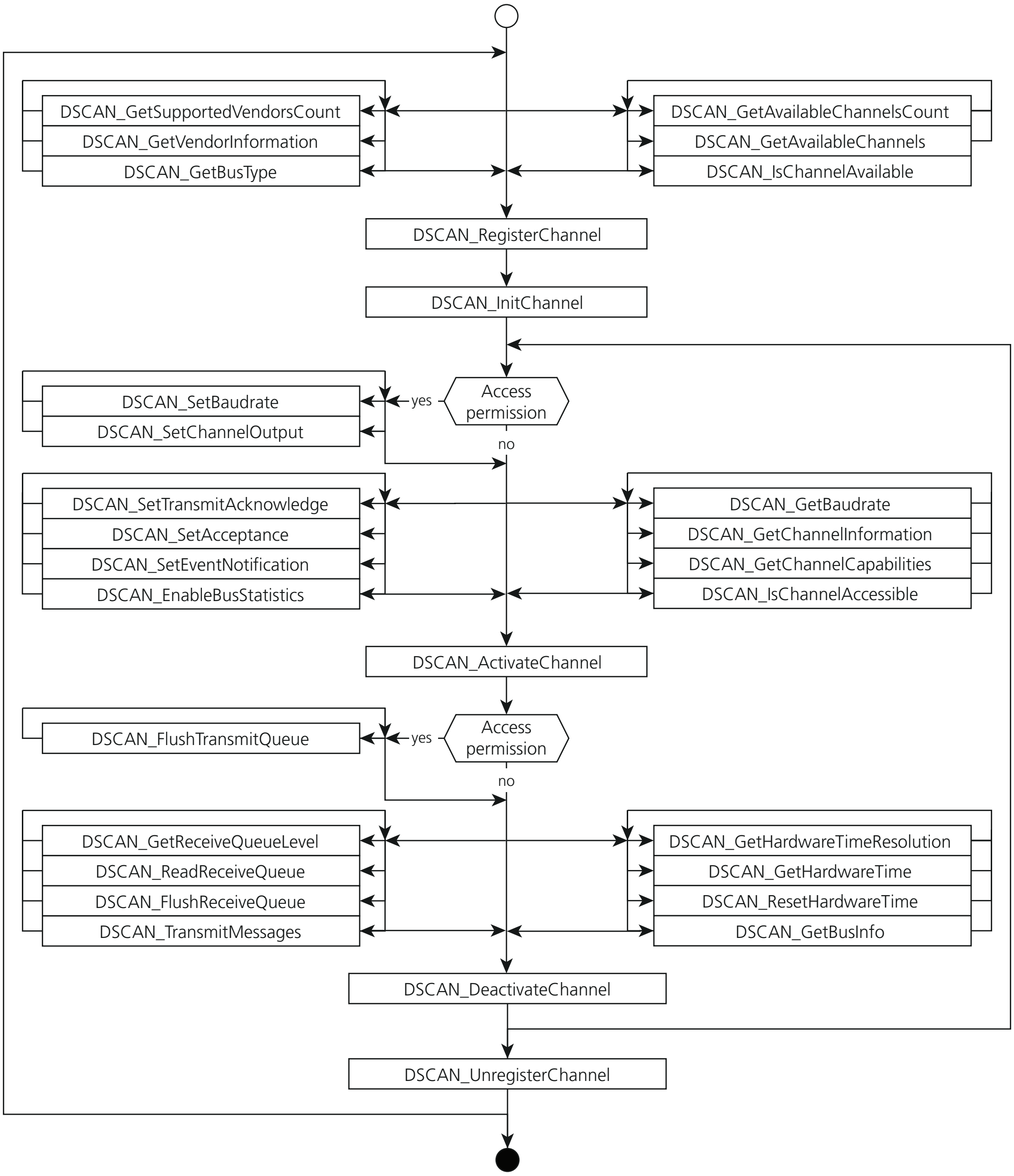
Basics

Demo for dSPACE CAN API 2.0.. 19

Overview of the API Functions and Their Dependencies

Overview

The following illustration shows an overview of the dSPACE CAN API 2.0 functions and the possible function sequences.



Convenience function The following dSPACE CAN API 2.0 function is provided for convenience:

- DSCAN_ReadReceiveQueueAndDeactivateChannel

Error handling functions The following dSPACE CAN API 2.0 functions are provided for error handling:

- DSCAN_GetErrorText
- DSCAN_GetLastVendorSpecificError

Auxiliary functions The dSPACE CAN API 2.0 provides the following auxiliary functions:

- DSCAN_ConvertBaudrateToBitTimingParameters
- DSCAN_ConvertBaudratesToBitTimingParameters
- DSCAN_ConvertBaudrateToBitTimingParametersWithSameSPAndBRP
- DSCAN_ConvertBusTimingRegistersToBitTimingParameters
- DSCAN_ConvertBitTimingParametersToBaudrate
- DSCAN_ConvertBitTimingParametersToBusTimingRegisters
- DSCAN_ConvertByteCountToDlc
- DSCAN_ConvertDlcToByteCount
- DSCAN_CalculateAcceptanceFilter
- DSCAN_MergeAcceptanceFilter
- DSCAN_EncodeBusStatistics
- DSCAN_ConvertApiVersionToString

Related topics

Basics

Basics of the dSPACE CAN API 2.0..9

Files of dSPACE CAN API 2.0

Files and their locations The following table shows the locations of the important files of dSPACE CAN API 2.0 after installation.

File Name	Path	Description
Software Module with Programming Interface		
`DSCanApi20.dll`	`%SystemRoot%\system32`	dSPACE CAN API 2.0 DLL
▪ `DSCanApi20.lib` ▪ `DSCanApi20.h` ▪ `DSCanApi20Itfs.h`	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6 \DSCanApi_4.0.6 \DsCanApi20`	Lib and header for the C++ demo and for your own applications.
`DSCanApi20DotNet.dll`	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6 \DSCanApi_4.0.6 \DsCanApi20`	.NET DLL for the C# demo and for your own applications.

File Name	Path	Description
▪ `dscanapi20lib.pyc` ▪ `dscanapi20itfslib.pyc`	`%ProgramFiles%\Python27\Lib\site-packages\dSPACECommon`	Python wrapper for the Python demo and for your own applications.
C++ Demo Application		
`DSCanApi20_GettingStarted.exe`	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6\DSCanApi_4.0.6\Demos\DsCanApi20\GettingStarted\C++\bin\x64\Release\`	C++ demo application. Refer to Steps Shown in the Demo on page 20.
`DSCanApi20_GettingStarted.sln`	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6\DSCanApi_4.0.6\Demos\DsCanApi20\GettingStarted\C++\src\`	Source code for the C++ demo application.
C# Demo Application		
`DSCanApi20_GettingStarted.exe`	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6\DSCanApi_4.0.6\Demos\DsCanApi20\GettingStarted\C#\bin\x64\Release\`	C# demo application. Refer to Steps Shown in the Demo on page 20.
`DSCanApi20_GettingStarted.sln`	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6\DSCanApi_4.0.6\Demos\DsCanApi20\GettingStarted\C#\src\`	Source code for the C# demo application.
Python Demo Script		
`DsCanApi20_GettingStarted.py`[1]	`%CommonProgramFiles%\dSPACE\DSCanApi_4.0.6\DSCanApi_4.0.6\Demos\DsCanApi20\GettingStarted\Python\`	Demo script for Python. Refer to Steps Shown in the Demo on page 20.

[1] The Python demo imports the `dscanapi20lib` Python module, which implements the dSPACE CAN API 2.0 in the Python programming language.

Related topics

Basics

Basics of the dSPACE CAN API 2.0 9

Software Requirements for Working with dSPACE CAN API 2.0

Software requirements

Operating systems The dSPACE CAN API 2.0 supports the following operating systems:

- Windows 10 (64-bit version)

CAN driver software For information on the CAN driver software required for CAN interfaces from Eberspächer GmbH, Kvaser or Vector Informatik GmbH, refer to Overview of Required Third-Party Software (Installing dSPACE Software).

Related topics

References

Overview of Required Third-Party Software (Installing dSPACE Software 🕮)

Basics on CAN FD

Introduction

Using the CAN FD protocol allows data rates higher than 1 MBit/s and payloads longer than 8 bytes per message.

Basics on CAN FD

CAN FD stands for *CAN with Flexible Data Rate*. The CAN FD protocol is based on the CAN protocol as specified in ISO 11898-1. Compared with the classic CAN protocol, CAN FD comes with an increased bandwidth for the serial communication. The improvement is based on two factors:

- The CAN FD protocol allows you to use CAN messages with longer data fields (up to 64 bytes).
- The CAN FD protocol allows you to use a higher bit rate (typically higher by a factor of 8). It is possible to switch inside the message to the faster bit rate.

Arbitration phase and data phase CAN FD messages consist of two phases:

- Arbitration phase

  CAN FD still uses the *CAN bus arbitration* method. During the arbitration process, the standard data rate is used.
- Data phase

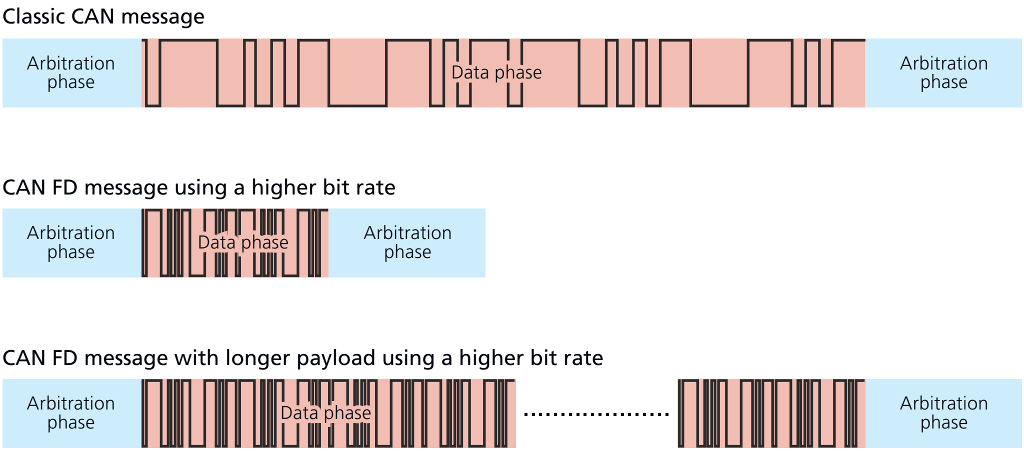
  The data phase spans the phase where the data bits, CRC and length information are transferred. The data phase can be configured to have a higher bit rate than the arbitration phase, so that data bits are transferred with the preconfigured higher bit rate.

  At the end of the data phase, CAN FD returns to the standard data rate.

The following illustration shows:

- A classic CAN message
- A CAN FD message using a higher bit rate during the data phase
- A CAN FD message with longer payload using a higher bit rate

You can see the implications of the CAN FD features: The arbitration phases are identical in all cases, because the standard bit rate is always used. The lengths of the data phases differ depending on the payload length and bit rate used.



Classic CAN message

CAN FD message using a higher bit rate

CAN FD message with longer payload using a higher bit rate

CAN FD protocols

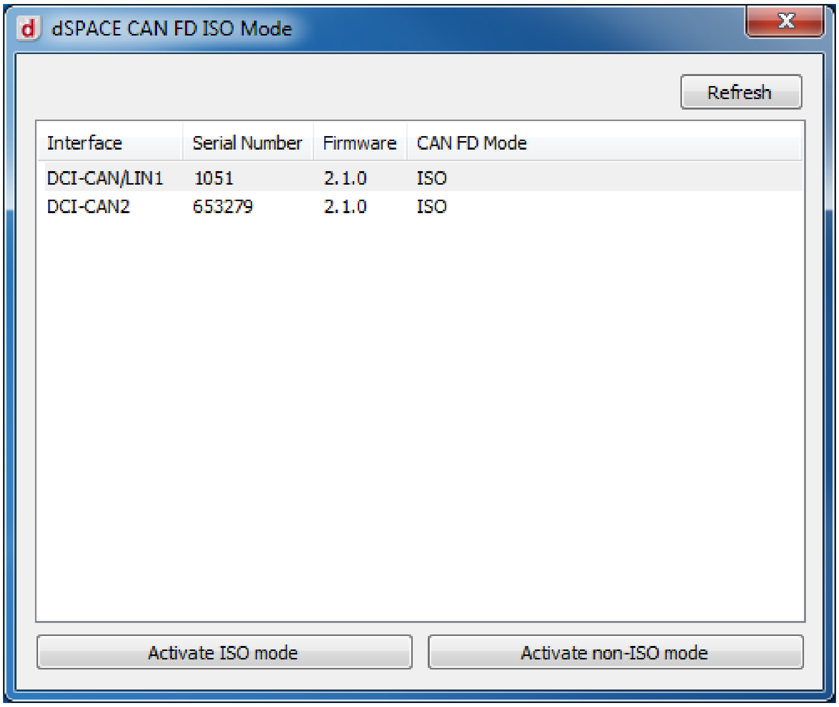
Currently, there are two CAN FD protocols on the market, which are not compatible with each other.

- The *non-ISO CAN FD protocol* represents the original CAN FD protocol from Bosch.
- The *ISO CAN FD protocol* represents the CAN FD protocol according to the ISO 11898-1:2015 standard.

Compared to the non-ISO CAN FD protocol, the ISO CAN FD protocol comes with an improved failure detection capability.

The DCI-CAN2 and the DCI-CAN/LIN1 support both CAN FD protocols.

Switching between ISO CAN FD and non-ISO CAN FD To switch between ISO CAN FD and non-ISO CAN FD, you can use the dSPACE CAN FD ISO Mode (`DsCanFdIsoMode.exe`) tool. It is installed in the `C:\Program Files <(x86)>\Common Files\dSPACE\ DSCanApi_<Version>\` folder.



Interface	Serial Number	Firmware	CAN FD Mode
DCI-CAN/LIN1	1051	2.1.0	ISO
DCI-CAN2	653279	2.1.0	ISO

Demo for dSPACE CAN API 2.0

Where to go from here

Information in this section

Overview of the Demo

Introduction

The dSPACE CAN API installation comprises a demo for getting started with dSPACE CAN API 2.0.

Using physical or virtual CAN interface channels

The demo uses either physical or virtual CAN interface channels:

- Physical CAN interface channels are used if at least two CAN interface channels are connected to the host PC (e.g., one DCI-CAN/LIN1 with two CAN interface channels, or two DCI-CAN2 each of which has one CAN interface channel).
- Virtual CAN interface channels are used if less than two physical CAN interface channels are connected to the host PC.

Different programming languages

The demo is available in the following programming languages:

- C++
- C#
- Python

You can use the demo code (or parts of it) as the starting point for your own applications and scripts.

Location of the demo files

For the location of the demo files, refer to Files of dSPACE CAN API 2.0 on page 14.

Related topics

Basics

Steps Shown in the Demo

Steps

The demo performs the following steps:

1. Getting available interface channels by using the `DSCAN_GetAvailableChannels` function.
2. Selecting two interface channels for demo use, and checking whether the channels support CAN FD by using the `DSCAN_GetChannelCapabilities` function.
3. Registering three channels by using the `DSCAN_RegisterChannel` function.

   The following settings are used:

Channel (Client)	Interface Channel
1	1
2	1[1]
3	2

   [1] Channels 1 and 2 are registered with the data of the same interface channel. This is an example for a multiclient scenario in which several clients (= channels) connect to the same CAN interface channel.

