Protocol API

CAN Data Link

Packet Interface

1.0.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC110207API03EN | Revision 3 | English | 2013-05 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This manual describes the application interface of the CAN DL task designed for use with the CANopen Slave from V3.0.x.x and CANopen Master from V2.7.x.x protocol stacks.

The CAN_DL Task handles the interface of the XC and is responsible for sending and receiving of CAN-Frames, configuration and processing events.

1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
1	2011-04-12	RG/EZ	all	Created
2	2012-02-07	RG/HH	1.5.1	Integrated in CANopen Master from V2.7.x.x Integrated in CANopen Slave from V3.0.x.x
3	2013-05-21	RG	2.4	Correction of COB-ID example

Table 1: List of Revisions

1.3 Intended Audience

This manual is suitable for software developers with the following background:

- Knowledge of the programming language C
- Knowledge of the use of the real-time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model
- Knowledge of the CAN (or DeviceNet) protocol

1.4 System Requirements

This software package has following system requirements to its environment:

- netX chip as CPU hardware platform
- operating system for task scheduling required

1.5 Specifications

1.5.1 Technical Data

Features	Parameter
Baud rates	10 kBaud to 1000 kBaud
Frame types	11 Bits (Standard frame) 29 Bits (Extended frame)
Operation modes	- Normal RxTx mode - Listen only - Additional monitoring of up to 2 IO-pins
Filter functionality	Filtering of Identifier with Code and Mask
Max. number of applications	5
Availability	netX10, netX50, netX51,netX100/500
Interfaces	Packets and/or Functions
Integrated in CANopen Master Firmware	V2.7.x.x and higher
Integrated in CANopen Slave Firmware	V3.0.x.x and higher

Table 1: Technical Data of CAN-DL-Task

1.6 Terms, Abbreviations and Definitions

Term	Description
AP	Application (task)
CAN	Controller Area Network
DL	Data Link Layer
DLC	Data length code
RTR	Remote transmission request
Rx	Receive
Tx	Transmission

Table 2: Terms, Abbreviations and Definitions

All variables, parameters, and data used in this manual have the LSB/MSB (“Intel”) data format. This corresponds to the convention of the Microsoft C Compiler.

1.7 References to Documents

This document based on the following specification or documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX based products. Revision 12, English, 2012

[2] Hilscher Gesellschaft für Systemautomation mbH: CANopen Slave V3 Protocol API Manual. Revision 3, English, 2013

Table 3: References to Documents

1.8 Legal Notes

1.8.1 Copyright

1.8.2 Important Notes

The user manual, accompanying texts and the documentation were created for the use of the products by qualified experts, however, errors cannot be ruled out. For this reason, no guarantee can be made and neither juristic responsibility for erroneous information nor any liability can be assumed. Descriptions, accompanying texts and documentation included in the user manual do not present a guarantee nor any information about proper use as stipulated in the contract or a warranted feature. It cannot be ruled out that the user manual, the accompanying texts and the documentation do not correspond exactly to the described features, standards or other data of the delivered product. No warranty or guarantee regarding the correctness or accuracy of the information is assumed.

We reserve the right to change our products and their specification as well as related user manuals, accompanying texts and documentation at all times and without advance notice, without obligation to report the change. Changes will be included in future manuals and do not constitute any obligations. There is no entitlement to revisions of delivered documents. The manual delivered with the product applies.

Hilscher Gesellschaft für Systemautomation mbH is not liable under any circumstances for direct, indirect, incidental or follow-on damage or loss of earnings resulting from the use of the information contained in this publication.

1.8.3 Exclusion of Liability

The software was produced and tested with utmost care by Hilscher Gesellschaft für Systemautomation mbH and is made available as is. No warranty can be assumed for the performance and flawlessness of the software for all usage conditions and cases and for the results produced when utilized by the user. Liability for any damages that may result from the use of the hardware or software or related documents, is limited to cases of intent or grossly negligent violation of significant contractual obligations. Indemnity claims for the violation of significant contractual obligations are limited to damages that are foreseeable and typical for this type of contract.

It is strictly prohibited to use the software in the following areas:

- for military purposes or in weapon systems;
- for the design, construction, maintenance or operation of nuclear facilities;
- in air traffic control systems, air traffic or air traffic communication systems;
- in life support systems;
- in systems in which failures in the software could lead to personal injury or injuries leading to death.

We inform you that the software was not developed for use in dangerous environments requiring fail-proof control mechanisms. Use of the software in such an environment occurs at your own risk. No liability is assumed for damages or losses due to unauthorized use.

1.8.4 Export

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different counters, in particular those of Germany and the USA. The software may not be exported to countries where this is prohibited by the United States Export Administration Act and its additional provisions. You are obligated to comply with the regulations at your personal responsibility. We wish to inform you that you may require permission from state authorities to export, re-export or import the product.

2 Getting started / Configuration

2.1 Overview about Essential Functionality

You can find the most commonly used functionality of the CAN DL task API within the following sections of this document:

Topic	Section Number	Section Name
Configuration	CANopen (or DeviceNet) Slave Protocol API Manual V3	See according sections of the CANopen Slave Protocol API Manual V3 (not yet published) or DeviceNet Slave Protocol API Manual V3 (not yet published).
Register Application	4.1.5	`CAN_DL_CMD_AP_REGISTER_REQ/CNF` – Register Request/ Confirmation
Enable Rx ID	4.1.9	`CAN_DL_CMD_ENABLE_RXID_REQ/CNF` – Enable Identifier Request/ Confirmation
Receive Data	4.1.1	`CAN_DL_CMD_DATA_IND/RES` – Data Indication/Response
Transmit Data	4.1.2	`CAN_DL_CMD_DATA_REQ/CNF` – Data Request/Confirmation

Table 4: Overview about essential Functionality (Cyclic and acyclic Data Transfer and Alarm Handling).

2.2 Stack Structure

The CAN DL task is part of the CANopen stack as shown in the following figure:

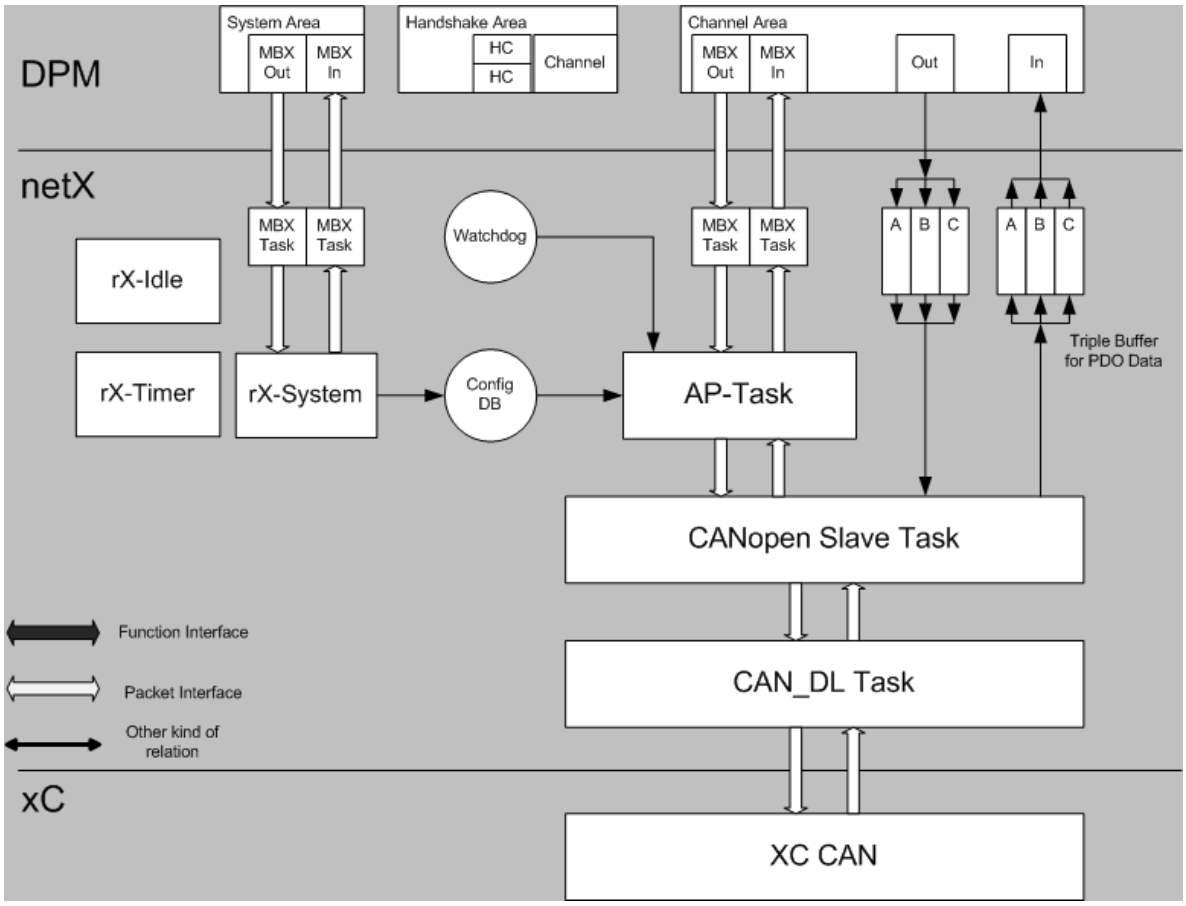

Figure 1: CANopen Stack Structure

2.3 Configuration

The CAN DL task is configured by the CANOPEN_APS_SET_CONFIGURATION_REQ packet of the CANopen task. See sections 4.2 and 5.1.3 of the CANopen Slave Protocol API Manual for more information.

Set bit 0 of the system flags to the value 1 indicating APPLICATION_CONTROLLED.

2.4 Sequence of Packets for Initialization

This section explains how to get started by a simple example:

Note: The example is based on use of a CANopen Slave running the Hilscher CANopen Slave Protocol Stack using the CAN DL task. However, the CAN DL task may also be used by future versions of the CANopen Master or DeviceNet Master or Slave protocol stacks.

The CANopen Slave is connected with any CANopen Master device. We assume this CANopen Master device sends data requests to the Hilscher CANopen Slave. The following example shows you how to prepare the Hilscher CANopen Slave to receive indications of these requests:

2.4.1 Set Configuration (Initialization of the Hilscher CANopen Slave Protocol Stack)

For initialization and configuration the CAN DL task does not provide an own packet. Therefore use the CANOPEN_APS_SET_CONFIGURATION_REQ packet (command code 0x2E04) of the CANopen APS task instead. This packet is described in the CANopen Slave Protocol API Manual (reference [2]).

- Set bit 0 of the system flags to the value 1 indicating APPLICATION_CONTROLLED.
- Deactivate the watchdog supervision by using the value 0.
- Set the node ID of the CANopen Slave to 2, for instance. Have in mind, the node ID may not be equal to the CANopen Master's ID!
- Adjust the baud rate ulBaudrate to the same value as used at the CANopen Master. The coding to use can be found in *Table 41: Codes and Corresponding Baud Rates of CANopen Network* of the CANopen Slave Protocol APU Manual. For example, if the master runs at a baudrate of 1 MBit/s set ulBaudrate to the value 0.
- Finally, set all other parameters not mentioned here to the value 0.

You should then receive the confirmation packet with the command code ulCmd equal to 0x2E05 (CANOPEN_SLAVE_SET_BUSPARAM_CNF) and the status code ulSta equal to 0. Check these for correctness.

2.4.2 Channel Init

Before continuing, it is necessary to perform a channel init. See section „4.9.3.3 Channel Initialization Request“ on page 139 of the Dual-Port Memory Interface Manual (reference #1) for a description of the *Channel Init* packet.

2.4.3 Registering at CAN-DL Task

When the CANopen stack with the integrated CAN DL task has been successfully configured, in order to receive indications it is necessary to register at the CAN DL task next. This is done using the CAN_DL_CMD_AP_REGISTER_REQ/CNF – Register Request/ Confirmation packet described in subsection 4.1.5 on page 31 of this document.

Set variable ulInitMode of the packet to the value 1 (CAN_DL_REGISTER_MODE_REGISTER_11BIT_ONLY) in order to register for the „11 bit only“ mode.

Note: If you register at the CAN DL task, you enable indications to be received. As long as the registration is valid, it is necessary to process these incoming indications in order to avoid buffer overruns and loss of data.

You should then receive the confirmation packet with the command code ulCmd equal to 0x2A0B (CAN_DL_CMD_AP_REGISTER_CNF) and the status code ulSta equal to 0. Check these for correctness.

2.4.4 Enable CAN Identifier for Reception

After registering at the CAN DL task, it is necessary to enable the CAN identifier for being received. In order to do so, use the packet CAN_DL_CMD_ENABLE_RXID_REQ/CNF – Enable Identifier Request/ Confirmation described in subsection 4.1.9 on page 46 of this document.

- Set the bits of the bit mask variable ulControlBit as follows:
  - Bit 0: This is the enable/disable bit. Set it to 1 (enable)!
  - Bit 1: This is the range start bit. In order to define a single CAN identifier, set this bit to 0.
  - Bit 2: This is the return UUID bit. Set it to 1 in order to make use of the UUID feature easing the identification of the packets by adding an additional identification.
- In order to specify the CAN identifier, set variable ulIdentifier of the packet to the correct value. For instance, for reading the output data of the master, the correct address would be 514 (=0x202), if the slave address is 2.
- Set variable ulUniqueID of the packet to the value „1234“, for instance. This value might be helpful in identying responses .

You should then receive the confirmation packet with the command code ulCmd equal to 0x2A13 (CAN_DL_CMD_ENABLE_RXID_CNF) and the status code ulSta equal to 0. Check these for correctness.

2.4.5 Receive Data Indication

If data requests coming from the CANopen Master arrive at the CANopen slave, they will cause data indications (packet `CAN_DL_CMD_DATA_IND/RES` – Data Indication/Response) to be received. These are described in subsection 4.1.1 on page 18 of this document.

If you correctly received a CAN_DL_CMD_DATA_IND packet, you will receive a CAN Receive Frame as data part of this packet, see *Table 10: CAN Receive Frame* on page 18.

The parameters of the CAN Receive Frame should have the following values:

- Variable `ulUniqueID` of the CAN_DL_CMD_DATA_IND packet should have the value „`1234`“ in this example.
- Variable `ulFrameInfo` of the CAN_DL_CMD_DATA_IND packet should contain a bit mask.
- Variable `ulIdentifier` of the CAN_DL_CMD_DATA_IND packet should contain the value of the identifier, in the example above this would be `514 (=0x202)`.
- Variable `abData[]` of the CAN_DL_CMD_DATA_IND packet should contain the data sent from the CANopen Master (up to 8 bytes depending on the settings of the CANopen Master).

Note: If for a longer time more CAN telegrams arrive than are processed, this will cause a buffer overrun after either hardware (xC) or software buffers (stack) have run full. Loss of data will occur then. So take care of processing received indications immediately by sending the `CAN_DL_CMD_DATA_RES` response packet, see *Table 13: CAN_DL_PACKET_DATA_RES_T – Response to Data Indication*.