4. Initializing and getting access permission to CAN interface channels by using the `DSCAN_InitChannel` function.

   The following settings are used:

Channel (Client)	CAN Message Identifier Types to Receive	Receive Queue Size	CAN FD Support Required	Access Permission
1	Standard and extended	Maximum buffer	Yes[1]	Yes
2	Standard only	Maximum buffer	Yes[1]	No[2]

Channel (Client)	CAN Message Identifier Types to Receive	Receive Queue Size	CAN FD Support Required	Access Permission
3	Standard and extended	Maximum buffer	Yes[1]	Yes

[1] If supported by the used interface channel

[2] Channel 1 already accesses this interface channel.

5. Setting the baud rate of the channels by using the `DSCAN_SetBaudrate` function.

   The `DSCAN_ConvertBaudrateToBitTimingParameters` function is used to convert the clock frequency and the desired baud rate into bit timing parameters.

   The following settings are used for all the three channels:

	Clock Frequency	Baud Rate
Classic CAN	8 MHz	1 MBit/s
CAN FD	80 MHz	▪ 1 MBit/s (arbitration phase) ▪ 4 MBit/s (data phase)

   For channels that have no access permission, the current baud rate of the interface channel is returned by using the `DSCAN_GetBaudrate` function.

6. Setting acceptance filters for the channels by using the `DSCAN_SetAcceptance` function.

   The `DSCAN_CalculateAcceptanceFilter` function is used to calculate acceptance filters.

   The following settings are used:

Channel (Client)	CAN Message Identifier Types to Receive	Mask Filter
1	Standard and extended[1]	None, i.e., all the incoming messages are accepted.
2	Standard only[1]	Only the incoming messages with the ID 10 are accepted.
3	Standard and extended[1]	Only the incoming messages with the standard ID 20 and the extended ID 10000 are accepted.

[1] As configured by using the `DSCAN_InitChannel` function.

7. Setting transmit acknowledgements for the channels by using the `DSCAN_SetTransmitAcknowledge` function.

   The following settings are used:

Channel (Client)	Transmit Acknowledge
1	Enabled
2	Disabled
3	Disabled

8. Activating the channels by using the `DSCAN_ActivateChannel` function.
9. Getting the hardware time resolution of the channels by using the `DSCAN_GetHardwareTimeResolution` function.
10. Getting the hardware time of the channels by using the `DSCAN_GetHardwareTime` function.
11. Flushing the transmit queue of the channels by using the `DSCAN_FlushTransmitQueue` function.
12. Flushing the receive queue of the channels by using the `DSCAN_FlushReceiveQueue` function.
13. Transmitting CAN messages by each channel by using the `DSCAN_TransmitMessages` function.

    Each channel transmits the following messages:

	Messages to be Transmitted
Classic CAN	▪ 3 messages with standard IDs 10, 20 and 30 ▪ 2 messages with extended IDs 10000 and 10100
CAN FD[1)]	▪ 3 standard CAN messages with standard IDs 10, 20 and 30 ▪ 2 standard CAN messages with extended IDs 10000 and 10100 ▪ 3 CAN FD messages with standard IDs 10, 20 and 30 and with baud rate switch ▪ 2 CAN FD messages with extended IDs 10000 and 10100 and without baud rate switch

[1)] If supported by the used interface channels

The following ways of message transmission are used:

Channel (Client)	Message Transmission
1	One message after the other
2	All messages at once
3	All messages at once

14. Reading the receive queues of the channels by using the `DSCAN_ReadReceiveQueue` function.

    The following settings are used for message reception:

Channel (Client)	Message Reception
1	All the incoming standard and extended messages, and messages transmitted by the channel itself are received.[1)]
2	Only incoming standard messages with the ID 10 are received. Messages transmitted by the channel itself are not received.[1)]
3	Only incoming standard messages with the ID 20 and extended messages with the ID 10000 are received. Messages transmitted by the channel itself are not received. [1)]

[1)] As configured by using the functions `DSCAN_InitChannel`, `DSCAN_SetAcceptance`, and `DSCAN_SetTransmitAcknowledge`.

15. Setting an event notification when channel 3 receives a CAN message by using the `DSCAN_SetEventNotification` function.
16. Getting bus information from the channels by using the `DSCAN_GetBusInfo` function.

17. Deactivating the channels by using the `DSCAN_DeactivateChannel` function.
18. Unregistering the channels by using the `DSCAN_UnregisterChannel` function.

Related topics

Basics

Basics of the dSPACE CAN API 2.0 9
Overview of the Demo 19

General Handling

Where to go from here

Information in this section

Data Types, Constants

Where to go from here

Information in this section

Data Types

Data types

DSTEventHandle

Data Type	Description
typedef HANDLE_PTR DSTEventHandle	Event handle

DSTCanHandle

Data Type	Description
typedef long DSTCanHandle	Channel handle

DSTCanError

Data Type	Description
typedef long DSTCanError	Error codes. For the list of error codes, refer to Error Codes on page 100.

Constants

Constants

DSCAN_MAX_DATA_LENGTH

Value	Description
64	Maximum data length of a CAN message

DSCAN_MAX_NAME_LENGTH

Value	Description
256	Maximum length of any naming strings (e.g. vendor name, interface name, etc.)

DSCAN_INVALID_CAN_HANDLE

Value	Description
-1	Invalid channel handle

DSCAN_INVALID_EVENT_HANDLE

Value	Description
0	Invalid event handle

DSCAN_MAX_TEXT_LENGTH

Value	Description
2048	Maximum length of any text strings (e.g. error text)

DSCAN_MAX_RX_QUEUE_SIZE

Value	Description
32768	Maximum receive queue size

DSCAN_CRYSTAL_FREQUENCY_SJA1000

Value	Description
16000000	16 MHz

DSCAN_CLOCK_FREQUENCY_SJA1000

Value	Description
(DSCAN_CRYSTAL_FREQUENCY_SJA1000 / 2)	8 MHz (16 MHz crystal frequency with the fixed prescaler of 2)

CAN Interface Bus Types

CAN interface bus types The following constants are predefined:

Predefined Constants	Value	Description
DSCAN_INTERFACE_BUS_TYPE_UNKNOWN	Unknown bus	Unknown bus type
DSCAN_INTERFACE_BUS_TYPE_VIRTUAL	Virtual	Virtual bus
DSCAN_INTERFACE_BUS_TYPE_USB	USB	USB
DSCAN_INTERFACE_BUS_TYPE_PCMCIA	PCMCIA	PCMCIA
DSCAN_INTERFACE_BUS_TYPE_PCI_EXPRESS	PCI Express	PCI Express
DSCAN_INTERFACE_BUS_TYPE_ETHERNET	Ethernet	Ethernet

Functions

DSCAN_GetBusType

Purpose To return the bus type of a CAN interface.

Syntax

```
DSTCanError DSCAN_GetBusType(char  szVendorName[DSCAN_MAX_NAME_LENGTH],
                             char  szInterfaceName[DSCAN_MAX_NAME_LENGTH],
                             char* pszBusType);
```

Parameters (In) **szVendorName** Specifies the vendor name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

szInterfaceName Specifies the interface name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_UNKNOWN	Unknown interface	Unknown interface
DSCAN_INTERFACE_NAME_DSPACE_DCI_CAN2	DCI-CAN2	dSPACE DCI-CAN2
DSCAN_INTERFACE_NAME_DSPACE_DCI_CANLIN1	DCI-CAN/LIN1	dSPACE DCI-CAN/LIN1
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II[1)]	FlexCard Cyclone II	Eberspächer FlexCard Cyclone II
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II_SE[1)]	FlexCard Cyclone II SE	Eberspächer FlexCard Cyclone II SE
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_USB[1)]	Eberspaecher FlexCard USB	Eberspächer FlexCard USB
DSCAN_INTERFACE_NAME_KVASER_LAPCAN[1)]	LAPcan	Kvaser LAPcan
DSCAN_INTERFACE_NAME_KVASER_LEAF	Leaf	Kvaser Leaf
DSCAN_INTERFACE_NAME_KVASER_MEMORATOR_PRO	Memorator Professional	Kvaser Memorator Professional
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_II	USBcan II	Kvaser USBcan II

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_PRO	USBcan Professional	Kvaser USBcan Professional
DSCAN_INTERFACE_NAME_PEAK_PCAN_MINI_PCIE_FD	PCAN-miniPCIe FD	PEAK PCAN-miniPCIe FD
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XL[1)]	CANcardXL	Vector CANcardXL
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XLE[1)]	CANcardXLe	Vector CANcardXLe
DSCAN_INTERFACE_NAME_VECTOR_CANCASE_XL	CANcaseXL	Vector CANcaseXL
DSCAN_INTERFACE_NAME_VECTOR_VN1610	VN1610	Vector VN1610
DSCAN_INTERFACE_NAME_VECTOR_VN1611	VN1611	Vector VN1611
DSCAN_INTERFACE_NAME_VECTOR_VN1630	VN1630	Vector VN1630
DSCAN_INTERFACE_NAME_VECTOR_VN1640	VN1640	Vector VN1640
DSCAN_INTERFACE_NAME_VECTOR_VN5610	VN5610	Vector VN5610
DSCAN_INTERFACE_NAME_VECTOR_VN5610A	VN5610A	Vector VN5610A
DSCAN_INTERFACE_NAME_VECTOR_VN7600	VN7600	Vector VN7600
DSCAN_INTERFACE_NAME_VECTOR_VN8900	VN8900	Vector VN8900
DSCAN_INTERFACE_NAME_VIRTUAL	Virtual	Virtual interface

[1)] Deprecated

Parameters (Out)

pszBusType Specifies the bus type of the interface.

The following constants are predefined:

Predefined Constants	Value	Description
DSCAN_INTERFACE_BUS_TYPE_UNKNOWN	Unknown bus	Unknown bus type
DSCAN_INTERFACE_BUS_TYPE_VIRTUAL	Virtual	Virtual bus
DSCAN_INTERFACE_BUS_TYPE_USB	USB	USB
DSCAN_INTERFACE_BUS_TYPE_PCMCIA	PCMCIA	PCMCIA
DSCAN_INTERFACE_BUS_TYPE_PCI_EXPRESS	PCI Express	PCI Express
DSCAN_INTERFACE_BUS_TYPE_ETHERNET	Ethernet	Ethernet

The size of this parameter must be at least `DSCAN_MAX_TEXT_LENGTH` bytes. The memory must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Overview of the API Functions and Their Dependencies.. 13

Vendor Information

Where to go from here

Information in this section

Data Types, Constants

CAN Vendor Names

CAN vendor names

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

Structures

DSSCanVendorInfo

Purpose Structure for vendor CAN API information

Syntax

```
typedef struct DSSCanVendorInfo
{
    char          szVendorName[DSCAN_MAX_NAME_LENGTH];
    char          szVendorApiDllName[DSCAN_MAX_NAME_LENGTH];
    unsigned long ulVendorApiVersion;
    unsigned long ulRequiredVendorApiVersion;
    DSTCanError   tVendorApiState;
} DSSCanVendorInfo;
```

Parameters

szVendorName Specifies the vendor name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

szVendorApiDllName Vendor CAN API DLL name

ulVendorApiVersion Vendor CAN API version. It is provided as unsigned long and has the following format:

Byte 3	Byte2	Byte 1	Byte 0
Major	Minor	Build	

The vendor CAN API version is available only if the vendor CAN API DLL could be loaded successfully.

Use `DSCAN_ConvertApiVersionToString` to convert a CAN API version to string.

ulRequiredVendorApiVersion Required vendor CAN API version. It is provided as unsigned long and has the following format:

Byte 3	Byte2	Byte 1	Byte 0
Major	Minor	Build	

Use `DSCAN_ConvertApiVersionToString` to convert a CAN API version to string.

tVendorApiState Vendor CAN API validation result.

The vendor CAN API validation result indicates whether the vendor CAN API can be used:

- The vendor CAN API validation result is `DSCAN_ERR_NO_ERROR`, i.e., the vendor CAN API can be used if the following conditions are met:
  - The vendor CAN API DLL can be loaded successfully.
  - The vendor CAN API DLL contains all the functions required by the dSPACE CAN API.
  - The actual vendor CAN API version is equal to or greater than the required version.
- Otherwise, the vendor CAN API validation result contains an error code that describes the validation problem.

Functions

Where to go from here

Information in this section

Basics on Vendor Information Functions

Steps to obtain vendor CAN API information

To obtain information about supported vendor CAN APIs, perform the following steps:

1. Call the `DSCAN_GetSupportedVendorsCount` function to get the count of supported vendor CAN APIs.
2. Allocate memory for the vendor CAN API information.
3. Call the `DSCAN_GetVendorInformation` function to get the information on supported vendor CAN APIs.
4. Free the allocated memory after processing the information on supported vendor CAN APIs.

See the following example.

Example

The following example shows how to obtain information about supported vendor CAN APIs.

```
DSTCanError       tErrorCode     = DSCAN_ERR_NO_ERROR;
unsigned long     ulVendorsCount = 0;
DSSCanVendorInfo* ptVendorsArray = NULL;
// get count of supported vendor CAN APIs
tErrorCode = DSCAN_GetSupportedVendorsCount(&ulVendorsCount);
if (DSCAN_ERR_NO_ERROR == tErrorCode)
{
   // allocate memory for supported vendor CAN APIs information
   ptVendorsArray = new DSSCanVendorInfo[ulVendorsCount];
   // get supported vendor CAN APIs information
   tErrorCode = DSCAN_GetVendorInformation(&ulVendorsCount, ptVendorsArray);
   // process supported vendor CAN APIs information
   for (unsigned long i = 0; i < ulVendorsCount; i++)
   {
       // do anything with ptVendorsArray[i]
   }
```

```
    // free memory for supported vendor CAN APIs information
    delete[] ptVendorsArray;
}
```

Vendor CAN API information

Vendor CAN API information is described by the `DSSCanVendorInfo` structure. Refer to DSSCanVendorInfo on page 33.

Related topics

References

DSCAN_GetSupportedVendorsCount.. 36
DSCAN_GetVendorInformation.. 37

DSCAN_GetSupportedVendorsCount

Purpose

To return the count of supported vendor CAN APIs.

Syntax

```
DSTCanError DSCAN_GetSupportedVendorsCount(unsigned long* pulVendorsCount);
```

Parameters (Out)

pulVendorsCount Count of supported vendor CAN APIs.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Vendor Information Functions.. 35
Overview of the API Functions and Their Dependencies... 13

DSCAN_GetVendorInformation

Purpose

To return information about supported vendor CAN APIs.

Syntax

```
DSTCanError DSCAN_GetVendorInformation(unsigned long*         pulVendorsCount,
                                       DSSCanVendorInfo*      ptVendorsArray);
```

Parameters (In, Out)

pulVendorsCount Maximum number of supported vendor CAN APIs.

- If used as In parameter:

  Lets you specify the maximum number of supported vendor CAN APIs to obtain. To obtain the information, call the function with `pulVendorsCount` set to the maximum number of vendor CAN APIs to get. This is usually the size of the `ptVendorsArray` parameter.

- If used as Out parameter:

  Lets you get the number of obtained vendor CAN APIs. The `pulVendorsCount` parameter contains the actual number of supported vendor CAN APIs which have been written to the `ptVendorsArray` parameter (the actual number is always less than or equal to the initial maximum value).

Parameters (Out)

ptVendorsArray Vendor CAN APIs information array.

You can call the function with the NULL pointer instead of `ptVendorsArray`. In this case, only the number of supported vendor CAN APIs is obtained. This function call is equal to a call of the `DSCAN_GetSupportedVendorsCount` function.

The memory for `ptVendorsArray` must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Vendor Information Functions.. 35
Overview of the API Functions and Their Dependencies... 13

Channel Information

Where to go from here

Information in this section

Data Types, Constants

Where to go from here

Information in this section

CAN Interface Names

CAN interface names

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_UNKNOWN	Unknown interface	Unknown interface
DSCAN_INTERFACE_NAME_DSPACE_DCI_CAN2	DCI-CAN2	dSPACE DCI-CAN2
DSCAN_INTERFACE_NAME_DSPACE_DCI_CANLIN1	DCI-CAN/LIN1	dSPACE DCI-CAN/LIN1
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II[1)]	FlexCard Cyclone II	Eberspächer FlexCard Cyclone II
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II_SE[1)]	FlexCard Cyclone II SE	Eberspächer FlexCard Cyclone II SE
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_USB[1)]	Eberspaecher FlexCard USB	Eberspächer FlexCard USB
DSCAN_INTERFACE_NAME_KVASER_LAPCAN[1)]	LAPcan	Kvaser LAPcan
DSCAN_INTERFACE_NAME_KVASER_LEAF	Leaf	Kvaser Leaf
DSCAN_INTERFACE_NAME_KVASER_MEMORATOR_PRO	Memorator Professional	Kvaser Memorator Professional
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_II	USBcan II	Kvaser USBcan II
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_PRO	USBcan Professional	Kvaser USBcan Professional
DSCAN_INTERFACE_NAME_PEAK_PCAN_MINI_PCIE_FD	PCAN-miniPCIe FD	PEAK PCAN-miniPCIe FD
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XL[1)]	CANcardXL	Vector CANcardXL
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XLE[1)]	CANcardXLe	Vector CANcardXLe
DSCAN_INTERFACE_NAME_VECTOR_CANCASE_XL	CANcaseXL	Vector CANcaseXL
DSCAN_INTERFACE_NAME_VECTOR_VN1610	VN1610	Vector VN1610
DSCAN_INTERFACE_NAME_VECTOR_VN1611	VN1611	Vector VN1611
DSCAN_INTERFACE_NAME_VECTOR_VN1630	VN1630	Vector VN1630

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_VECTOR_VN1640	VN1640	Vector VN1640
DSCAN_INTERFACE_NAME_VECTOR_VN5610	VN5610	Vector VN5610
DSCAN_INTERFACE_NAME_VECTOR_VN5610A	VN5610A	Vector VN5610A
DSCAN_INTERFACE_NAME_VECTOR_VN7600	VN7600	Vector VN7600
DSCAN_INTERFACE_NAME_VECTOR_VN8900	VN8900	Vector VN8900
DSCAN_INTERFACE_NAME_VIRTUAL	Virtual	Virtual interface

[1] Deprecated

CAN Channel Capabilities

CAN channel capabilities The following constants are predefined:

Predefined Constant	Description
DSCAN_CHANNEL_CAPABILITY_FD Value: `0x00000001`	The channel supports CAN FD.[1]
DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO Value: `0x00000008`	Relevant only if DSCAN_CHANNEL_CAPABILITY_FD is set. ▪ If DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO is set, the channel uses the non-ISO CAN FD communication[1]. ▪ If DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO is not set, the channel uses the ISO CAN FD communication.
DSCAN_CHANNEL_CAPABILITY_FIXED_CONTROLLER_CONFIGURATION Value: `0x00000002`	The channel supports only fixed CAN controller configuration. Baud rate settings cannot be modified.
DSCAN_CHANNEL_CAPABILITY_BUS_LOAD_INFO Value: `0x00000004`	The channel supports bus load information.
DSCAN_CHANNEL_CAPABILITY_BUS_STATISTICS Value: `0x00000010`	The channel supports bus statistics.

[1] For more information on CAN FD, refer to Basics on CAN FD on page 16.

Enumerations

Enumerations **DSECanChannelsSearchAttributeType** CAN channels search attribute types.

Enumerator	Value	Description
DSCAN_SEARCH_ARRTIBUTE_TYPE_IP_V4_ADDRESS	1	IPv4 address

Structures

Where to go from here

Information in this section

DSSCanChannelInfo

Purpose

Structure for CAN channel information

Syntax

```
typedef struct DSSCanChannelInfo
{
    char          szVendorName[DSCAN_MAX_NAME_LENGTH];
    char          szInterfaceName[DSCAN_MAX_NAME_LENGTH];
    char          szInterfaceSerialNumber[DSCAN_MAX_NAME_LENGTH];
    char          szChannelIdentifier[DSCAN_MAX_NAME_LENGTH];
    unsigned long ulChannelCapabilities;
} DSSCanChannelInfo;
```

Parameters

szVendorName Specifies the vendor name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

szInterfaceName Specifies the interface name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_UNKNOWN	Unknown interface	Unknown interface
DSCAN_INTERFACE_NAME_DSPACE_DCI_CAN2	DCI-CAN2	dSPACE DCI-CAN2

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_DSPACE_DCI_CANLIN1	DCI-CAN/LIN1	dSPACE DCI-CAN/LIN1
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II[1)]	FlexCard Cyclone II	Eberspächer FlexCard Cyclone II
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II_SE[1)]	FlexCard Cyclone II SE	Eberspächer FlexCard Cyclone II SE
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_USB[1)]	Eberspaecher FlexCard USB	Eberspächer FlexCard USB
DSCAN_INTERFACE_NAME_KVASER_LAPCAN[1)]	LAPcan	Kvaser LAPcan
DSCAN_INTERFACE_NAME_KVASER_LEAF	Leaf	Kvaser Leaf
DSCAN_INTERFACE_NAME_KVASER_MEMORATOR_PRO	Memorator Professional	Kvaser Memorator Professional
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_II	USBcan II	Kvaser USBcan II
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_PRO	USBcan Professional	Kvaser USBcan Professional
DSCAN_INTERFACE_NAME_PEAK_PCAN_MINI_PCIE_FD	PCAN-miniPCIe FD	PEAK PCAN-miniPCIe FD
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XL[1)]	CANcardXL	Vector CANcardXL
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XLE[1)]	CANcardXLe	Vector CANcardXLe
DSCAN_INTERFACE_NAME_VECTOR_CANCASE_XL	CANcaseXL	Vector CANcaseXL
DSCAN_INTERFACE_NAME_VECTOR_VN1610	VN1610	Vector VN1610
DSCAN_INTERFACE_NAME_VECTOR_VN1611	VN1611	Vector VN1611
DSCAN_INTERFACE_NAME_VECTOR_VN1630	VN1630	Vector VN1630
DSCAN_INTERFACE_NAME_VECTOR_VN1640	VN1640	Vector VN1640
DSCAN_INTERFACE_NAME_VECTOR_VN5610	VN5610	Vector VN5610
DSCAN_INTERFACE_NAME_VECTOR_VN5610A	VN5610A	Vector VN5610A
DSCAN_INTERFACE_NAME_VECTOR_VN7600	VN7600	Vector VN7600
DSCAN_INTERFACE_NAME_VECTOR_VN8900	VN8900	Vector VN8900
DSCAN_INTERFACE_NAME_VIRTUAL	Virtual	Virtual interface

[1)] Deprecated

szInterfaceSerialNumber Serial number of the interface.

szChannelIdentifier Identifier of the channel.

ulChannelCapabilities Specifies the channel capabilities as a combination of one or more of the following parameter values.

The following constants are predefined:

Predefined Constant	Description
DSCAN_CHANNEL_CAPABILITY_FD Value: `0x00000001`	The channel supports CAN FD.[1)]
DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO Value: `0x00000008`	Relevant only if DSCAN_CHANNEL_CAPABILITY_FD is set. ▪ If DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO is set, the channel uses the non-ISO CAN FD communication[1)]. ▪ If DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO is not set, the channel uses the ISO CAN FD communication.

Predefined Constant	**Description**
DSCAN_CHANNEL_CAPABILITY_FIXED_CONTROLLER_CONFIGURATION Value: `0x00000002`	The channel supports only fixed CAN controller configuration. Baud rate settings cannot be modified.
DSCAN_CHANNEL_CAPABILITY_BUS_LOAD_INFO Value: `0x00000004`	The channel supports bus load information.
DSCAN_CHANNEL_CAPABILITY_BUS_STATISTICS Value: `0x00000010`	The channel supports bus statistics.

[1] For more information on CAN FD, refer to Basics on CAN FD on page 16.

DSSCanChannelsSearchAttribute

Purpose Structure for CAN channels search attribute

Syntax

```
typedef struct DSSCanChannelsSearchAttribute
{
    DSECanChannelsSearchAttributeType tSearchAttributeType;
    char                              szSearchAttribute[DSCAN_MAX_NAME_LENGTH];
} DSSCanChannelsSearchAttribute;
```

Parameters

tSearchAttributeType Search attribute type.

szSearchAttribute Search attribute.

Functions

Where to go from here

Information in this section

Basics on Channel Information Functions

Steps to obtain information about available CAN channels

To obtain information about available CAN channels, perform the following steps:

1. Call the `DSCAN_GetAvailableChannelsCount` function to get the count of available CAN channels.
2. Allocate memory for the available CAN channels.
3. Call the `DSCAN_GetAvailableChannels` function to get the available CAN channels.
4. Free the allocated memory after processing the available CAN channels.

See the following example.

Example

The following example shows how to obtain information about available CAN channels.

```
DSTCanError        tErrorCode      = DSCAN_ERR_NO_ERROR;
unsigned long      ulChannelsCount = 0;
DSSCanChannelInfo* ptChannelsArray = NULL;
// get count of available CAN channelst
ErrorCode = DSCAN_GetAvailableChannelsCount(&ulChannelsCount, 0, NULL);
if (DSCAN_ERR_NO_ERROR == tErrorCode)
{
   // allocate memory for available CAN channels
   ptChannelsArray = new DSSCanChannelInfo[ulChannelsCount];
   // get available CAN channels
   tErrorCode = DSCAN_GetAvailableChannels(&ulChannelsCount, ptChannelsArray, 0, NULL);
   // process available CAN channels
   for (unsigned long i = 0; i < ulChannelsCount; i++)
   {
       // do anything with ptChannelsArray[i]
   }
   // free memory for available CAN channels
   delete[] ptChannelsArray;
}
```

CAN channel information

CAN channel information is described by the `DSSCanChannelInfo` structure. For details, refer to DSSCanChannelInfo on page 42.

Related topics

References

DSCAN_GetAvailableChannels.. 47
DSCAN_GetAvailableChannelsCount... 46

DSCAN_GetAvailableChannelsCount

Purpose

To return the count of available CAN channels.

Syntax

```
DSTCanError DSCAN_GetAvailableChannelsCount(unsigned long*                 pulChannelsCount,
                                            unsigned long                  ulAdditionalSearchAttributesCount,
                                            DSSCanChannelsSearchAttribute* ptAdditionalSearchAttributesArray);
```

Parameters (In)

ulAdditionalSearchAttributesCount Reserved for future use. Use `0` instead.

ptAdditionalSearchAttributesArray Reserved for future use. Use the NULL pointer instead.

Parameters (Out)

pulChannelsCount Count of available CAN channels.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

DSCAN_GetAvailableChannels

Purpose

To return information about available CAN channels.

Syntax

```
DSTCanError DSCAN_GetAvailableChannels(unsigned long*                 pulChannelsCount,
                                        DSSCanChannelInfo*             ptChannelsArray,
                                        unsigned long                  ulAdditionalSearchAttributesCount,
                                        DSSCanChannelsSearchAttribute* ptAdditionalSearchAttributesArray)
```

Parameters (In)

ulAdditionalSearchAttributesCount Reserved for future use. Use `0` instead.

ptAdditionalSearchAttributesArray Reserved for future use. Use the NULL pointer instead.

Parameters (In, Out)

pulChannelsCount Count of available CAN channels.

- If used as In parameter:

  Lets you specify the maximum number of available channels to obtain. To obtain the information, call the function with `pulChannelsCount` set to the maximum number of CAN channels to get. This is usually the size of the `ptChannelsArray` parameter.
- If used as Out parameter:

  Lets you get the number of available CAN channels which have been written to the `ptChannelsArray` parameter (the actual count is always less then or equal to the initial maximum value).

Parameters (Out)

ptChannelsArray Available CAN channels array.

You can call the function with the NULL pointer instead of `ptChannelsArray`. In this case, only the number of available CAN channels is obtained. This function call is equal to a call of the `DSCAN_GetAvailableChannelsCount` function.

The memory for `ptChannelsArray` must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

DSCAN_IsChannelAvailable

Purpose

To check whether the specified CAN channel is available.

Syntax

```
DSTCanError DSCAN_IsChannelAvailable(char                           szVendorName[DSCAN_MAX_NAME_LENGTH],
                                     char                           szInterfaceName[DSCAN_MAX_NAME_LENGTH],
                                     char                           szInterfaceSerialNumber[DSCAN_MAX_NAME_LENGTH],
                                     char                           szChannelIdentifier[DSCAN_MAX_NAME_LENGTH],
                                     bool*                          pbChannelIsAvailable,
                                     unsigned long                  ulAdditionalSearchAttributesCount,
                                     DSSCanChannelsSearchAttribute* ptAdditionalSearchAttributesArray);
```

Parameters (In)

szVendorName Specifies the vendor name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

szInterfaceName Specifies the interface name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_UNKNOWN	Unknown interface	Unknown interface
DSCAN_INTERFACE_NAME_DSPACE_DCI_CAN2	DCI-CAN2	dSPACE DCI-CAN2
DSCAN_INTERFACE_NAME_DSPACE_DCI_CANLIN1	DCI-CAN/LIN1	dSPACE DCI-CAN/LIN1
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II[1)]	FlexCard Cyclone II	Eberspächer FlexCard Cyclone II
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II_SE[1)]	FlexCard Cyclone II SE	Eberspächer FlexCard Cyclone II SE

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_USB[1]	Eberspaecher FlexCard USB	Eberspächer FlexCard USB
DSCAN_INTERFACE_NAME_KVASER_LAPCAN[1]	LAPcan	Kvaser LAPcan
DSCAN_INTERFACE_NAME_KVASER_LEAF	Leaf	Kvaser Leaf
DSCAN_INTERFACE_NAME_KVASER_MEMORATOR_PRO	Memorator Professional	Kvaser Memorator Professional
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_II	USBcan II	Kvaser USBcan II
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_PRO	USBcan Professional	Kvaser USBcan Professional
DSCAN_INTERFACE_NAME_PEAK_PCAN_MINI_PCIE_FD	PCAN-miniPCIe FD	PEAK PCAN-miniPCIe FD
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XL[1]	CANcardXL	Vector CANcardXL
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XLE[1]	CANcardXLe	Vector CANcardXLe
DSCAN_INTERFACE_NAME_VECTOR_CANCASE_XL	CANcaseXL	Vector CANcaseXL
DSCAN_INTERFACE_NAME_VECTOR_VN1610	VN1610	Vector VN1610
DSCAN_INTERFACE_NAME_VECTOR_VN1611	VN1611	Vector VN1611
DSCAN_INTERFACE_NAME_VECTOR_VN1630	VN1630	Vector VN1630
DSCAN_INTERFACE_NAME_VECTOR_VN1640	VN1640	Vector VN1640
DSCAN_INTERFACE_NAME_VECTOR_VN5610	VN5610	Vector VN5610
DSCAN_INTERFACE_NAME_VECTOR_VN5610A	VN5610A	Vector VN5610A
DSCAN_INTERFACE_NAME_VECTOR_VN7600	VN7600	Vector VN7600
DSCAN_INTERFACE_NAME_VECTOR_VN8900	VN8900	Vector VN8900
DSCAN_INTERFACE_NAME_VIRTUAL	Virtual	Virtual interface

[1] Deprecated

szInterfaceSerialNumber Serial number of the interface.

szChannelIdentifier Identifier of the channel.

ulAdditionalSearchAttributesCount Reserved for future use. Use `0` instead.

ptAdditionalSearchAttributesArray Reserved for future use. Use the NULL pointer instead.