2.4.6 Overview Diagram

The following figure shows a diagram illustrating the timely sequence of requests, confirmations and indications of the process described above (time axis from top to bottom):

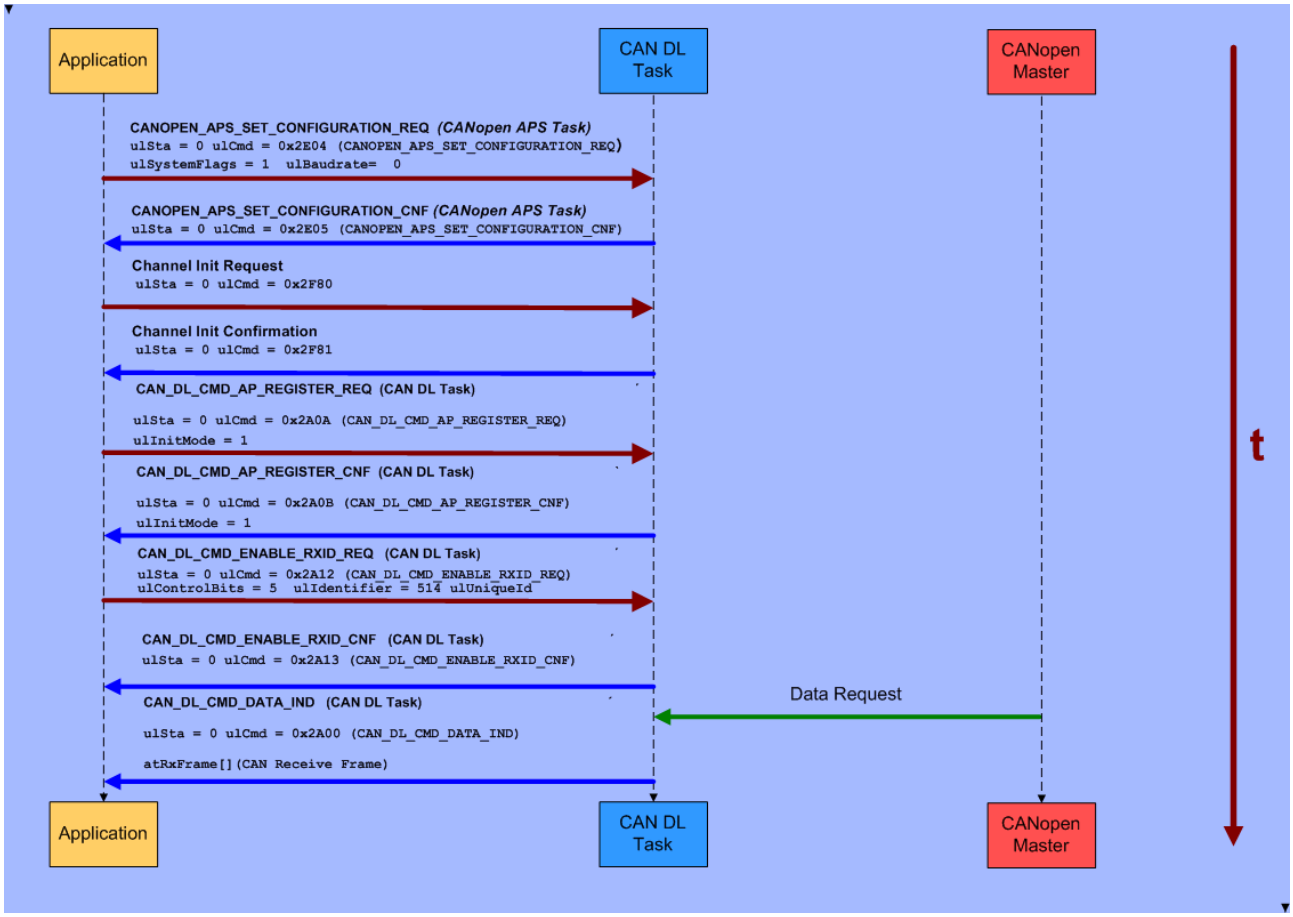

Figure 2: CAN DL Sequence of Packets

3 Diagnosis

The following diagnostic data are available from the CAN DL task by the CAN_DL_CMD_DIAG_REQ/CNF – Diagnosis Request/ Confirmation packet.

The kind of data to be delivered can be selected by setting parameter ulDiagType of the request packet to the values 0, 1 or 2.

1. Setting ulDiagType to 0 will cause a complete diagnostic report to be returned in the confirmation packet. This report contains a lot of counters:

Variable	Type	Explanation
ulCmdDataReq	UINT32	Counter for data requests (CAN_DL_CMD_DATA_REQ/CNF – Data Request/Confirmation)
ulCmdDataCnfPos	UINT32	Counter for positive data confirmations
ulCmdDataCnfNeg	UINT32	Counter for negative data confirmations
ulCanDlDataInd	UINT32	Counter for CAN DL data indications (CAN_DL_CMD_DATA_IND/RES – Data Indication/Response)
ulCanDlDataRes	UINT32	Counter for CAN DL data responses
ulCmdStartReq	UINT32	Counter for start requests (CAN_DL_CMD_START_REQ/CNF –Start Request/ Confirmation)
ulCmdStartCnfPos	UINT32	Counter for positive start confirmations
ulCmdStartCnfNeg	UINT32	Counter for negative start confirmations
ulCmdStopReq	UINT32	Counter for stop requests (CAN_DL_CMD_STOP_REQ/CNF – Stop Request/ Confirmation)
ulCmdStopCnfPos	UINT32	Counter for positive stop confirmations
ulCmdStopCnfNeg	UINT32	Counter for negative stop confirmations
ulCmdApRegReq	UINT32	Counter for application register requests (CAN_DL_CMD_AP_REGISTER_REQ/CNF – Register Request/ Confirmation)
ulCmdApRegCnfPos	UINT32	Counter for positive application register confirmations
ulCmdApRegCnfNeg	UINT32	Counter for negative application register confirmations
ulCmdSetPrmReq	UINT32	Counter for set parameter requests (CAN_DL_CMD_SET_PRM_REQ/CNF – Set Parameter Request/ Confirmation)
ulCmdSetPrmCnfPos	UINT32	Counter for positive set parameter confirmations
ulCmdSetPrmCnfNeg	UINT32	Counter for negative set parameter confirmations
ulCmdSetFilterReq	UINT32	Counter for set filter requests (CAN_DL_CMD_SET_FILTER_REQ/CNF – Set Filter Request/ Confirmation)
ulCmdSetFilterCnfPos	UINT32	Counter for positive set filter confirmations
ulCmdSetFilterCnfNeg	UINT32	Counter for negative set filter confirmations
ulCmdEnRxIdReq	UINT32	Counter for enable rx requests (CAN_DL_CMD_ENABLE_RXID_REQ/CNF – Enable Identifier

		Request/ Confirmation)
ulCmdEnRxIdCnfPos	UINT32	Counter for positive enable rx confirmations
ulCmdEnRxIdCnfNeg	UINT32	Counter for negative enable rx confirmations
ulCmdEveInd	UINT32	Counter for event indications (CAN_DL_CMD_EVENT_IND/RES – Event Indication/Response)
ulCmdEveRes	UINT32	Counter for event responses
ulCmdEveAckReq	UINT32	Counter for event acknowledge requests (CAN_DL_CMD_EVENT_ACK_REQ/CNF – Event Acknowledge Request/ Confirmation)
ulCmdEveAckCnfPos	UINT32	Counter for positive event acknowledge confirmations
ulCmdEveAckCnfNeg	UINT32	Counter for negative event acknowledge confirmations
ulCmdTxAbortReq	UINT32	Counter for transmission abort requests (CAN_DL_CMD_TX_ABORT_REQ/CNF – Transmission Abort)
ulCmdTxAbortCnfPos	UINT32	Counter for positive transmission abort confirmations
ulCmdTxAbortCnfNeg	UINT32	Counter for negative transmission abort confirmations
ulCmdInitReq	UINT32	Counter for init requests
ulCmdInitCnfPos	UINT32	Counter for positive init confirmations
ulCmdInitCnfNeg	UINT32	Counter for negative init confirmations
ulCmdDataHiReq	UINT32	Counter for high priority data requests (not supported)
ulCmdDataHiCnfPos	UINT32	Counter for positive high priority data confirmations
ulCmdDataHiCnfNeg	UINT32	Counter for negative high priority data confirmations
ulApUnknown	UINT32	Counter for unknown packets