Parameters (Out)

pbChannelIsAvailable Flag indicating whether the channel is available.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Configuration

Where to go from here

Information in this section

Data Types, Constants

Where to go from here

Information in this section

CAN Baud Rates

CAN baud rates

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_BAUDRATE_1000_KBAUD	1000000	1000 kBit/s
DSCAN_BAUDRATE_500_KBAUD	500000	500 kBit/s
DSCAN_BAUDRATE_250_KBAUD	250000	250 kBit/s
DSCAN_BAUDRATE_125_KBAUD	125000	125 kBit/s
DSCAN_BAUDRATE_100_KBAUD	100000	100 kBit/s
DSCAN_BAUDRATE_50_KBAUD	50000	50 kBit/s
DSCAN_BAUDRATE_20_KBAUD	20000	20 kBit/s
DSCAN_BAUDRATE_10_KBAUD	10000	10 kBit/s

CAN Acceptance

CAN acceptance

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_ACCEPTANCE_CODE_BLOCK_NONE	0x00000000	Code to block no CAN messages
DSCAN_ACCEPTANCE_MASK_BLOCK_NONE	0x00000000	Mask to block no CAN messages
DSCAN_ACCEPTANCE_CODE_BLOCK_ALL_STD	0x00000FFF	Code to block all standard CAN messages
DSCAN_ACCEPTANCE_MASK_BLOCK_ALL_STD	0x00000FFF	Mask to block all standard CAN messages

Predefined Constant	Value	Description
DSCAN_ACCEPTANCE_CODE_BLOCK_ALL_XTD	0xFFFFFFFF	Code to block all extended CAN messages
DSCAN_ACCEPTANCE_MASK_BLOCK_ALL_XTD	0xFFFFFFFF	Mask to block all extended CAN messages

Enumerations

Enumerations

DSECanIdentifierType CAN identifier types

The following CAN identifier types are predefined:

Enumerator	Value	Description
DSCAN_IDENTIFIER_TYPE_STD	0x01	Standard CAN identifier (11 bits)
DSCAN_IDENTIFIER_TYPE_XTD	0x02	Extended CAN identifier (29 bits)
DSCAN_IDENTIFIER_TYPE_STD_XTD	DSCAN_IDENTIFIER_TYPE_STD | DSCAN_IDENTIFIER_TYPE_XTD	Both standard and extended CAN identifier

Structures

DSSCanBitTimingParameters

Purpose

Structure for CAN bit timing parameters

Syntax

```
typedef struct DSSCanBitTimingParameters
{
    unsigned long ulSJW;
    unsigned long ulBRP;
    unsigned long ulSAM;
    unsigned long ulTSEG1;
    unsigned long ulTSEG2;
} DSSCanBitTimingParameters;
```

Parameters

ulSJW Synchronization jump width

ulBRP Baud rate prescaler

ulSAM Sample mode

ulTSEG1 Bit time segment 1

ulTSEG2 Bit time segment 2

Functions

Where to go from here

Information in this section

Basics on Configuration Functions

Steps to configure a CAN channel

To configure a CAN channel, perform the following steps:

1. Call the `DSCAN_RegisterChannel` function to register a CAN channel.

   Note

   You must register a CAN channel before you can use it.

   Tip

   To register a CAN channel, the related CAN interface hardware does not have to be connected to the host PC.

2. Call the `DSCAN_InitChannel` function to initialize the registered CAN channel and get access permission to it. To initialize a CAN channel, the related CAN interface hardware must be connected to the host PC.

   You can use the same CAN channel hardware multiple times with the same application or even for different applications simultaneously.

   Note

   However, if you initialize an already initialized CAN channel, you get no access permission for that channel. As a consequence, you will not be able to modify the communication configuration of the CAN channel hardware.

3. If you have access permission to the CAN channel, call the `DSCAN_SetBaudrate` function to specify the desired baud rate.

   For details, refer to Basics on Bit Timing Parameters and Baud Rates on page 57.

4. After you have worked with the CAN channel, call the `DSCAN_UnregisterChannel` function to unregister the CAN channel.

   Note

   If a CAN channel remains registered, the dependencies in the driver cannot be cleared so that subsequent calls may fail, or you may not get access permission.

CAN bit timing parameters

CAN bit timing parameters are described by the `DSSCanBitTimingParameters` structure. For details, refer to DSSCanBitTimingParameters on page 54.

Related topics

References

DSSCanBitTimingParameters.. 54

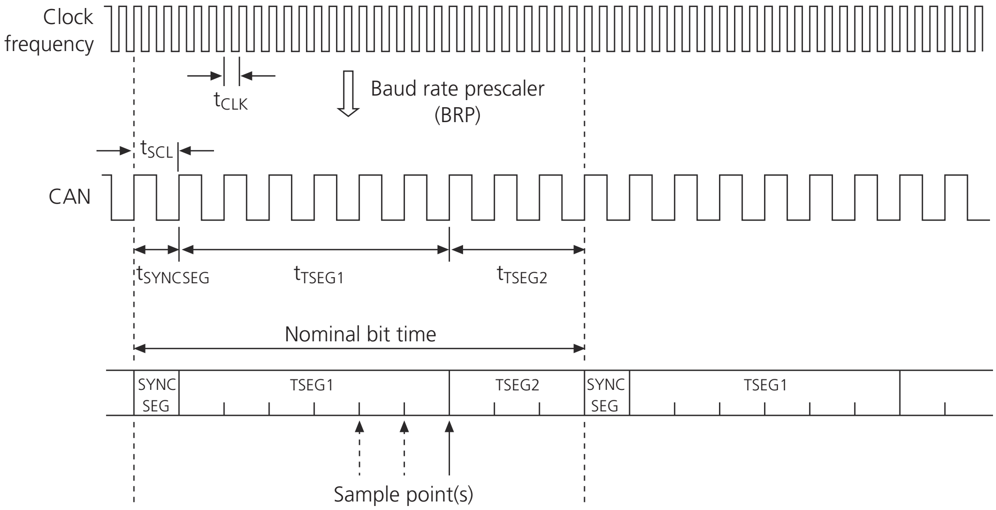
Basics on Bit Timing Parameters and Baud Rates

CAN bit timing parameters

CAN channels can be configured via the following bit timing parameters:

- Synchronization jump width
- Baud rate prescaler
- Sample mode
- Bit time segment 1
- Bit time segment 2

The illustration below shows the meaning and the interdependencies of the parameters:



t_{CLK} = Time period of the quartz frequency (1/quartz frequency)

t_{SCL} = Time period of the CAN system clock

$t_{SYNCSEG}$ = Time period of the synchronization segment

t_{TSEG} = Time period of the time segment

Using dSPACE CAN API 2.0, bit timing parameters are described by the `DSSCanBitTimingParameters` structure.

Baud rate

The baud rate is determined by the CAN interface clock frequency and the bit timing parameters.

The baud rate value is calculated via the following formula:

$$\text{Baud rate} = \frac{\mathit{Clock\ frequency}}{\mathit{Baud\ rate\ prescaler}\ \cdot (1 + \mathit{Bit\ time\ segment}\ 1 + \mathit{Bit\ time\ segment}\ 2)}$$

Setting the baud rate With dSPACE CAN API 2.0, you can set the baud rate of a CAN channel by using the `DSCAN_SetBaudrate` function.

Getting the baud rate With dSPACE CAN API 2.0, you can get the baud rate of a CAN channel by using the `DSCAN_GetBaudrate` function.

Sample mode

The sample mode specifies the number of sample points per bit that is used to evaluate whether the bit is recessive or dominant,

Sample Mode	Number of Sample Points per Bit	Description
0	1	Each bit is sampled only once.
1	3	Each bit is sampled three times. The majority of samples is used to decide whether the bit is recessive or dominant.

Converting baud rates into bit timing parameters and vice versa

To convert a baud rate value to bit timing parameters and vice versa, dSPACE CAN API 2.0 provides the following functions:

- `DSCAN_ConvertBaudrateToBitTimingParameters`
- `DSCAN_ConvertBaudratesToBitTimingParameters`
- `DSCAN_ConvertBaudrateToBitTimingParametersWithSameSPAndBRP`
- `DSCAN_ConvertBitTimingParametersToBaudrate`

- For the most commonly used classic CAN baud rate values (`DSCAN_BAUDRATE_xxx`) and the clock frequency of 8 MHz (`DSCAN_CLOCK_FREQUENCY_SJA1000`) used for classic CAN baud rates and SJA1000-compatible CAN controllers, the `DSCAN_ConvertBaudrateToBitTimingParameters` function returns the following predefined bit timing parameters:

Baud Rate	Synch. Jump Width	Baud Rate Prescaler	Sample Mode	Bit Time Segment 1	Bit Time Segment 2	Sample Point
1000 kBit/s	2	1	0	5	2	75%
500 kBit/s	2	2	0	5	2	75%
250 kBit/s	2	4	0	5	2	75%
125 kBit/s	2	8	0	5	2	75%
100 kBit/s	2	10	0	5	2	75%
50 kBit/s	2	20	0	5	2	75%

Baud Rate	Synch. Jump Width	Baud Rate Prescaler	Sample Mode	Bit Time Segment 1	Bit Time Segment 2	Sample Point
20 kBit/s	2	50	0	5	2	75%
10 kBit/s	2	50	0	13	2	87%

- For all other combinations of baud rate and clock frequency values, the bit timing parameters are determined according to the following conditions:
  - Sample point is equal or close to:
    - 75 % for classic CAN baud rates (≤ 1 Mbit/s)
    - 80 % for CAN FD baud rates (> 1 Mbit/s)
  - Lowest nominal bit time for the desired sample point
  - Sample mode is 0

Converting bit timing parameters into bus timing registers and vice versa

For classic CAN baud rates of up to 1 Mbit/s), bit timing parameters can be stored in two bus timing registers.

To convert bit timing parameters to bus timing registers and vice versa, dSPACE CAN API 2.0 provides the following functions:

- `DSCAN_ConvertBitTimingParametersToBusTimingRegisters`
- `DSCAN_ConvertBusTimingRegistersToBitTimingParameters`

DSCAN_RegisterChannel

Purpose

To register a channel.

Syntax

```
DSTCanError DSCAN_RegisterChannel(char          szVendorName[DSCAN_MAX_NAME_LENGTH],
                                  char          szInterfaceName[DSCAN_MAX_NAME_LENGTH],
                                  char          szInterfaceSerialNumber [DSCAN_MAX_NAME_LENGTH],
                                  char          szChannelIdentifier[DSCAN_MAX_NAME_LENGTH],
                                  DSTCanHandle* ptChannelHandle);
```

Description

You must register a CAN channel before you can use it.

> **Tip**
>
> To register a CAN channel, the related CAN interface hardware does not have to be connected to the host PC.

Parameters (In)

szVendorName Specifies the vendor name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

szInterfaceName Specifies the interface name.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_UNKNOWN	Unknown interface	Unknown interface
DSCAN_INTERFACE_NAME_DSPACE_DCI_CAN2	DCI-CAN2	dSPACE DCI-CAN2
DSCAN_INTERFACE_NAME_DSPACE_DCI_CANLIN1	DCI-CAN/LIN1	dSPACE DCI-CAN/LIN1
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II[1)]	FlexCard Cyclone II	Eberspächer FlexCard Cyclone II
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II_SE[1)]	FlexCard Cyclone II SE	Eberspächer FlexCard Cyclone II SE
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_USB[1)]	Eberspaecher FlexCard USB	Eberspächer FlexCard USB
DSCAN_INTERFACE_NAME_KVASER_LAPCAN[1)]	LAPcan	Kvaser LAPcan
DSCAN_INTERFACE_NAME_KVASER_LEAF	Leaf	Kvaser Leaf
DSCAN_INTERFACE_NAME_KVASER_MEMORATOR_PRO	Memorator Professional	Kvaser Memorator Professional
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_II	USBcan II	Kvaser USBcan II
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_PRO	USBcan Professional	Kvaser USBcan Professional
DSCAN_INTERFACE_NAME_PEAK_PCAN_MINI_PCIE_FD	PCAN-miniPCIe FD	PEAK PCAN-miniPCIe FD
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XL[1)]	CANcardXL	Vector CANcardXL
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XLE[1)]	CANcardXLe	Vector CANcardXLe
DSCAN_INTERFACE_NAME_VECTOR_CANCASE_XL	CANcaseXL	Vector CANcaseXL
DSCAN_INTERFACE_NAME_VECTOR_VN1610	VN1610	Vector VN1610
DSCAN_INTERFACE_NAME_VECTOR_VN1611	VN1611	Vector VN1611
DSCAN_INTERFACE_NAME_VECTOR_VN1630	VN1630	Vector VN1630
DSCAN_INTERFACE_NAME_VECTOR_VN1640	VN1640	Vector VN1640
DSCAN_INTERFACE_NAME_VECTOR_VN5610	VN5610	Vector VN5610
DSCAN_INTERFACE_NAME_VECTOR_VN5610A	VN5610A	Vector VN5610A
DSCAN_INTERFACE_NAME_VECTOR_VN7600	VN7600	Vector VN7600

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_VECTOR_VN8900	VN8900	Vector VN8900
DSCAN_INTERFACE_NAME_VIRTUAL	Virtual	Virtual interface

[1] Deprecated

szInterfaceSerialNumber Serial number of the interface.

szChannelIdentifier Identifier of the channel.

Parameters (Out)

ptChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions.. 56
Overview of the API Functions and Their Dependencies.. 13

DSCAN_InitChannel

Purpose

To initialize a CAN channel and get access permission to it.

Syntax

```
DSTCanError DSCAN_InitChannel(DSTCanHandle         tChannelHandle,
                              DSECanIdentifierType tIdentifierType,
                              unsigned long        ulRxQueueSize,
                              bool                 bFD,
                              bool*                pbAccessPermission);
```

Description

You must initialize a CAN channel before you can use it. To initialize a CAN channel, the related CAN interface hardware must be connected to the host PC.

You can use the same CAN channel hardware multiple times with the same application or even for different applications simultaneously.

Note

However, if you initialize an already initialized CAN channel, you get no access permission for that channel. As a consequence, you will not be able to modify the communication configuration of the CAN channel hardware.

Access permission for a CAN interface channel If the function returns no access permission for the channel, you are not allowed to modify the hardware CAN channel communication configuration since the hardware CAN channel is already used by another client.

In this case, you cannot use the following functions:

- `DSCAN_SetBaudrate`: to set the baud rate of the hardware CAN channel
- `DSCAN_SetChannelOutput`: to enable/disable the silent mode on the hardware CAN channel
- `DSCAN_FlushTransmitQueue`: to clear the transmit queue of the hardware CAN channel

To obtain the current baud rate and CAN FD settings of the channel, you can use the `DSCAN_GetBaudrate` function.

Parameters (In)

tChannelHandle Channel handle.

tIdentifierType Type of CAN message identifiers to receive.

The following CAN identifier types are predefined:

Enumerator	Value	Description
DSCAN_IDENTIFIER_TYPE_STD	0x01	Standard CAN identifier (11 bits)
DSCAN_IDENTIFIER_TYPE_XTD	0x02	Extended CAN identifier (29 bits)
DSCAN_IDENTIFIER_TYPE_STD_XTD	DSCAN_IDENTIFIER_TYPE_STD | DSCAN_IDENTIFIER_TYPE_XTD	Both standard and extended CAN identifier

ulRxQueueSize Receive queue size.

bFD Flag indicating if CAN FD support whether required.