Table 5: Diagnostic Report

2. Setting ulDiagType to CAN_DL_DIAG_TYPE_CMD2 = 1 will return the command statistics in the confirmation packet. The command statistics is structured as follows:

Variable	Type	Explanation
ulCyclicEvent	UINT32	Cyclic Event
ulDrvCanRxInd	UINT32	Driver CAN Receive Indication
ulDrvCanEveInd	UINT32	Driver CAN Event Indication
ulCmdDiagReq	UINT32	Command Diag Request
ulCmdDiagCnfPos	UINT32	Command Diag Confirmation Positive
ulCmdDiagCnfNeg	UINT32	Command Diag Confirmation Negative

Table 6: Command Statistics

3. Setting ulDiagType to CAN_DL_DIAG_TYPE_STATUS = 2 will return the CAN Status List in the confirmation packet.

Variable	Type	Explanation
ulStatus	UINT32	CAN Status
ulTxFrameSucceed	UINT32	Transmission Frame Succeed

ulTxErrorSummary	UINT32	Transmission Error Summary
ulRxFrameSucceed	UINT32	Receive Frame Succeed
ulRxErrorSummary	UINT32	Receive Error Summary
ulTxErrCnt	UINT32	Transmission Error Count
ulRxErrCnt	UINT32	Receive Error Count
ulArbitrationLost	UINT32	Arbitration Lost
ulIndDroppedDueFifoFull	UINT32	Indication dropped due FIFO Full
ulConDroppedDueFifoFull	UINT32	Confirmation dropped due FIFO Full
ulRxStdFramesFilterd	UINT32	Receive Standard Frames filtered
ulRxExtFramesFilterd	UINT32	Receive Extended Frames filtered
ulRxStdFramesPassed	UINT32	Receive Standard Frames passed
ulRxExtFramesPassed	UINT32	Receive Extended Frames passed

Table 7: CAN Status List

4 The Application Interface

The following describes the packet API of the CAN DL task. There is also a function API available for which there is a separate description (not yet published). If you want to use the function API, you need to get two handles using the CAN_DL_CMD_GET_HANDLE_REQ/CNF – Get Handle packet described on page 61 of this document.

4.1 The CAN-DL-Task

The CAN DL task is the interface between dual port memory and the overlaying stack (CANopen Master or Slave or DeviceNet Master or Slave). All services should send to this task. For addressing a packet to the CAN DL task the destination address 0x20 is used.

To get the handle of the process queue of the CAN DL task the Macro TLR_QUE_IDENTIFY() needs to be used.

ASCII queue name	Description
"CAN_DL_QUE"	Name of the CAN DL task process queue

Table 8: CAN DL Task Process Queue

The returned handle has to be used as value ulDest in all request packets to be sent to the CAN DL task. This handle is the same handle that has to be used in conjunction with the macros like TLR_QUE_SENDPACKET_FIFO/LIFO() for sending a packet to the CAN DL task.

In detail, the following functionality is provided by the CAN DL task:

Overview over the Packets of the CAN DL Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
4.1.1	CAN_DL_CMD_DATA_IND/RES – Data Indication/Response	0x2A00/ 0x2A01	18
4.1.2	CAN_DL_CMD_DATA_REQ/CNF – Data Request/Confirmation	0x2A02/ 0x2A03	22
4.1.3	CAN_DL_CMD_START_REQ/CNF –Start Request/ Confirmation	0x2A06/ 0x2A07	27
4.1.4	CAN_DL_CMD_STOP_REQ/CNF – Stop Request/ Confirmation	0x2A08/ 0x2A09	29
4.1.5	CAN_DL_CMD_AP_REGISTER_REQ/CNF – Register Request/ Confirmation	0x2A0A/ 0x2A0B	31
4.1.6	CAN_DL_CMD_SET_FILTER_REQ/CNF – Set Filter Request/ Confirmation	0x2A0C/ 0x2A0D	34
4.1.7	CAN_DL_CMD_SET_PRM_REQ/CNF – Set Parameter Request/ Confirmation	0x2A0E/ 0x2A0F	37
4.1.8	CAN_DL_CMD_EVENT_IND/RES – Event Indication/Response	0x2A10/ 0x2A11	43
4.1.9	CAN_DL_CMD_ENABLE_RXID_REQ/CNF – Enable Identifier Request/ Confirmation	0x2A12/ 0x2A13	46
4.1.10	CAN_DL_CMD_EVENT_ACK_REQ/CNF – Event Acknowledge Request/ Confirmation	0x2A14/ 0x2A15	49
4.1.11	CAN_DL_CMD_DIAG_REQ/CNF – Diagnosis Request/ Confirmation	0x2A16/ 0x2A17	52
4.1.12	CAN_DL_CMD_TX_ABORT_REQ/CNF – Transmission Abort	0x2A1A/ 0x2A1B	56
4.1.13	CAN_DL_CMD_AUTO_BAUD_IND/RES – Auto Baud Detection Complete Indication	0x2AE8/ 0x2AE9	58
4.1.14	CAN_DL_CMD_GET_HANDLE_REQ/CNF – Get Handle	0x2AEA/ 0x2AEB	61
4.1.15	CAN_DL_CMD_SET_AUTOBAUD_FLAGS_REQ/CNF – Set Autobaud Flags	0x2AEC/ 0x2AED	63
4.1.16	CAN_DL_CMD_SET_EVENTS_TO_INDICATE_REQ/CNF – Register Events	0x2AEE/ 0x2AEF	66

Table 9: Overview over the Packets of the CAN DL Task

4.1.1 CAN_DL_CMD_DATA_IND/RES – Data Indication/Response

The CAN DL Task indicates the reception of data telegrams from the CAN network to the Fieldbus Task. The CAN DL Task will indicate data telegrams only if the sending CAN node is included within the filter of configured CAN identifiers (concerning filtering see section “CAN_DL_CMD_SET_FILTER_REQ/CNF – Set Filter Request/ Confirmation “ on page 35).

The packet may contain one or more data telegrams and the related CAN identifier of the sending node.

The CAN receive frame is structured as follows:

Variable	Type	Range	Explanation
ulUniqueId	UINT32	$0..2^{32}-1$	Unique value that was specified in the registration of ID’s. (See CAN_DL_CMD_ENABLE_RXID_REQ) Not sent to wire!
ulFrameInfo	UINT32	Bit mask, see below	Frame info
ulIdentifier	UINT32	$0..2^{11}-1$ (11-bit) $0..2^{29}-1$ (29-bit)	The 11-bit or 29-bit CAN identifier of the frame
abData[]	UINT8[]	Array	Data field (0 up to 8 bytes)

Table 10: CAN Receive Frame

The frame info in the CAN Receive Frame looks like:

Variable	Value	Bits	Explanation
MSK_CAN_DL_CAN_FRAME_INFO_DLC	0x0000000F	0-3	Data length code
MSK_CAN_DL_CAN_FRAME_INFO_RTR	0x00000010	4	Remote transmission request
MSK_CAN_DL_CAN_FRAME_INFO_CNF_REQUESTED	0x00000020	5	Confirmation request
MSK_CAN_DL_CAN_FRAME_INFO_SINGLE_SHOT	0x00000040	6	Send frame as single shot
MSK_CAN_DL_CAN_FRAME_INFO_RES1	0x7fffff80	7-30	Reserved bits
MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT	0x80000000	31	29 Bit frame format

Table 11: Contents of Frame Info

ulIdentifier of the CAN receive frame should be in the range $0..2^{11}-1$ if MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT equals 0 and in the range $0..2^{29}-1$ otherwise.

The packet is unsolicited from the viewpoint of the Fieldbus Task.

Packet Structure Reference

```
/* Data indication */

#define CAN_DL_MAX_ENABLE_RXID_PER_PACKET       (8)
#define CAN_DL_MAX_BYTE_PER_FRAME               (8)

/* Masks CMD data request */
#define MSK_CAN_DL_CAN_FRAME_INFO_DLC            0x0000000fU /* Data length code */
#define MSK_CAN_DL_CAN_FRAME_INFO_RTR            0x00000010U /* Remote transmission
request */
#define MSK_CAN_DL_CAN_FRAME_INFO_CNF_REQUESTED  0x00000020U /* Confirmation request */
#define MSK_CAN_DL_CAN_FRAME_INFO_SINGLE_SHOT    0x00000040U /* Send frame as single
shot*/
#define MSK_CAN_DL_CAN_FRAME_INFO_RES1           0x7fffff80U /* Reserved bits*/
#define MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT   0x80000000U /* 29 Bit frame format*/

/* CAN receive frame */
typedef struct CAN_DL_CAN_RX_FRAME_Ttag {
  TLR_UINT32  ulUniqueId;
  TLR_UINT32  ulFrameInfo;
     TLR_UINT32  ulIdentifier;
     TLR_UINT8   abData[CAN_DL_MAX_BYTE_PER_FRAME];
} CAN_DL_CAN_RX_FRAME_T;

typedef struct CAN_DL_SDU_DATA_IND_Ttag {
  CAN_DL_CAN_RX_FRAME_T atRxFrame[CAN_DL_MAX_ENABLE_RXID_PER_PACKET];
} CAN_DL_SDU_DATA_IND_T;

typedef struct  CAN_DL_PACKET_DATA_IND_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_DATA_IND_T         tData;        /* Packet Data Unit */
} CAN_DL_PACKET_DATA_IND_T;
```

Packet Description

Structure Information CAN_DL_PACKET_DATA_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	160	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A00	CAN_DL_CMD_DATA_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_DATA_IND_T			
atRxFrame[]	CAN_DL_CAN_RX[CAN_DL_MAX_ENABLE_RXID_PER_PACKET]		CAN receive frame as described in *Table 10: CAN Receive Frame*.

Table 12: `CAN_DL_PACKET_DATA_IND_T` – Data Indication

Packet Structure Reference

```
/* Data response */
typedef struct  CAN_DL_PACKET_DATA_RES_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_DATA_RES_T;
```

Packet Description

Structure Information CAN_DL_PACKET_DATA_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A01	CAN_DL_CMD_DATA_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 13: CAN_DL_PACKET_DATA_RES_T – Response to Data Indication

4.1.2 CAN_DL_CMD_DATA_REQ/CNF – Data Request/Confirmation

The Fieldbus Task uses the Send Data Packet in order to pass data telegrams and CAN identifiers to the CAN DL Task. The packet may contain one or more data telegrams and the related CAN identifier of the receiving node.

The CAN transmit frame is structured as follows:

Variable	Type	Range	Explanation
ulUniqueId	UINT32	$0..2^{32}$-1	Unique packet identifier. Not sent to wire
ulFrameInfo	UINT32	Bit mask, see below	Frame info
ulIdentifier	UINT32	$0..2^{11}$-1 (11-bit) $0..2^{29}$-1 (29-bit)	The 11-bit or 29-bit CAN identifier of the frame
abData[]	UINT8[]	Array	Data field (0 up to 8 bytes)

Table 14: CAN Transmit Frame

The frame info looks like:

Variable	Value	Bits	Explanation
MSK_CAN_DL_CAN_FRAME_INFO_DLC	0x0000000F	0-3	Data length code
MSK_CAN_DL_CAN_FRAME_INFO_RTR	0x00000010	4	Remote transmission request
MSK_CAN_DL_CAN_FRAME_INFO_CNF_REQUESTED	0x00000020	5	Confirmation request
MSK_CAN_DL_CAN_FRAME_INFO_SINGLE_SHOT	0x00000040	6	Send frame as single shot
MSK_CAN_DL_CAN_FRAME_INFO_RES1	0x7fffff80	7-30	Reserved bits
MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT	0x80000000	31	29 Bit frame format

Table 15: Contents of Frame Info

`ulIdentifier` of the CAN transmit frame should be in the range $0..2^{11}$-1 if MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT equals 0 and in the range $0..2^{29}$-1 otherwise.

Packet Structure Reference

```
/* Data request */

#define CAN_DL_MAX_FRAME_PER_PACKET             (16)

/* Masks CMD data request */
#define MSK_CAN_DL_CAN_FRAME_INFO_DLC            0x0000000fU /* Data length code       */
#define MSK_CAN_DL_CAN_FRAME_INFO_RTR            0x00000010U /* Remote transmission
request */
#define MSK_CAN_DL_CAN_FRAME_INFO_CNF_REQUESTED  0x00000020U /* Confirmation request   */
#define MSK_CAN_DL_CAN_FRAME_INFO_SINGLE_SHOT    0x00000040U /* Send frame as singe shot
*/
#define MSK_CAN_DL_CAN_FRAME_INFO_RES1           0x7fffff80U /* Reserved bits
*/
#define MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT   0x80000000U /* 29 Bit frame format
*/

/* CAN transmit frame */
typedef struct CAN_DL_CAN_TX_FRAME_Ttag {
  TLR_UINT32  ulUniqueId;
  TLR_UINT32  ulFrameInfo;
     TLR_UINT32  ulIdentifier;
     TLR_UINT8   abData[];
} CAN_DL_CAN_TX_FRAME_T;

typedef struct CAN_DL_SDU_DATA_REQ_Ttag {
  CAN_DL_CAN_TX_FRAME_T atTxFrame[CAN_DL_MAX_FRAME_PER_PACKET];
} CAN_DL_SDU_DATA_REQ_T;

typedef struct CAN_DL_PACKET_DATA_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_DATA_REQ_T         tData;        /* Packet Data Unit */
} CAN_DL_PACKET_DATA_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_DATA_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	320	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A02	CAN_DL_CMD_DATA_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_DATA_REQ_T			
atTxFrame[]	CAN_DL_CAN_TX_FRAME_T[CAN_DL_MAX_FRAME_PER_PACKET]		CAN transmit frame as described in *Table 14: CAN Transmit Frame*

Table 16: CAN_DL_PACKET_DATA_REQ_T – Data Request

Packet Structure Reference

```
/* Data confirmation */

#define CAN_DL_MAX_FRAME_PER_PACKET             (16)

/* Masks CMD data request */
#define MSK_CAN_DL_CAN_FRAME_INFO_DLC            0x0000000fU /* Data length code
*/
#define MSK_CAN_DL_CAN_FRAME_INFO_RTR            0x00000010U /* Remote transmission
request */
#define MSK_CAN_DL_CAN_FRAME_INFO_CNF_REQUESTED  0x00000020U /* Confirmation request
*/
#define MSK_CAN_DL_CAN_FRAME_INFO_SINGLE_SHOT    0x00000040U /* Send frame as singe shot
*/
#define MSK_CAN_DL_CAN_FRAME_INFO_RES1           0x7fffff80U /* Reserved bits
*/
#define MSK_CAN_DL_CAN_FRAME_INFO_FRAME_FORMAT   0x80000000U /* 29 Bit frame format
*/