Parameters (Out)

pbAccessPermission Access permission for the channel.

See Description on page 61 for details.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions.. 56
Overview of the API Functions and Their Dependencies.. 13

DSCAN_UnregisterChannel

Purpose

To unregister a channel.

Syntax

```
DSTCanError DSCAN_UnregisterChannel(DSTCanHandle tChannelHandle);
```

Description

After you have used a channel, you must unregister it.

> **Note**
>
> If a CAN channel remains registered, the dependencies in the driver cannot be cleared so that subsequent calls may fail, or you may not get access permission.

Parameters (In)

tChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions.. 56
Overview of the API Functions and Their Dependencies.. 13

DSCAN_GetChannelInformation

Purpose

To return information of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetChannelInformation(DSTCanHandle tChannelHandle,
                                        char*        pszVendorName,
                                        char*        pszInterfaceName,
                                        char*        pszInterfaceSerialNumber,
                                        char*        pszChannelIdentifier);
```

Parameters (In) **tChannelHandle** Channel handle.

Parameters (Out) **pszVendorName** Vendor name.

The size of this parameter must be at least 256 bytes (`DSCAN_MAX_NAME_LENGTH`). The memory must be allocated and freed by the caller.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_VENDOR_NAME_UNKNOWN	Unknown vendor	Unknown vendor
DSCAN_VENDOR_NAME_DSPACE	dSPACE	dSPACE GmbH
DSCAN_VENDOR_NAME_EBERSPAECHER	Eberspaecher	Eberspächer GmbH
DSCAN_VENDOR_NAME_KVASER	Kvaser	Kvaser
DSCAN_VENDOR_NAME_PEAK	PEAK-System	PEAK-System Technik GmbH
DSCAN_VENDOR_NAME_VECTOR	Vector Informatik	Vector Informatik GmbH

pszInterfaceName Interface name.

The size of this parameter must be at least 256 bytes (`DSCAN_MAX_NAME_LENGTH`). The memory must be allocated and freed by the caller.

The following constants are predefined:

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_UNKNOWN	Unknown interface	Unknown interface
DSCAN_INTERFACE_NAME_DSPACE_DCI_CAN2	DCI-CAN2	dSPACE DCI-CAN2
DSCAN_INTERFACE_NAME_DSPACE_DCI_CANLIN1	DCI-CAN/LIN1	dSPACE DCI-CAN/LIN1
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II[1)]	FlexCard Cyclone II	Eberspächer FlexCard Cyclone II
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_CYCLONE_II_SE[1)]	FlexCard Cyclone II SE	Eberspächer FlexCard Cyclone II SE
DSCAN_INTERFACE_NAME_EBERSPAECHER_FLEX_CARD_USB[1)]	Eberspaecher FlexCard USB	Eberspächer FlexCard USB
DSCAN_INTERFACE_NAME_KVASER_LAPCAN[1)]	LAPcan	Kvaser LAPcan
DSCAN_INTERFACE_NAME_KVASER_LEAF	Leaf	Kvaser Leaf
DSCAN_INTERFACE_NAME_KVASER_MEMORATOR_PRO	Memorator Professional	Kvaser Memorator Professional
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_II	USBcan II	Kvaser USBcan II
DSCAN_INTERFACE_NAME_KVASER_USB_CAN_PRO	USBcan Professional	Kvaser USBcan Professional
DSCAN_INTERFACE_NAME_PEAK_PCAN_MINI_PCIE_FD	PCAN-miniPCIe FD	PEAK PCAN-miniPCIe FD
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XL[1)]	CANcardXL	Vector CANcardXL
DSCAN_INTERFACE_NAME_VECTOR_CANCARD_XLE[1)]	CANcardXLe	Vector CANcardXLe
DSCAN_INTERFACE_NAME_VECTOR_CANCASE_XL	CANcaseXL	Vector CANcaseXL
DSCAN_INTERFACE_NAME_VECTOR_VN1610	VN1610	Vector VN1610
DSCAN_INTERFACE_NAME_VECTOR_VN1611	VN1611	Vector VN1611

Predefined Constant	Value	Description
DSCAN_INTERFACE_NAME_VECTOR_VN1630	VN1630	Vector VN1630
DSCAN_INTERFACE_NAME_VECTOR_VN1640	VN1640	Vector VN1640
DSCAN_INTERFACE_NAME_VECTOR_VN5610	VN5610	Vector VN5610
DSCAN_INTERFACE_NAME_VECTOR_VN5610A	VN5610A	Vector VN5610A
DSCAN_INTERFACE_NAME_VECTOR_VN7600	VN7600	Vector VN7600
DSCAN_INTERFACE_NAME_VECTOR_VN8900	VN8900	Vector VN8900
DSCAN_INTERFACE_NAME_VIRTUAL	Virtual	Virtual interface

[1] Deprecated

pszInterfaceSerialNumber Interface serial number.

The size of this parameter must be at least 256 bytes (`DSCAN_MAX_NAME_LENGTH`). The memory must be allocated and freed by the caller.

pszChannelIdentifier Channel identifier.

The size of this parameter must be at least 256 bytes (`DSCAN_MAX_NAME_LENGTH`). The memory must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions.. 56
Overview of the API Functions and Their Dependencies.. 13

DSCAN_GetChannelCapabilities

Purpose

To return the capabilities of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetChannelCapabilities(DSTCanHandle   tChannelHandle,
                                         unsigned long* pulChannelCapabilities);
```

Parameters (In)

tChannelHandle Channel handle.

Parameters (Out)

pulChannelCapabilities Specifies the channel capabilities as a combination of one or more of the following parameter values.

The following constants are predefined:

Predefined Constant	Description
DSCAN_CHANNEL_CAPABILITY_FD Value: `0x00000001`	The channel supports CAN FD.[1]
DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO Value: `0x00000008`	Relevant only if DSCAN_CHANNEL_CAPABILITY_FD is set. ▪ If DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO is set, the channel uses the non-ISO CAN FD communication[1]. ▪ If DSCAN_CHANNEL_CAPABILITY_FD_NON_ISO is not set, the channel uses the ISO CAN FD communication.
DSCAN_CHANNEL_CAPABILITY_FIXED_CONTROLLER_CONFIGURATION Value: `0x00000002`	The channel supports only fixed CAN controller configuration. Baud rate settings cannot be modified.
DSCAN_CHANNEL_CAPABILITY_BUS_LOAD_INFO Value: `0x00000004`	The channel supports bus load information.
DSCAN_CHANNEL_CAPABILITY_BUS_STATISTICS Value: `0x00000010`	The channel supports bus statistics.

[1] For more information on CAN FD, refer to Basics on CAN FD on page 16.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

DSCAN_IsChannelAccessible

Purpose

To check if a CAN channel is accessible.

Syntax

```
DSTCanError DSCAN_IsChannelAccessible(DSTCanHandle tChannelHandle,
                                      bool*        pbChannelIsAccessible);
```

Description

If the function returns that the channel is not accessible, the channel cannot be used any more since it has no longer connection to the corresponding hardware CAN channel.

In this case:

1. Call `DSCAN_UnregisterChannel` to unregister the channel.
2. Use `DSCAN_GetAvailableChannels` to check whether the hardware CAN channel is available.
3. If it is, call `DSCAN_RegisterChannel` to register the new channel.

Parameters (In)

tChannelHandle Channel handle.

Parametes (Out)

pbChannelIsAccessible Flag indicating if the channel is accessible.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions... 56
Overview of the API Functions and Their Dependencies.. 13

DSCAN_SetBaudrate

Purpose

To set the baud rate of a CAN interface channel.

Note

To call the function you must have access permission for the CAN interface channel.

Syntax

```
DSTCanError DSCAN_SetBaudrate(DSTCanHandle                tChannelHandle,
                              unsigned long               ulClockFrequency,
                              DSSCanBitTimingParameters*  ptBitTimingParameters,
                              DSSCanBitTimingParameters*  ptBitTimingParameters_FD);
```

Description

Depending on how the channel was initialized, the bit timing parameters `ptBitTimingParameters` and `ptBitTimingParameters_FD` have different meanings.

	Meaning of ...	
	`ptBitTimingParameters`	**`ptBitTimingParameters_FD`**
CAN FD is not to be used	Nominal baud rate	Is ignored. Use the NULL pointer instead.
CAN FD is to be used	CAN FD arbitration baud rate	CAN FD data baud rate

Parameters (In)

tChannelHandle Channel handle.

ulClockFrequency Clock frequency.

For classic CAN baud rates (up to 1 Mbit/s) and SJA1000-compatible CAN controllers, the clock frequency is always 8 MHz, so you can use the `DSCAN_CLOCK_FREQUENCY_SJA1000` definition.

ptBitTimingParameters Bit timing parameters.

ptBitTimingParameters_FD Bit timing parameters describing the CAN FD data baud rate.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates.......... 57
Basics on Configuration Functions.......... 56
Overview of the API Functions and Their Dependencies.......... 13

DSCAN_GetBaudrate

Purpose

To return the baud rate of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetBaudrate(DSTCanHandle                tChannelHandle,
                              unsigned long*              pulClockFrequency,
                              DSSCanBitTimingParameters*  ptBitTimingParameters,
                              bool*                       pbFD,
                              DSSCanBitTimingParameters*  ptBitTimingParameters_FD);
```

Description

Depending on whether CAN FD is used, the bit timing parameters `ptBitTimingParameters` and `ptBitTimingParameters_FD` have different meanings.

	Meaning of ...	
	`ptBitTimingParameters`	**`ptBitTimingParameters_FD`**
CAN FD is not used	Nominal baud rate	- (not relevant; can be ignored)
CAN FD is used	CAN FD arbitration baud rate	CAN FD data baud rate

Parameters (In)

tChannelHandle Channel handle.

Parameters (Out)

pulClockFrequency Clock frequency.

ptBitTimingParameters Bit timing parameters.

pbFD Flag indicating if CAN FD is used.

ptBitTimingParameters_FD Bit timing parameters describing the CAN FD data baud rate.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates..57
Basics on Configuration Functions..56
Overview of the API Functions and Their Dependencies...13

DSCAN_SetTransmitAcknowledge

Purpose

To set the transmit acknowledge state of a CAN channel.

Syntax

```
DSTCanError DSCAN_SetTransmitAcknowledge(DSTCanHandle tChannelHandle,
                                         bool         bTransmitAcknowledge);
```

Description

- If the *transmit acknowledge* is *enabled*, the transmitting hardware CAN channel generates a transmit acknowledge message in its receive queue when the CAN messages was successfully received by another CAN bus member.

  A transmit acknowledge message is the copy of the transmitted CAN message with the `DSCAN_RX_MESSAGE_FLAG_TX_ACKNOWLEDGE` flag set.

  The transmit acknowledge is enabled by default.
- If the *transmit acknowledge* is *disabled*, no transmit acknowledge messages are generated.

Parameters (In)

tChannelHandle Channel handle.

bTransmitAcknowledge Flag indicating whether the transmit acknowledge is enabled.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

DSCAN_SetChannelOutput

Purpose

To set the output mode of a CAN channel.

> **Note**
>
> To call the function, you must have access permission for the channel.

Syntax

```
DSTCanError DSCAN_SetChannelOutput(DSTCanHandle tChannelHandle,
                                   bool         bSilentMode);
```

Description

- If the *silent mode* is *disabled*, the hardware CAN channel generates a receive acknowledge on the CAN bus whenever a CAN message was received successfully.

  The silent mode is disabled by default.

- If the *silent mode* is *enabled*, the hardware CAN channel neither generates receive acknowledges for incoming CAN messages nor transmits CAN messages.

Parameters (In)

tChannelHandle Channel handle.

bSilentMode Flag indicating whether the silent mode is enabled.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions..56
Overview of the API Functions and Their Dependencies..13

DSCAN_SetAcceptance

Purpose

To set the CAN message acceptance filter of a CAN channel.

Syntax

```
DSTCanError DSCAN_SetAcceptance(DSTCanHandle  tChannelHandle,
                                unsigned long ulCodeStd,
                                unsigned long ulMaskStd,
                                unsigned long ulCodeXtd,
                                unsigned long ulMaskXtd);
```

Description

Via the mask and code filter, you can specify which bits of the received CAN IDs are checked and which values are accepted:

- A **1** in the mask filter specifies that the corresponding bit of the received CAN ID is checked.
- If the checked bits of the CAN ID match the bits of the code filter, the CAN message is passed to the receive buffer.

Note

- Both CAN identifier types have their own code and mask filter definitions. However, only the code and mask filters for the currently active CAN identifier type are used. The active identifier type is specified via the `DSCAN_InitChannel` function.
- Suppose a CAN bus has messages of both the STD and the XTD identifier type. dSPACE CAN interfaces can receive messages of both identifier types regardless of the `tIdentifierType` parameter value that was passed to the `DSCAN_InitChannel` function. This requires further filtering in your application.
- Every message that corresponds to the specified filter is accepted. However, depending on the filter settings and the messages on the bus, even more messages can pass the filter. This requires further filtering in your application.

Example The following table shows you which mask and code filters you have to specify in three example cases:

Case	Required Mask Filter	Required Code Filter
Block all IDs	Check all bits...	... for a pattern that is not in use...

Standard identifier	`0xfff -> 1111 1111 1111`	`0xfff -> 1111 1111 1111`
		or
		`0x000 -> 0000 0000 0000`

Extended identifier	`0xffffffff`	`0x00000000`
		or
		`0xffffffff`

Block no ID	Do not check any bit...	... so that the code filter is ignored...
Standard identifier	`0x000 -> 0000 0000 0000`	`0x000 -> 0000 0000 0000`
Extended identifier	`0x00000000`	`0x00000000`
Allow only ID 0x00A	Check all bits...	... for the desired pattern...
Standard identifier	`0x7ff -> 0111 1111 1111`	`0x00a -> 0000 0000 1010`
Extended identifier	`0x1fffffff`	`0x0000000a`

Note

If you want to filter for several specific CAN IDs, you might not be able to define a mask and code filter that blocks all the undesired CAN IDs. This is the case if the differences between the desired CAN IDs are located in different bits. Below are rules for finding your optimal mask and code filters to filter for 1 … n CAN IDs.

Calculating mask and code filters Use the following formulas to calculate your mask and code filters if you want to receive the CAN IDs `ID(0)` … `ID(n)`:

```
code = ID(0) | ID(1) | .... | ID(n)
mask = 0x7ff                    // for standard identifiers
mask = 0x1fff ffff              // for extended identifiers
for (i = 0; i <= n; i++)
{
   mask = (~(ID(i) & mask) ^ (code & mask)) & mask
}
```

Note

The total number of different CAN IDs that pass the acceptance filter can be calculated via the number of `0`'s in the mask (see example below):
Number of CAN IDs = $2^{\text{Number of 0's}}$

Example Suppose you want to receive the two CAN IDs:

- `ID1 = 0x00A = 000 0000 1010`
- `ID2 = 0x056 = 000 0101 0110`

In this case, the code can be calculated as:

```
code = ID1 | ID2
code = 000 0000 1010 | 000 0101 0110
code = 000 0101 1110
```

The mask can be calculated as:

```
mask = 0x7ff = 111 1111 1111
```

ID1:

```
mask = (~(000 0000 1010 & 111 1111 1111) XOR
         (000 0101 1110 & 111 1111 1111)) & 111 1111 1111
mask = (-(000 0000 1010) XOR (000 0101 1110)) & 111 1111 1111
mask = ((111 1111 0101) XOR (000 0101 1110)) & 111 1111 1111
mask = (111 1010 1011) & 111 1111 1111
mask = 111 1010 1011
```

ID2:

```
mask = (~(000 0101 0110 & 111 1010 1011) XOR
         (000 0101 1110 & 111 1010 1011)) & 111 1010 1011
mask = (-(000 0000 0010) XOR (000 0000 1010)) & 111 1010 1011
mask = ((111 1111 1101) XOR (000 0000 1010)) & 111 1010 1011
mask = (111 1111 0111) & 111 1010 1011
mask = 111 1010 0011
```

Number of IDs that pass the acceptance filter:
$2^{\text{Number of 0's}} = 2^4 = 16$

Parameters (In)

tChannelHandle Channel handle.

ulCodeStd Code for standard CAN identifiers.

ulMaskStd Mask for standard CAN identifiers.

ulCodeXtd Code for extended CAN identifiers.

ulMaskXtd Mask for extended CAN identifiers.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

DSCAN_SetEventNotification

Purpose

To set the event notification of a CAN channel.

Syntax

```
DSTCanError DSCAN_SetEventNotification(DSTCanHandle   tChannelHandle,
                                       DSTEventHandle tEventHandle,
                                       unsigned long  ulReceiveQueueLevel);
```

Description

To enable the event notification, call the function with a valid Windows event handle. Call the function with a NULL pointer instead of the event handle to disable the event notification.

If the event notification is enabled, the CAN channel informs the application via the event when the specified number of CAN messages has been received.

> **Note**
>
> The following limitations apply:
>
> - dSPACE and Kvaser CAN interfaces:
>   The specified receive queue level is ignored since the interfaces do not support a receive queue level greater than 1. Instead, 1 is always taken.
> - Eberspächer CAN interfaces:
>   The specified receive queue level is ignored since the interfaces do not support notification by receive queue level. Instead, the event is triggered every 5 ms, even if no new CAN messages have been received.

Parameters (In)

tChannelHandle Channel handle.

tEventHandle Windows event handle.

ulReceiveQueueLevel Receive queue level for event triggering.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions.. 56
Overview of the API Functions and Their Dependencies.. 13

DSCAN_EnableBusStatistics

Purpose

To enable or disable the periodic generation of bus statistics messages for a CAN channel.

Syntax

```
DSTCanError DSCAN_EnableBusStatistics(DSTCanHandle tChannelHandle,
                                      bool         bEnable);
```

Description

If the option is enabled, the hardware CAN channel periodically generates a bus statistics message in its receive queue.

If the option is disabled, no bus statistics messages are generated.

Parameters

tChannelHandle	Channel handle.
bEnable	Bus statistics enable.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Configuration Functions.. 56
Overview of the API Functions and Their Dependencies.. 13

Communication

Where to go from here

Information in this section

Data Types, Constants

Where to go from here

Information in this section

CAN Message TX Flags

CAN message TX flags

The following constants are predefined:

Flag	Value	Description
DSCAN_TX_MESSAGE_FLAG_FD	0x00000001	Transmit CAN FD message
DSCAN_TX_MESSAGE_FLAG_FD_BAUDRATE_SWITCH	0x00000002	Transmit CAN FD message with baud rate switch

CAN Message RX Flags

CAN message RX flags

The following constants are predefined:

Flag	Value	Description
DSCAN_RX_MESSAGE_FLAG_TX_ACKNOWLEDGE	0x00000001	Transmit acknowledge CAN message
DSCAN_RX_MESSAGE_FLAG_FD	0x00000100	CAN FD message
DSCAN_RX_MESSAGE_FLAG_FD_BAUDRATE_SWITCH	0x00000200	CAN FD message transmitted with baud rate switch
DSCAN_RX_MESSAGE_FLAG_RX_BUFFER_OVERRUN	0x00010000	Vendor API receive buffer overrun
DSCAN_RX_MESSAGE_FLAG_HW_RX_BUFFER_OVERRUN	0x00020000	CAN controller receive buffer overrun
DSCAN_RX_MESSAGE_FLAG_FD_ERROR_STATE_INDICATOR	0x00040000	CAN FD error state indicator

CAN Bus Statistics Flags

CAN bus statistics flags

The following constants are predefined:

Flag	Value	Description
DSCAN_BUS_STATISTICS_FLAG_ERROR_FRAMES	0x00000001	The bus statistics frame provides the value for the number of error frames
DSCAN_BUS_STATISTICS_FLAG_RX_STD_FRAMES	0x00000002	The bus statistics frame provides the value for the number of received standard CAN frames
DSCAN_BUS_STATISTICS_FLAG_TX_STD_FRAMES	0x00000004	The bus statistics frame provides the value for the number of transmitted standard CAN frames
DSCAN_BUS_STATISTICS_FLAG_RX_EXT_FRAMES	0x00000008	The bus statistics frame provides the value for the number of received extended CAN frames
DSCAN_BUS_STATISTICS_FLAG_TX_EXT_FRAMES	0x00000010	Bus statistics value for the number of available transmitted extended CAN frames
DSCAN_BUS_STATISTICS_FLAG_RX_STD_FD_FRAMES	0x00000020	Bus statistics value for the number of available received standard CAN FD frames
DSCAN_BUS_STATISTICS_FLAG_TX_STD_FD_FRAMES	0x00000040	Bus statistics value for the number of available transmitted standard CAN FD frames
DSCAN_BUS_STATISTICS_FLAG_RX_EXT_FD_FRAMES	0x00000080	Bus statistics value for the number of available received extended CAN FD frames
DSCAN_BUS_STATISTICS_FLAG_TX_EXT_FD_FRAMES	0x00000100	Bus statistics value for the number of available transmitted extended CAN FD frames

Enumerations

Enumerations

DSECanBusStatus CAN bus states

The following CAN bus states are predefined:

Enumerator	Value	Description
DSCAN_BUS_STATUS_UNKNOWN	0	Unknown
DSCAN_BUS_STATUS_ACTIVE	1	Bus active
DSCAN_BUS_STATUS_PASSIVE	2	Bus passive
DSCAN_BUS_STATUS_WARNING	3	Bus warning
DSCAN_BUS_STATUS_BUS_OFF	4	Bus off

DSECanMessageType CAN message type

The following CAN message types are predefined:

Enumerator	Value	Description
DSCAN_MESSAGE_TYPE_DATA	1	Data CAN message
DSCAN_MESSAGE_TYPE_REMOTE	2	Remote CAN message
DSCAN_MESSAGE_TYPE_ERROR	3	Error CAN message

Enumerator	Value	Description
DSCAN_MESSAGE_TYPE_BUS_INFO	4	CAN bus info message
DSCAN_MESSAGE_TYPE_BUS_STATISTICS	5	CAN bus statistics message

Structures

Where to go from here

Information in this section

DSSCanBusInfo

Purpose

Structure for CAN bus information

Syntax

```
typedef struct DSSCanBusInfo
{
    DSECanBusStatus tBusStatus;
    unsigned short  usRxErrorCounter;
    unsigned short  usTxErrorCounter;
    unsigned char   ucBusLoad;
} DSSCanBusInfo;
```

Parameters

tBusStatus CAN bus status

The following CAN bus states are predefined:

Enumerator	Value	Description
DSCAN_BUS_STATUS_UNKNOWN	0	Unknown
DSCAN_BUS_STATUS_ACTIVE	1	Bus active
DSCAN_BUS_STATUS_PASSIVE	2	Bus passive
DSCAN_BUS_STATUS_WARNING	3	Bus warning
DSCAN_BUS_STATUS_BUS_OFF	4	Bus off

usRxErrorCounter Receive errors counter

usTxErrorCounter Transmit errors counter

ucBusLoad CAN bus load in percent (only for channels having the `DSCAN_CHANNEL_CAPABILITY_BUS_LOAD_INFO` channel capability)

DSSCanMessage

Purpose

Structure for a CAN message

Syntax

```
typedef struct DSSCanMessage
{
DSECanMessageType    tMessageType;
    unsigned __int64     ui64Timestamp;
    unsigned long        ulCanIdentifier;
    DSECanIdentifierType tCanIdentifierType;
    unsigned long        ulFlags;
    unsigned short       usDLC;
    unsigned char        ucData[DSCAN_MAX_DATA_LENGTH];
    DSSCanBusInfo        tBusInfo;
} DSSCanMessage;
```

Parameters

tMessageType Message type

The following CAN message types are predefined:

Enumerator	Value	Description
DSCAN_MESSAGE_TYPE_DATA	1	Data CAN message
DSCAN_MESSAGE_TYPE_REMOTE	2	Remote CAN message
DSCAN_MESSAGE_TYPE_ERROR	3	Error CAN message
DSCAN_MESSAGE_TYPE_BUS_INFO	4	CAN bus info message
DSCAN_MESSAGE_TYPE_BUS_STATISTICS	5	CAN bus statistics message

Note

CAN bus statistics messages, i.e., messages with the `DSCAN_MESSAGE_TYPE_BUS_STATISTICS` message type, provide:

- *CAN bus information* via the `tBusInfo` parameter
- *CAN bus statistics information* via the `ucData` parameter

To encode the CAN bus statistics information from data bytes, use the `DSCAN_EncodeBusStatistics` function.

ui64Timestamp Timestamp

For dSPACE CAN interfaces, you can specify the transmission time of a CAN message by setting the CAN message timestamp.

To transmit a CAN message immediately, set the CAN message timestamp to '0'. When you first pass a CAN message with specified CAN message timestamp to the CAN channel and then one for immediate transmission, the latter message is delayed until the first message is sent.

ulCanIdentifier CAN identifier

tCanIdentifierType CAN identifier type

The following CAN identifier types are predefined:

Enumerator	Value	Description
DSCAN_IDENTIFIER_TYPE_STD	0x01	Standard CAN identifier (11 bits)
DSCAN_IDENTIFIER_TYPE_XTD	0x02	Extended CAN identifier (29 bits)
DSCAN_IDENTIFIER_TYPE_STD_XTD	DSCAN_IDENTIFIER_TYPE_STD | DSCAN_IDENTIFIER_TYPE_XTD	Both standard and extended CAN identifier

ulFlags Flags (combination of `DSCAN_RX_MESSAGE_FLAG_xxx` for `DSCAN_ReadReceiveQueue` and `DSCAN_TX_MESSAGE_FLAG_xxx` for `DSCAN_TransmitMessages`).

usDLC Data length code.

To convert between the data length code (DLC) and the data bytes count of a CAN message, use the functions `DSCAN_ConvertByteCountToDlc` and `DSCAN_ConvertDlcToByteCount`.

ucData Specifies either data bytes (only for messages with the `DSCAN_MESSAGE_TYPE_DATA` message type) or CAN bus statistics information (only for messages with the `DSCAN_MESSAGE_TYPE_BUS_STATISTICS` message type).

To encode the CAN bus statistics information from data bytes, use the `DSCAN_EncodeBusStatistics` function.

tBusInfo Specifies CAN bus information (only for messages with the `DSCAN_MESSAGE_TYPE_BUS_INFO` or `DSCAN_MESSAGE_TYPE_BUS_STATISTICS` message type).

Refer to `DSSCanBusInfo` on page 81.

Related topics

References

DSCAN_EncodeBusStatistics..116

DSSCanBusStatistics

Purpose

Structure for CAN bus statistics information

Syntax

```
typedef struct DSSCanBusStatistics
{
    unsigned long ulFlags;
    unsigned long ulErrorFrames;
    unsigned long ulRxStdFrames;
    unsigned long ulTxStdFrames;
    unsigned long ulRxExtFrames;
    unsigned long ulTxExtFrames;
    unsigned long ulRxStdFDFrames;
    unsigned long ulTxStdFDFrames;
    unsigned long ulRxExtFDFrames;
    unsigned long ulTxExtFDFrames;
} DSSCanBusStatistics;
```

Parameters

ulFlags Flags (combination of `DSCAN_BUS_STATISTICS_FLAG_xxx`).

ulErrorFrames Number of error frames

ulRxStdFrames Number of received standard CAN frames

ulTxStdFrames Number of transmitted standard CAN frames

ulRxExtFrames Number of received extended CAN frames

ulTxExtFrames Number of transmitted extended CAN frames

ulRxStdFDFrames Number of received standard CAN FD frames

ulTxStdFDFrames Number of transmitted standard CAN FD frames

ulRxExtFDFrames Number of received extended CAN FD frames

ulTxExtFDFrames Number of transmitted extended CAN FD frames

Related topics

References

CAN Bus Statistics Flags.. 79

Functions

Where to go from here

Information in this section

Basics on Communication Functions

Steps to perform CAN communication with a channel

To receive and transmit CAN messages with a channel, perform the following steps:

1. Call the `DSCAN_ActivateChannel` function to activate the CAN channel.

   > **Note**
   >
   > You must activate a channel before you can use it to transmit and receive CAN messages.

2. Call the `DSCAN_TransmitMessages` function to transmit CAN messages.

   The dSPACE CAN API 2.0 lets you transmit messages of the following message types:

   - `DSCAN_MESSAGE_TYPE_DATA`
   - `DSCAN_MESSAGE_TYPE_REMOTE` (for classic CAN only)

   Messages are transmitted in the order you pass them to the CAN channel, i.e., the send buffer is a first-in-first-out buffer.

3. To reconfigure a CAN channel, call the `DSCAN_DeactivateChannel` function to deactivate the channel beforehand.

Specifying the transmission time

For dSPACE CAN interfaces, you can specify the transmission time of a CAN message by setting the CAN message timestamp.

To transmit a CAN message immediately, set the CAN message timestamp to '0'. When you first pass a CAN message with specified CAN message timestamp to the CAN channel and then one for immediate transmission, the latter message is delayed until the first message is sent.

See the following example.

Example

The following example shows how to receive and transmit CAN messages with a channel.