/* CAN transmit frame */
typedef struct CAN_DL_CAN_TX_FRAME_Ttag {
  TLR_UINT32  ulUniqueId;
  TLR_UINT32  ulFrameInfo;
     TLR_UINT32  ulIdentifier;
     TLR_UINT8   abData[];
} CAN_DL_CAN_TX_FRAME_T;

typedef struct CAN_DL_SDU_DATA_CNF_Ttag {
  CAN_DL_CAN_TX_FRAME_T atTxFrame[CAN_DL_MAX_FRAME_PER_PACKET];
} CAN_DL_SDU_DATA_CNF_T;

typedef struct CAN_DL_PACKET_DATA_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_DATA_CNF_T         tData;        /* Packet Data Unit */
} CAN_DL_PACKET_DATA_CNF_T;
```

Packet Description

Structure Information CAN_DL_PACKET_DATA_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	320	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A03	CAN_DL_CMD_DATA_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_DATA_CNF_T			
atTxFrame[]	CAN_DL_CAN_TX_FRAME_T[CAN_DL_MAX_FRAME_PER_PACKET]		CAN transmit frame as described in *Table 14: CAN Transmit Frame*

Table 17: CAN_DL_PACKET_DATA_CNF_T –Confirmation to Data Request

4.1.3 CAN_DL_CMD_START_REQ/CNF –Start Request/ Confirmation

The Fieldbus protocol stack sends this request packet to start the CAN controller after successful configuration of the CAN DL task. After processing of this command, receiver and transceiver interrupt are enabled.

Possible error situations which might occur are:

- Configuration locked (Error code: TLR_E_CAN_DL_CONF_LOCKED)
- Invalid packet length (Error code: TLR_E_CAN_DL_CMD_LENGTH_MISMATCH)

Packet Structure Reference

```
/* Start request */
typedef struct CAN_DL_PACKET_START_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_START_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_START_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A06	CAN_DL_CMD_START_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 18: CAN_DL_PACKET_START_REQ_T – Start Request

Packet Structure Reference

```
/* Start confirmation */
typedef struct CAN_DL_PACKET_START_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_START_CNF_T;
```

Packet Description

Structure Information CAN_DL_PACKET_START_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A07	CAN_DL_CMD_START_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 19: CAN_DL_PACKET_START_CNF_T – Confirmation to Start Request

4.1.4 CAN_DL_CMD_STOP_REQ/CNF – Stop Request/ Confirmation

This packet stops the CAN controller.

Note: All filter settings are removed when stopping the communication.

Possible error situations which might occur are:

- Configuration locked (Error code: TLR_E_CAN_DL_CONF_LOCKED)
- Invalid packet length (Error code: TLR_E_CAN_DL_CMD_LENGTH_MISMATCH)

Packet Structure Reference

```
/* Stop request */
typedef struct CAN_DL_PACKET_STOP_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_STOP_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_STOP_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A08	CAN_DL_CMD_STOP_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 20: CAN_DL_PACKET_STOP_REQ_T – Stop Request

Packet Structure Reference

```
/* Stop Confirmation
typedef struct CAN_DL_PACKET_STOP_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_STOP_CNF_T;
```

Packet Description

Structure Information CAN_DL_PACKET_STOP_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A09	CAN_DL_CMD_STOP_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 21: CAN_DL_PACKET_STOP_CNF_T – Confirmation to Stop Request

4.1.5 CAN_DL_CMD_AP_REGISTER_REQ/CNF – Register Request/ Confirmation

This packet is used in order to register (or unregister using `ulInitMode = 2`) to the CAN-DL task. Every application, that uses the CAN DL task, has to perform this command. Also when an application intends only sending of frames, it has to be registered at the CAN DL task.

After registration is performed successfully, indication packets are sent from the CAN-DL task to the source queue of this request packet.

Note: Up to five applications can be registered on CAN DL task for events and sending/receiving frames with certain identifiers.

The behavior of this packet depends on the choice of the parameter `ulInitMode`. Actually, three modes are available for registering:

Variable `ulInitMode`		
Option	**Value**	**Explanation**
`CAN_DL_REGISTER_MODE_REGISTER`	0	Register with extended frames enabled
`CAN_DL_REGISTER_MODE_REGISTER_11BIT_ONLY`	1	Register with extended frames disabled
`CAN_DL_REGISTER_MODE_UNREGISTER`	2	Unregister application

Table 22: Possible Values of Parameter `ulInitMode`

`ulInitMode = 0` will register the application in a combined 11 and 29 bit mode. This mode opens extended (29 bit) frames automatically. All extended frames pass through if they match the chosen filter settings.

`ulInitMode = 1` will register the application in 11 bit only mode. Only 11 bit standard frames pass through. Extended frames (with 29 bit identifiers) will be prohibited.

`ulInitMode = 2` will unregister the application as stated above.

Possible error situations which might occur are:

- Invalid length (Error code: `TLR_E_CAN_DL_CMD_LENGTH_MISMATCH`)
- Not enough memory (Error code: `TLR_E_OUTOFMEMORY`)
- This application is already registered (Error code: `TLR_E_CAN_DL_AP_ALREADY_REGISTERED`)
- Unknown application (Error code: `TRL_E_CAN_DL_UNKNOWN_APPLICATION`)

Packet Structure Reference

```
typedef struct CAN_DL_SDU_AP_REGISTER_REQ_Ttag {
  TLR_UINT32              ulInitMode;
} CAN_DL_SDU_AP_REGISTER_REQ_T;

typedef struct CAN_DL_PACKET_AP_REGISTER_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_AP_REGISTER_REQ_T  tData;        /* Packet data      */
} CAN_DL_PACKET_AP_REGISTER_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_AP_REGISTER_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A0A	CAN_DL_CMD_AP_REGISTER_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_AP_REGISTER_REQ_T			
ulInitMode	UINT32	0-2	Register request mode

Table 23: CAN_DL_PACKET_AP_REGISTER_REQ_T – Register Request

Packet Structure Reference

```
/* Init confirmation */

typedef struct CAN_DL_SDU_AP_REGISTER_CNF_Ttag {
  TLR_UINT32              ulInitMode;
} CAN_DL_SDU_AP_REGISTER_CNF_T;

typedef struct CAN_DL_PACKET_AP_REGISTER_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_AP_REGISTER_CNF_T  tData;        /* Packet data      */
} CAN_DL_PACKET_AP_REGISTER_CNF_T; };
```

Packet Description

Structure Information CAN_DL_PACKET_AP_REGISTER_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A0B	CAN_DL_CMD_AP_REGISTER_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_AP_REGISTER_CNF_T			
ulInitMode	UINT32	0-2	Register request mode

Table 24: CAN_DL_PACKET_AP_REGISTER_CNF_T – Confirmation to Register Request

4.1.6 CAN_DL_CMD_SET_FILTER_REQ/CNF – Set Filter Request/ Confirmation

This request packet can be used by the user application to set filters determining whether received CAN messages will be routed to the user application based on their identifier, or not.

If a filter has been set, only those packets whose identifiers pass through this filter (see description of the filtering process below) will be routed to the application.

Each acceptance filter is defined by a **mask** parameter `ulMask` and a **code** parameter `ulCode`. The mask parameter defines, for each bit of the identifier, whether the filtering process cares about this bit or not. A 0 means ”don’t care” and a 1 means ”do care.” The code parameter then defines, for each bit of the identifier, that the filtering process cares about (defined by the mask parameter), what the bit value has to be (0 or 1).