```
DSTCanError      tErrorCode         = DSCAN_ERR_NO_ERROR;
DSTCanHandle     tChannelHandle     = DSCAN_INVALID_CAN_HANDLE;
unsigned long    ulMessagesCount    = 0;
DSSCanMessage*   ptMessagesArray    = NULL;
DSSCanMessage    tMessage;
unsigned __int64 ui64TimeResolution = 0;
unsigned __int64 ui64CurrentTime    = 0;
// ...
// Receive CAN messages
// --------------------
// Get count of CAN messages
tErrorCode = DSCAN_GetReceiveQueueLevel(tChannelHandle, &ulMessagesCount);
if (DSCAN_ERR_NO_ERROR == tErrorCode)
{
    // Allocate memory for CAN messages
    ptMessagesArray = new DSSCanMessage[ulMessagesCount];
    // Get CAN messages
    tErrorCode = DSCAN_ReadReceiveQueue(tChannelHandle, &ulMessagesCount, ptMessagesArray);
    // Process CAN messages
    for (unsigned long i = 0; i < ulMessagesCount; i++)
    {
        // Do anything with ptMessagesArray[i]
    }
    // Free memory for CAN messages
    delete[] ptMessagesArray;
}
// Transmit CAN messages
// ---------------------
// Send 1 standard CAN message immediately
tMessage.tMessageType       = DSCAN_MESSAGE_TYPE_DATA;
tMessage.ulCanIdentifier    = 0x100;
tMessage.tCanIdentifierType = DSCAN_IDENTIFIER_TYPE_STD;
tMessage.ulFlags            = 0;
tMessage.usDLC              = DSCAN_ConvertByteCountToDlc(3);
memset(tMessage.ucData, 0, DSCAN_MAX_DATA_LENGTH);
for (unsigned char i = 0; i < 3; i++)
{
    tMessage.ucData[0] = i;
}
tMessage.ui64Timestamp = 0;
tErrorCode = DSCAN_TransmitMessages(tChannelHandle, 1, &tMessage);
// Get hardware time resolution
tErrorCode = DSCAN_GetHardwareTimeResolution(tChannelHandle, &ui64TimeResolution);
if (DSCAN_ERR_NO_ERROR == tErrorCode)
{
    // Allocate memory for 10 CAN messages
    ptMessagesArray = new DSSCanMessage[10];
    // Send 10 extended CAN FD messages with 100ms delay
    tErrorCode = DSCAN_GetHardwareTime(tChannelHandle, &ui64CurrentTime);
    if (DSCAN_ERR_NO_ERROR == tErrorCode)
    {
        for (int iMessageIndex = 0; iMessageIndex < 10; iMessageIndex++)
        {
            ptMessagesArray[iMessageIndex].tMessageType       = DSCAN_MESSAGE_TYPE_DATA;
            ptMessagesArray[iMessageIndex].ulCanIdentifier    = 0x100 + iMessageIndex;
            ptMessagesArray[iMessageIndex].tCanIdentifierType = DSCAN_IDENTIFIER_TYPE_XTD;
            ptMessagesArray[iMessageIndex].ulFlags            = DSCAN_TX_MESSAGE_FLAG_FD |
                                                                DSCAN_TX_MESSAGE_FLAG_FD_BAUDRATE_SWITCH;
            ptMessagesArray[iMessageIndex].usDLC              = DSCAN_ConvertByteCountToDlc(30);
            memset(ptMessagesArray[iMessageIndex].ucData, 0, DSCAN_MAX_DATA_LENGTH);
            for (unsigned char i = 0; i < 30; i++)
            {
                ptMessagesArray[iMessageIndex].ucData[0] = i;
            }
            ptMessagesArray[iMessageIndex].ui64Timestamp = (unsigned __int64)((((double)ui64CurrentTime / (double)
      ui64TimeResolution) + (0.100 * (iMessageIndex + 1))) * (double)ui64TimeResolution);
        }
        if (DSCAN_ERR_NO_ERROR == tErrorCode)
        {
            tErrorCode = DSCAN_TransmitMessages(tChannelHandle, 10, ptMessagesArray);
        }
    }
    // Free memory for CAN messages
    delete[] ptMessagesArray;
}
```

Related topics

Basics

DSCAN_ActivateChannel..88
DSCAN_TransmitMessages..95

References

DSCAN_DeactivateChannel..89

DSCAN_ActivateChannel

Purpose

To activate CAN communication of a specific CAN channel.

Syntax

```
DSTCanError DSCAN_ActivateChannel(DSTCanHandle tChannelHandle);
```

Description

If the CAN communication is activated, the CAN channel can send and receive CAN messages.

Parameters (In)

tChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions..86
Overview of the API Functions and Their Dependencies..13

DSCAN_DeactivateChannel

Purpose

To deactivate CAN communication of a specific CAN channel.

Syntax

```
DSTCanError DSCAN_DeactivateChannel(DSTCanHandle tChannelHandle);
```

Description

If the CAN communication is deactivated, the CAN channel cannot send and receive CAN messages.

Parameters (In)

tChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions..86
Overview of the API Functions and Their Dependencies...13

References

DSCAN_ReadReceiveQueueAndDeactivateChannel..93

DSCAN_GetHardwareTimeResolution

Purpose

To return the hardware time resolution of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetHardwareTimeResolution(DSTCanHandle      tChannelHandle,
                                             unsigned __int64* pui64TimeResolution);
```

Description

The hardware time resolution defines the time unit of the hardware time which can be obtained by `DSCAN_GetHardwareTime`.

Parameters (In)

tChannelHandle Channel handle.

Parameters (Out)

pui64TimeResolution Hardware time resolution.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions.. 86
Overview of the API Functions and Their Dependencies.............................. 13

DSCAN_GetHardwareTime

Purpose

To return the hardware time of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetHardwareTime(DSTCanHandle      tChannelHandle,
                                  unsigned __int64* pui64Time);
```

Description

The hardware time is measured in time units defined by the hardware time resolution which can be obtained by the `DSCAN_GetHardwareTimeResolution` function.

To calculate the hardware time in seconds, use the following formula:

$$Hardware\ time\ in\ seconds\ = \frac{Hardware\ time}{Hardware\ time\ resolution}$$

Parameters (In)

tChannelHandle Channel handle.

Parameters (Out)

pui64Time Current hardware time.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions.. 86
Overview of the API Functions and Their Dependencies.. 13

DSCAN_ResetHardwareTime

Purpose

To reset the hardware time of a CAN channel.

> **Note**
>
> This is not supported for CAN interfaces from dSPACE and Kvaser.

Syntax

```
DSTCanError DSCAN_ResetHardwareTime(DSTCanHandle tChannelHandle);
```

Parameters (In)

tChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions.. 86
Overview of the API Functions and Their Dependencies.. 13

DSCAN_GetReceiveQueueLevel

Purpose

To return the count of CAN messages in the receive queue of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetReceiveQueueLevel(DSTCanHandle   tChannelHandle,
                                       unsigned long* pulCanMessagesCount);
```

Parameters (In)

tChannelHandle Channel handle.

Parameters (Out)

pulCanMessagesCount Count of CAN messages in the receive queue.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions.. 86
Overview of the API Functions and Their Dependencies... 13

DSCAN_ReadReceiveQueue

Purpose

To read CAN messages from the receive queue of a CAN channel.

Syntax

```
DSTCanError DSCAN_ReadReceiveQueue(DSTCanHandle   tChannelHandle,
                                   unsigned long* pulCanMessagesCount,
                                   DSSCanMessage* ptCanMessagesArray);
```

Description

You can call the function with the NULL pointer instead of `ptCanMessagesArray`. In this case only the count of CAN messages in the receive queue is obtained. This function call is equal to a call of `DSCAN_GetReceiveQueueLevel`.

Parameters (In)

tChannelHandle Channel handle.

Parameters (In, Out)

pulCanMessagesCount Count of CAN messages in the receive queue.

- If used as In parameter:

  Lets you specify the maximum number of CAN messages to obtain. To obtain the information, call the function with `pulCanMessagesCount` set to the maximum number of CAN messages to get. This is usually the size of the `ptCanMessagesArray` parameter.

- If used as Out parameter:

  Lets you get the number of CAN messages which have been written to the `ptCanMessagesArray` parameter (the actual count is always less than or equal to the initial maximum value).

Parameters (Out)

ptCanMessagesArray CAN messages array.

The memory for the CAN messages array `ptCanMessagesArray` must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions... 86
Overview of the API Functions and Their Dependencies.. 13

References

DSCAN_ReadReceiveQueueAndDeactivateChannel.. 93

DSCAN_ReadReceiveQueueAndDeactivateChannel

Purpose

To read CAN messages from the receive queue of a CAN channel, and deactivate CAN communication of the channel.

Syntax

```
DSTCanError DSCAN_ReadReceiveQueueAndDeactivateChannel(DSTCanHandle   tChannelHandle,
                                                        unsigned long* pulCanMessagesCount,
                                                        DSSCanMessage* ptCanMessagesArray);
```

Description

Using the function has the same effect as sequentially calling the following functions:

1. DSCAN_ReadReceiveQueue
2. DSCAN_DeactivateChannel

Tip

Unlike calling the two functions above, calling the `DSCAN_ReadReceiveQueueAndDeactivateChannel` function ensures that the last received CAN message is a *bus statistics message* if bus statistics is enabled. As a consequence, the bus statistics information is complete and valid when you use this function.

You can call the function with the NULL pointer instead of `ptCanMessagesArray`. In this case only the count of CAN messages in the receive queue is obtained.

Parameters (In)

tChannelHandle Channel handle.

Parameters (In, Out)

pulCanMessagesCount Count of CAN messages in the receive queue.

- If used as In parameter:

  Lets you specify the maximum number of CAN messages to obtain. To obtain the information, call the function with `pulCanMessagesCount` set to the maximum number of CAN messages to get. This is usually the size of the `ptCanMessagesArray` parameter.
- If used as Out parameter:

  Lets you get the number of CAN messages that were written to the `ptCanMessagesArray` parameter (the actual count is always less than or equal to the initial maximum value).

Parameters (Out)

ptCanMessagesArray CAN messages array.

The memory for the CAN messages array `ptCanMessagesArray` must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions.. 86
Overview of the API Functions and Their Dependencies.............................. 13

References

DSCAN_DeactivateChannel.. 89
DSCAN_EnableBusStatistics.. 75
DSCAN_ReadReceiveQueue.. 92

DSCAN_FlushReceiveQueue

Purpose

To clear the receive queue of a CAN channel.

Syntax

```
DSTCanError DSCAN_FlushReceiveQueue(DSTCanHandle tChannelHandle);
```

Description

All CAN messages in the receive queue are deleted and cannot be read any more.

Parameters (In)

tChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions..86
Overview of the API Functions and Their Dependencies..13

DSCAN_TransmitMessages

Purpose

To transmit CAN messages by the use of a CAN channel.

Syntax

```
DSTCanError DSCAN_TransmitMessages(DSTCanHandle   tChannelHandle,
                                   unsigned long  ulCanMessagesCount,
                                   DSSCanMessage* ptCanMessagesArray);
```

Parameters (In)

tChannelHandle Channel handle.

ulCanMessagesCount Number of CAN messages to transmit.

Parameters (Out)

ptCanMessagesArray CAN messages array.

The memory for the CAN messages array `ptCanMessagesArray` must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions..86
Overview of the API Functions and Their Dependencies..13

DSCAN_FlushTransmitQueue

Purpose

To clear the transmit queue of a CAN channel.

Note

To call the function you must have access permission for the channel.

Syntax

```
DSTCanError DSCAN_FlushTransmitQueue(DSTCanHandle tChannelHandle);
```

Description

All CAN messages in the transmit queue are deleted and will not be transmitted.

Parameters (In)

tChannelHandle Channel handle.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions..86
Overview of the API Functions and Their Dependencies..13

DSCAN_GetBusInfo

Purpose

To return the bus communication state of a CAN channel.

Syntax

```
DSTCanError DSCAN_GetBusInfo(DSTCanHandle   tChannelHandle,
                             DSSCanBusInfo* ptBusInfo);
```

Description

A bus communication state is described by a `DSSCanBusInfo` structure which provides the following information:

- CAN bus status
- Receive error counter
- Transmit error counter
- CAN bus load in percent (for channels having the capability `DSCAN_CHANNEL_CAPABILITY_BUS_LOAD_INFO`)

Parameters (In)

tChannelHandle Channel handle.

ptBusInfo Bus communication state.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Communication Functions.......86
Overview of the API Functions and Their Dependencies.......13

References

CAN Channel Capabilities.......41
DSSCanBusInfo.......81

Error Handling

Where to go from here

Information in this section

Data Types, Constants

Error Codes

Error codes

The following constants are predefined:

Error Code	Value	Description
DSCAN_ERR_NO_ERROR	0x00000000	No error
DSCAN_ERR_INVALID_POINTER	0x00000001	Invalid pointer
DSCAN_ERR_OUT_OF_MEMORY	0x00000002	Out of memory
DSCAN_ERR_FUNCTION_NOT_SUPPORTED	0x00000003	Unsupported function
DSCAN_ERR_INVALID_PARAMETER	0x00000004	Invalid or unsupported parameter
DSCAN_ERR_CAN_FD_NOT_SUPPORTED	0x00000005	CAN FD is not supported
DSCAN_ERR_LOAD_VENDOR_API	0x00000006	Cannot load vendor CAN API
DSCAN_ERR_MAP_VENDOR_API_FUNCTIONS	0x00000007	Cannot map required vendor CAN API functions
DSCAN_ERR_VENDOR_API_NOT_SUPPORTED	0x00000008	Unsupported vendor CAN API version
DSCAN_ERR_VENDOR_API_NOT_LOADED	0x00000009	Cannot execute function. The vendor CAN API is not loaded.
DSCAN_ERR_CHANNEL_NOT_FOUND	0x0000000A	Cannot find specified channel
DSCAN_ERR_INVALID_CHANNEL_HANDLE	0x0000000B	Invalid channel handle
DSCAN_ERR_CHANNEL_NOT_INITIALIZED	0x0000000C	Cannot execute function. The channel is not initialized.
DSCAN_ERR_CHANNEL_ACTIVATED	0x0000000D	Cannot execute function while the channel is activated
DSCAN_ERR_CHANNEL_NOT_ACTIVATED	0x0000000E	Cannot execute function. The channel is not activated.
DSCAN_ERR_NO_ACCESS_PERMISSION	0x0000000F	Cannot execute function without access permission
DSCAN_ERR_GET_VENDOR_INFORMATION	0x00000010	Cannot get vendor information
DSCAN_ERR_GET_AVAILABLE_CHANNELS	0x00000011	Cannot get available channels
DSCAN_ERR_REGISTER_CHANNEL	0x00000012	Cannot register channel
DSCAN_ERR_VENDOR_SPECIFIC	0xFFFFFFFF	Vendor-specific CAN API error. To get the code and description of a vendor-specific CAN API error, you can use the `DSCAN_GetLastVendorSpecificError` function.

Functions

Where to go from here

Information in this section

Basics on Error Handling Functions

dSPACE CAN API errors

You can get the error description for dSPACE CAN API error codes via the `DSCAN_GetErrorText` function.

Vendor CAN API errors

If a dSPACE CAN API function returns the `DSCAN_ERR_VENDOR_SPECIFIC` error code, an error occurred in the vendor-specific CAN API. In this case, you can get the code and description of the vendor-specific CAN API error via the `DSCAN_GetLastVendorSpecificError` function.

Related topics

References

DSCAN_GetLastVendorSpecificError..102

DSCAN_GetErrorText

Purpose

To get the error description for a dSPACE CAN API error code.

Syntax

```
DSTCanError DSCAN_GetErrorText(DSTCanError tErrorCode,
                               char*       pszErrorText);
```

Description

Returns the description of a dSPACE CAN API error code.

Parameters (In)

tErrorCode Error code. Refer to DSTCanError on page 26.

Parameters (Out)

pszErrorText Error description

The size of this parameter must be at least `DSCAN_MAX_TEXT_LENGTH` bytes. The memory must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Error Handling Functions..101
Overview of the API Functions and Their Dependencies...13

DSCAN_GetLastVendorSpecificError

Purpose

To get the code and description of a vendor-specific CAN API error.

Syntax

```
DSTCanError DSCAN_GetLastVendorSpecificError(DSTCanHandle tChannelHandle,
                                             long*        plVendorErrorCode,
                                             char*        pszVendorErrorText);
```

Description

If a dSPACE CAN API function returns the `DSCAN_ERR_VENDOR_SPECIFIC` error code, an error occurred in the vendor-specific CAN API. In this case, you can get the code and description of the vendor-specific CAN API error via the `DSCAN_GetLastVendorSpecificError` function.

Parameters (In)

tChannelHandle Channel handle.

Parameters (Out)

plVendorErrorCode Code of the vendor-specific CAN API error.

pszVendorErrorText Description of the vendor-specific CAN API error.
The size of this parameter must be at least `DSCAN_MAX_TEXT_LENGTH` bytes. The memory must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Error Handling Functions 101
Overview of the API Functions and Their Dependencies 13

References

DSCAN_GetLastVendorSpecificError 102

Auxiliary

Functions

Where to go from here

Information in this section

DSCAN_ConvertBaudrateToBitTimingParameters

Purpose

To convert the baud rate to bit timing parameters.