Standard identifier acceptance mask/code segmentation: `DataByte2[8], DataByte1[8], DLC[4], StdId[11], RTR[1]`

Extended identifier acceptance mask/code segmentation: `UNUSED[2], ExtId[29], RTR[1]`

`ulFilterInstance` may have the following values:

Variable `ulFilterInstance`		
Option	**Value**	**Explanation**
`CAN_DL_FILTER_INSTANCE_11BIT`	0	Filtering is done with 11 bit identifiers
`CAN_DL_FILTER_INSTANCE_29BIT`	1	Filtering is done with 29 bit identifiers

Table 25: Possible Values of `ulFilterInstance`

The structure `CAN_DL_ACCEPTANCE_FILTER_T` contains the following information about the filter:

Structure `CAN_DL_ACCEPTANCE_FILTER_T`		
Variable	**Type**	**Explanation**
`ulInstance`	UINT32	Must be 0 or 1. Indicates whether the specified filter is to be used with standard identifiers (11 bit, value equals 0) or extended identifiers (29 bit, value equals 1)
`ulCode`	UINT32	Acceptance code
`ulMask`	UINT32	Acceptance mask

Table 26: Structure `CAN_DL_ACCEPTANCE_FILTER_T`

The acceptance filtering process is performed bit-wise according to the following step-by-step description:

1. The bit of the instance to be checked is compared with the corresponding bit of the acceptance code. In case of equality the value 1 will result, otherwise 0.
2. The result is OR-related with the acceptance mask. If the result is 1 the bit is accepted, otherwise it is rejected.
3. Only in case all bits are accepted (i.e. an AND operation of all single bits) , the message will be routed to the user application.

Possible error situations which might occur are:

- Configuration locked (Error code: TLR_E_CAN_DL_CONF_LOCKED)
- Error while setting the filter (Error code: TLR_E_CAN_DL_SET_FILTER_FAILED)
- Invalid packet length (Error code: TLR_E_CAN_DL_CMD_LENGTH_MISMATCH)

Packet Structure Reference

```
/* Set filter request */

typedef struct CAN_DL_ACCEPTANCE_FILTER_Ttag {
  TLR_UINT32  ulInstance;
  TLR_UINT32  ulCode;
  TLR_UINT32  ulMask;
} CAN_DL_ACCEPTANCE_FILTER_T;

typedef struct CAN_DL_SDU_SET_FILTER_REQ_Ttag {
  TLR_UINT32                 ulFilterInstance;
  CAN_DL_ACCEPTANCE_FILTER_T  tAcceptanceFilter;
} CAN_DL_SDU_SET_FILTER_REQ_T;

typedef struct CAN_DL_PACKET_SET_FILTER_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_SET_FILTER_REQ_T   tData;        /* Packet Data Unit */
} CAN_DL_PACKET_SET_FILTER_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_SET_FILTER_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	16	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A0C	CAN_DL_CMD_SET_FILTER_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_SET_FILTER_REQ_T			
ulFilterInstance	UINT32	0,1	Filter instance indicating whether 11 bit or 29 bit data are used
tAcceptanceFilter	CAN_DL_ACCEPTANCE_FILTER_T		Acceptance filter

Table 27: CAN_DL_PACKET_SET_FILTER_REQ_T – Set Filter Request

Packet Structure Reference

```
/* Set filter confirmation */

typedef struct CAN_DL_PACKET_SET_FILTER_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_SET_FILTER_CNF_T;
```

Packet Description

Structure Information CAN_DL_PACKET_SET_FILTER_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A0D	CAN_DL_CMD_SET_FILTER_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 28: CAN_DL_PACKET_SET_FILTER_CNF_T – Confirmation to Set Filter Request

4.1.7 CAN_DL_CMD_SET_PRM_REQ/CNF – Set Parameter Request/ Confirmation

This packet contains information regarding the configuration of the xMAC/xPEC controller of the netX and is sent from the Fieldbus Task to the CAN DL Task. It is used to configure the CAN network as well as interrupts of the CAN DL Task, if necessary.

The packet may be used both to configure a whole set of parameters at once or a single parameter. This depends on the choice of variable ulPrmType which allows parameters of different types to be set. It is also possible to lock or unlock the configuration using this packet.

The following parameter types are available:

Variable ulPrmType - Parameter types for Set Parameter Command		
Option	**Value**	**Explanation**
CAN_DL_PRM_TYPE_PARMETER_SET	0	Parameter set
CAN_DL_PRM_TYPE_MODE	1	Mode
CAN_DL_PRM_TYPE_BAUDRATE	2	Baud rate
CAN_DL_PRM_TYPE_TXABORT_TIME	3	Transmission abort time
CAN_DL_PRM_TYPE_EVENTS_TO_INDICATE	4	Events to indicate
CAN_DL_PRM_TYPE_FILTER	5	Filter
CAN_DL_PRM_TYPE_CONF_LOCK	6	Configuration lock
CAN_DL_PRM_TYPE_CONF_UNLOCK	7	Configuration unlock

Table 29: Parameter types for Set Parameter Command

The parameter (or parameter structure) itself is then specified using the union unPrm of type CAN_DL_PARAMETER_T.

CAN_DL_PARAMETER_T looks as follows:

Union CAN_DL_PARAMETER_T			
Variable	**Type**	**Explanation**	**For use with Parameter Type**
tSet	CAN_DL_PARAMETER_SET_T	Parameter set	0
ulMode	UINT32	Mode	1
ulBaudrate	UINT32	Baud Rate	2
ulTxAbortTime	UINT32	Transmission abort time	3
ulEventsIndicated	UINT32	Events indicated	4
tFilter	CAN_DL_ACCEPTANCE_FILTER_T	Filter	5

Table 30: Structure CAN_DL_PARAMETER_T

Parameter **Parameter Set** (ulPrmType= CAN_DL_PRM_TYPE_PARMETER_SET = 0)

This option can be used for specifying a whole set of parameters entirely at once. This set of parameters is stored in a structure of type CAN_DL_PARAMETER_SET_T. It looks as follows:

Variable	Type	Explanation
ulMode	UINT32	Mode
ulBaudrate	UINT32	Baud Rate
ulTxAbortTime	UINT32	Transmission abort time
ulEventsIndicated	UINT32	Events indicated
atFilter[2]	CAN_DL_ACCEPTANCE_FILTER_T	Up to two acceptance filters can be defined here
ulNumRxIdEnable	UINT32	Number of enabled receive identifiers
ulFirstRxId	UINT32	First receive identifier

Table 31: Structure CAN_DL_PARAMETER_SET_T

Parameter **Mode** (`ulPrmType= CAN_DL_PRM_TYPE_MODE=1`)

This option can be used for specifying only a new mode:

Possible Values of Parameter Mode		
Option	**Value**	**Explanation**
`CAN_DL_MODE_RXTX`	0x00000000	Receive / transmit
`CAN_DL_MODE_LISTEN_ONLY`	0x00000001	Listen only (no transmit)
`CAN_DL_MODE_IO_0_MONITORING`	0x00010000	IO 0 monitoring
`CAN_DL_MODE_IO_1_MONITORING`	0x00020000	IO 1 monitoring
`CAN_DL_MODE_IO_0_INVERTED`	0x00040000	IO 0 inverted
`CAN_DL_MODE_IO_1_INVERTED`	0x00080000	IO 1 inverted

Table 32: Possible Values of Parameter Mode

Parameter **Baud Rate** (`ulPrmType= CAN_DL_PRM_TYPE_BAUDRATE=2`)

The configuration packet contains the baud rate of the CAN network. The following baud rates are supported: 1 MBit/s, 800 kBit/s, 500 kBit/s, 250 kBit/s, 125 kBit/s, 50 kBit/s, 20 kBit/s and 10 kBit/s.

The coding is as follows:

Possible Values of Parameter Baud Rate		
Option	**Value**	**Explanation**
`CAN_DL_BAUDRATE_AUTO_DETECT`	`0xAB`	Auto detect
`CAN_DL_BAUDRATE_1000kB`	`1000000`	1 MBit/s
`CAN_DL_BAUDRATE_800kB`	`800000`	800 kBit/s
`CAN_DL_BAUDRATE_500kB`	`500000`	500 kBit/s
`CAN_DL_BAUDRATE_250kB`	`250000`	250 kBit/s
`CAN_DL_BAUDRATE_125kB`	`125000`	125 kBit/s
`CAN_DL_BAUDRATE_100kB`	`100000`	100 kBit/s
`CAN_DL_BAUDRATE_50kB`	`50000`	50 kBit/s
`CAN_DL_BAUDRATE_20kB`	`20000`	20 kBit/s
`CAN_DL_BAUDRATE_12_5kB`	`12500`	12.5 kBit/s
`CAN_DL_BAUDRATE_10kB`	`10000`	10 kBit/s

Table 33: Possible Values of BaudRate

Parameter **Transmission abort time** (ulPrmType= CAN_DL_PRM_TYPE_TXABORT_TIME=3)

Here you can adjust the transmission abort time, i.e. the time after which a transmission will be aborted automatically. For an infinite transmission abort time, set the parameter to 0 (=CAN_DL_TX_ABORT_INFINITE)

Parameter **Events to indicate** (ulPrmType= CAN_DL_PRM_TYPE_EVENTS_TO_INDICATE =4)

This option can be used for specifying events to indicate.

Parameter **Filter** (ulPrmType= FILTER =5)

This option can be used for specifying a filter. This is done similarly to the description given in section “CAN_DL_CMD_SET_FILTER_REQ/CNF – Set Filter Request/ Confirmation “.

Parameter **Configuration lock** (ulPrmType= CAN_DL_PRM_TYPE_CONF_LOCK =6)

In this case, no data need to be specified. The configuration is locked.

Parameter **Configuration unlock** (ulPrmType= CAN_DL_PRM_TYPE_CONF_UNLOCK =7)

In this case, no data need to be specified. The configuration is unlocked.

Packet Structure Reference