Syntax

```
DSTCanError DSCAN_ConvertBaudrateToBitTimingParameters(unsigned long              ulClockFrequency,
                                                       unsigned long              ulBaudrate,
                                                       DSSCanBitTimingParameters* ptBitTimingParameters);
```

Description

- For the most commonly used classic CAN baud rate values (`DSCAN_BAUDRATE_xxx`) and the clock frequency of 8 MHz (`DSCAN_CLOCK_FREQUENCY_SJA1000`) used for classic CAN baud rates and SJA1000-compatible CAN controllers, the `DSCAN_ConvertBaudrateToBitTimingParameters` function returns the following predefined bit timing parameters:

Baud Rate	Synch. Jump Width	Baud Rate Prescaler	Sample Mode	Bit Time Segment 1	Bit Time Segment 2	Sample Point
1000 kBit/s	2	1	0	5	2	75%
500 kBit/s	2	2	0	5	2	75%
250 kBit/s	2	4	0	5	2	75%
125 kBit/s	2	8	0	5	2	75%
100 kBit/s	2	10	0	5	2	75%
50 kBit/s	2	20	0	5	2	75%
20 kBit/s	2	50	0	5	2	75%
10 kBit/s	2	50	0	13	2	87%

- For all other combinations of baud rate and clock frequency values, the bit timing parameters are determined according to the following conditions:
  - Sample point is equal or close to:
    - 75 % for classic CAN baud rates (≤ 1 Mbit/s)
    - 80 % for CAN FD baud rates (> 1 Mbit/s)
  - Lowest nominal bit time for the desired sample point
  - Sample mode is 0

Parameters (In)

ulClockFrequency Clock frequency.

For classic CAN baud rates (up to 1 Mbit/s) and SJA1000-compatible CAN controllers, the clock frequency is always 8 MHz, so you can use the `DSCAN_CLOCK_FREQUENCY_SJA1000` definition.

ulBaudrate Baud rate

Parameters (Out)

ptBitTimingParameters Bit timing parameters.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

DSCAN_ConvertBaudratesToBitTimingParameters

Purpose

To convert two baud rates to bit timing parameters with identical sample point.

Syntax

```
DSTCanError DSCAN_ConvertBaudratesToBitTimingParameters(unsigned long                ulClockFrequency,
                                                        unsigned long                ulReferenceSamplePoint,
                                                        unsigned long                ulBaudrate_1,
                                                        unsigned long                ulBaudrate_2,
                                                        DSSCanBitTimingParameters* ptBitTimingParameters_1,
                                                        DSSCanBitTimingParameters* ptBitTimingParameters_2);
```

Parameters (In)

ulClockFrequency Clock frequency.

For classic CAN baud rates (up to 1 Mbit/s) and SJA1000-compatible CAN controllers, the clock frequency is always 8 MHz, so you can use the `DSCAN_CLOCK_FREQUENCY_SJA1000` definition.

ulReferenceSamplePoint Reference sample point

The reference sample point is applied to the bit timing parameters of both baud rates during conversion.

ulBaudrate_1 Baud rate 1

ulBaudrate_2 Baud rate 2

Parameters (Out)

ptBitTimingParameters_1 Bit timing parameters of baud rate 1.

ptBitTimingParameters_2 Bit timing parameters of baud rate 2.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates.. 57
Overview of the API Functions and Their Dependencies.. 13

DSCAN_ConvertBaudrateToBitTimingParametersWithSameSPAndBRP

Purpose

To convert a baud rate to bit timing parameters whereof the same sample point and the baud rate prescaler are set to reference values.

Syntax

```
DSTCanError DSCAN_ConvertBaudrateToBitTimingParametersWithSameSPAndBRP(
                                                       unsigned long              ulClockFrequency,
                                                       DSSCanBitTimingParameters* ptReferenceBitTimingParameters,
                                                       unsigned long              ulBaudrate,
                                                       DSSCanBitTimingParameters* ptBitTimingParameters);
```

Parameters (In)

ulClockFrequency Clock frequency.

For classic CAN baud rates (up to 1 Mbit/s) and SJA1000-compatible CAN controllers, the clock frequency is always 8 MHz, so you can use the `DSCAN_CLOCK_FREQUENCY_SJA1000` definition.

ptReferenceBitTimingParameters Reference bit timing parameters.

The sample point and the baud rate prescaler of the reference bit timing parameters are applied to the bit timing parameters during conversion.

ulBaudrate Baud rate

Parameters (Out)

ptBitTimingParameters Bit timing parameters.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates.. 57
Overview of the API Functions and Their Dependencies.. 13

DSCAN_ConvertBusTimingRegistersToBitTimingParameters

Purpose

To convert bit timing registers to bit timing parameters.

Syntax

```
DSTCanError DSCAN_ConvertBusTimingRegistersToBitTimingParameters(unsigned char              ucBTR0,
                                                                 unsigned char              ucBTR1,
                                                                 DSSCanBitTimingParameters* ptBitTimingParameters);
```

Description

For classic CAN, bit timing parameters can be packed to bus timing registers (BTRs):

BTR0

7	6	5	4	3	2	1	0
SJW[1] - 1		BRP[2] - 1					

[1] Synchronization jump width

[2] Baud rate prescaler

BTR1

7	6	5	4	3	2	1	0
SAM[1]	TSEG2[2] - 1			TSEG1[3] - 1			

[1] Sample mode

[2] Bit time segment 2

[3] Bit time segment 1

Parameters (In)

ucBTR0 Bus timing register 1.

ucBTR1 Bus timing register 2.

Parameters (Out)

ptBitTimingParameters Bit timing parameters.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates.. 57
Overview of the API Functions and Their Dependencies.. 13

DSCAN_ConvertBitTimingParametersToBaudrate

Purpose

To convert bit timing parameters to baud rate.

Syntax

```
DSTCanError DSCAN_ConvertBitTimingParametersToBaudrate(unsigned long              ulClockFrequency,
                                                       DSSCanBitTimingParameters* ptBitTimingParameters,
                                                       unsigned long*             pulBaudrate);
```

Description

The baud rate value is calculated via the following formula:

$$\text{Baud rate} = \frac{Clock\ frequency}{Baud\ rate\ prescaler\ \cdot (1 + Bit\ time\ segment\ 1 + Bit\ time\ segment\ 2)}$$

Parameters (In)

ulClockFrequency Clock frequency.

For classic CAN baud rates (up to 1 Mbit/s) and SJA1000-compatible CAN controllers, the clock frequency is always 8 MHz, so you can use the `DSCAN_CLOCK_FREQUENCY_SJA1000` definition.

ptBitTimingParameters Bit timing parameters.

Parameters (Out)

pulBaudrate Baud rate.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates........ 57
Overview of the API Functions and Their Dependencies........ 13

DSCAN_ConvertBitTimingParametersToBusTimingRegisters

Purpose

To convert bit timing parameters to bit timing registers.

Syntax

```
DSTCanError DSCAN_ConvertBitTimingParametersToBusTimingRegisters(DSSCanBitTimingParameters* ptBitTimingParameters,
                                                                 unsigned char*             pucBTR0,
                                                                 unsigned char*             pucBTR1);
```

Description

For classic CAN, bit timing parameters can be packed to bus timing registers (BTRs):

BTR0

7	6	5	4	3	2	1	0
SJW[1] - 1		BRP[2] - 1					

[1] Synchronization jump width

[2] Baud rate prescaler

BTR1

7	6	5	4	3	2	1	0
SAM[1]	TSEG2[2] - 1			TSEG1[3] - 1			

[1] Sample mode

[2] Bit time segment 2

[3] Bit time segment 1

Parameters (In)

ptBitTimingParameters Bit timing parameters.

Parameters (Out)

pucBTR0 Bus timing register 1.

pucBTR1 Bus timing register 2.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Basics on Bit Timing Parameters and Baud Rates.. 57
Overview of the API Functions and Their Dependencies... 13

DSCAN_ConvertByteCountToDlc

Purpose

To convert data byte count to DLC.

Syntax

```
unsigned short DSCAN_ConvertByteCountToDlc(unsigned short usByteCount);
```

Description

The conversion between the data length code (DLC) and data byte count is performed according to the following mapping:

DLC	Byte count
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	12
10	16
11	20
12	24
13	32
14	48
15	64

Parameters (In)

usByteCount Data byte count.

Return value

unsigned short - Data length code (DLC).

Related topics

Basics

Overview of the API Functions and Their Dependencies.. 13

DSCAN_ConvertDlcToByteCount

Purpose

To convert DLC to data byte count.

Syntax

```
unsigned short DSCAN_ConvertDlcToByteCount(unsigned short usDLC);
```

Description

The conversion between the data length code (DLC) and data byte count is performed according to the following mapping:

DLC	Byte count
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	12
10	16
11	20
12	24
13	32
14	48
15	64

Parameters (In)

usDLC Data length code (DLC).

Return value

unsigned short - Data byte count.

Related topics

Basics

Overview of the API Functions and Their Dependencies.. 13

DSCAN_CalculateAcceptanceFilter

Purpose

To calculate CAN acceptance code and mask for an array of CAN identifiers.

Syntax

```
DSTCanError DSCAN_CalculateAcceptanceFilter(unsigned long*       pulCanIdentifiersArray,
                                            unsigned long        ulCanIdentifiersCount,
                                            DSECanIdentifierType tCanIdentifiersType,
                                            unsigned long*       pulCode,
                                            unsigned long*       pulMask);
```

Description

Refer to `DSCAN_SetAcceptance` on page 71 for the detailed description of acceptance filter code and mask.

Parameters (In)

pulCanIdentifiersArray CAN identifiers array.

ulCanIdentifiersCount CAN identifiers count.

tCanIdentifiersType CAN identifiers type.

Parameters (Out)

pulCode Acceptance filter code.

pulMask Acceptance filter mask.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Overview of the API Functions and Their Dependencies.. 13

References

DSCAN_SetAcceptance... 71

DSCAN_MergeAcceptanceFilter

Purpose

To merge two CAN acceptance codes and masks.

Syntax

```
DSTCanError DSCAN_MergeAcceptanceFilter(unsigned long        ulCode1,
                                        unsigned long        ulMask1,
                                        unsigned long        ulCode2,
                                        unsigned long        ulMask2,
                                        DSECanIdentifierType tCanIdentifiersType,
                                        unsigned long*       pulResultCode,
                                        unsigned long*       pulResultMask);
```

Description

Refer to `DSCAN_SetAcceptance` for the detailed description of acceptance filter code and mask.

Parameters (In)

ulCode1 Acceptance filter code 1.

ulMask1 Acceptance filter mask 1.

ulCode2 Acceptance filter code 2.

ulMask2 Acceptance filter mask 2.

tCanIdentifiersType CAN identifiers type.

Parameters (out)

pulResultCode Resulting acceptance filter code.

pulResultMask Resulting acceptance filter mask.

Return value

One of the error codes defined in DSTCanError.

Related topics

References

DSCAN_SetAcceptance.. 71

DSCAN_EncodeBusStatistics

Purpose

To encode CAN bus statistics information from data bytes of a CAN bus statistics message.

Syntax

```
DSTCanError DSCAN_EncodeBusStatistics(unsigned char       ucDataBytes[DSCAN_MAX_DATA_LENGTH],
                                      DSSCanBusStatistics* ptBusStatistics);
```

Parameters (In)

ucDataBytes Data bytes of a CAN bus statistics message

Parameters (Out)

ptBusStatistics CAN bus statistics

Return value

One of the error codes defined in DSTCanError.

Related topics

References

DSSCanBusStatistics.. 84

DSCAN_ConvertApiVersionToString

Purpose

To convert API version to string.

Syntax

```
DSTCanError DSCAN_ConvertApiVersionToString(unsigned long ulApiVersion,
                                            char*         pszApiVersion);
```

Parameters (In)

ulApiVersion CAN API version.

Parameters (Out)

pszApiVersion CAN API version string.

The size of this parameter must be at least `DSCAN_MAX_TEXT_LENGTH` bytes. The memory must be allocated and freed by the caller.

Return value

One of the error codes defined in DSTCanError.

Related topics

Basics

Overview of the API Functions and Their Dependencies.. 13

Appendix

Where to go from here

Information in this section

Troubleshooting

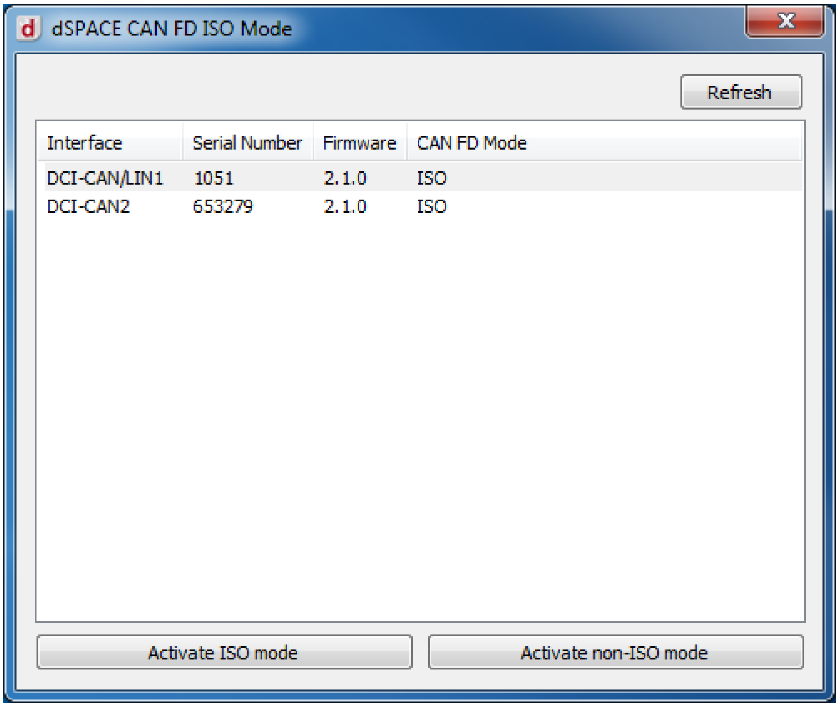
Problem in connection with incompatible CAN FD protocols

Currently, there are two CAN FD protocols on the market, which are not compatible with each other.

- The *non-ISO CAN FD protocol* represents the original CAN FD protocol from Bosch.
- The *ISO CAN FD protocol* represents the CAN FD protocol according to the ISO 11898-1:2015 standard.

The DCI-CAN2 and the DCI-CAN/LIN1 support both CAN FD protocols.

Switching between ISO CAN FD and non-ISO CAN FD To switch between ISO CAN FD and non-ISO CAN FD, you can use the dSPACE CAN FD ISO Mode (`DsCanFdIsoMode.exe`) tool. It is installed in the `C:\Program Files <(x86)>\Common Files\dSPACE\ DSCanApi_<Version>\` folder.



Interface	Serial Number	Firmware	CAN FD Mode
DCI-CAN/LIN1	1051	2.1.0	ISO
DCI-CAN2	653279	2.1.0	ISO

Limitations

Maximum number of CAN interface channels

- When you use the dSPACE CAN API in connection with CAN interfaces from dSPACE, you can use at most 16 CAN interface channels simultaneously.
- When you use the dSPACE CAN API in connection with CAN interfaces from other vendors, the maximum number of CAN interface channels depends on the vendor's CAN driver software.

Maximum number of clients

- When you use the dSPACE CAN API in connection with CAN interfaces from dSPACE, the number of clients is limited to 32.

  This applies to both CAN and CAN FD.
- When you use the dSPACE CAN API in connection with CAN interfaces from other vendors, the maximum number of clients depends on the vendor's CAN driver software.