```
/* Set parameter request */

typedef struct CAN_DL_ACCEPTANCE_FILTER_Ttag {
  TLR_UINT32  ulInstance;
  TLR_UINT32  ulCode;
  TLR_UINT32  ulMask;
} CAN_DL_ACCEPTANCE_FILTER_T;

/* Set parameter */
typedef struct CAN_DL_PARAMETER_SET_Ttag {
  TLR_UINT32                  ulMode;
  TLR_UINT32                  ulBaudrate;
  TLR_UINT32                  ulTxAbortTime;
  TLR_UINT32                  ulEventsIndicated;
  CAN_DL_ACCEPTANCE_FILTER_T  atFilter[2];
  TLR_UINT32                  ulNumRxIdEnable;
  TLR_UINT32                  ulFirstRxId;
} CAN_DL_PARAMETER_SET_T;

typedef union CAN_DL_PARAMETER_Ttag {
  CAN_DL_PARAMETER_SET_T      tSet;
  TLR_UINT32                  ulMode;
  TLR_UINT32                  ulBaudrate;
  TLR_UINT32                  ulTxAbortTime;
  TLR_UINT32                  ulEventsIndicated;
  CAN_DL_ACCEPTANCE_FILTER_T  tFilter;
} CAN_DL_PARAMETER_T;

typedef struct CAN_DL_SDU_SET_PRM_REQ_Ttag {
  TLR_UINT32              ulPrmType;
  CAN_DL_PARAMETER_T      unPrm;
} CAN_DL_SDU_SET_PRM_REQ_T;

typedef struct CAN_DL_PACKET_SET_PRM_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_SET_PRM_REQ_T      tData;        /* Packet Data Unit */
} CAN_DL_PACKET_SET_PRM_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_SET_PRM_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	80	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A0E	CAN_DL_CMD_SET_PRM_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_SET_PRM_REQ_T			
ulPrmType	UINT32	0-7	Parameter type (as described in *Table 29: Parameter types for Set Parameter Command*)
unPrm	CAN_DL_PARAMETER_T		Parameter structure

Table 34: CAN_DL_PACKET_SET_PRM_REQ_T – Set Parameter Request

Packet Structure Reference

```
/* Set parameter confirmation */

typedef struct CAN_DL_ACCEPTANCE_FILTER_Ttag {
  TLR_UINT32  ulInstance;
  TLR_UINT32  ulCode;
  TLR_UINT32  ulMask;
} CAN_DL_ACCEPTANCE_FILTER_T;

/* Set parameter */
typedef struct CAN_DL_PARAMETER_SET_Ttag {
  TLR_UINT32                  ulMode;
  TLR_UINT32                  ulBaudrate;
  TLR_UINT32                  ulTxAbortTime;
  TLR_UINT32                  ulEventsIndicated;
  CAN_DL_ACCEPTANCE_FILTER_T  atFilter[2];
  TLR_UINT32                  ulNumRxIdEnable;
  TLR_UINT32                  ulFirstRxId;
} CAN_DL_PARAMETER_SET_T;

typedef __PACKED_PRE  union CAN_DL_PARAMETER_Ttag {
  CAN_DL_PARAMETER_SET_T      tSet;
  TLR_UINT32                  ulMode;
  TLR_UINT32                  ulBaudrate;
  TLR_UINT32                  ulTxAbortTime;
  TLR_UINT32                  ulEventsIndicated;
  CAN_DL_ACCEPTANCE_FILTER_T  tFilter;
} __PACKED_POST CAN_DL_PARAMETER_T;


typedef struct CAN_DL_SDU_SET_PRM_REQ_Ttag {
  TLR_UINT32              ulPrmType;
  CAN_DL_PARAMETER_T      unPrm;
} CAN_DL_SDU_SET_PRM_REQ_T;

typedef CAN_DL_SDU_SET_PRM_REQ_T  CAN_DL_SDU_SET_PRM_CNF_T;

typedef struct CAN_DL_PACKET_SET_PRM_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_SET_PRM_CNF_T      tData;        /* Packet Data Unit */
} CAN_DL_PACKET_SET_PRM_CNF_T;
};
```

Packet Description

Structure Information CAN_DL_PACKET_SET_PRM_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	80	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A0F	CAN_DL_CMD_SET_PRM_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_SET_PRM_CNF_T			
ulPrmType	UINT32	0-7	Parameter type (as described in *Table 29: Parameter types for Set Parameter Command*)
unPrm	CAN_DL_PARAMETER_T		Parameter structure

Table 35: CAN_DL_PACKET_SET_PRM_CNF_T – Confirmation to Set Parameter Request

4.1.8 CAN_DL_CMD_EVENT_IND/RES – Event Indication/Response

Using this indication the application can get aware of and react to various events such as overruns of receive or transmit buffers or changes of the bus state (ERROR_PASSIVE and BUS_OFF events, for example).

The ulEvents parameter contains the information on the type of event that has occurred. It is coded as follows:

Coding of parameter ulEvents		
Bits	**Mask name**	**Explanation**
0..1	MSK_CAN_DL_BUS_STATE	Current bus state
2	MSK_CAN_DL_EVENT_BUS_STATE_CHANGE	Change of bus state
3	MSK_CAN_DL_EVENT_RX_OVERRUN	Receive overrun
4	MSK_CAN_DL_EVENT_TX_OVERRUN	Transmit overrun
16..17	MSK_CAN_DL_IO_INPUT_STATE	IO input state
16	MSK_CAN_DL_IO_0_INPUT_STATE	IO 0 input state
17	MSK_CAN_DL_IO_1_INPUT_STATE	IO 1 input state
18	MSK_CAN_DL_EVENT_IO_CHANGED	IO changed

Table 36: Coding of parameter ulEvents

The other bits are currently unused.

The bus state in bits 0 and 1 is coded as follows:

Value	Bus state
0	CAN_DL_BUSSTATE_ERR_ACTIVE
1	CAN_DL_BUSSTATE_ERR_WARNING
2	CAN_DL_BUSSTATE_ERR_PASSIVE
3	CAN_DL_BUSSTATE_BUS_OFF

Table 37: Bus State in Bits 0 and 1 of parameter ulEvents

This indication packet must be answered with the according response.

Packet Structure Reference

```
/* Event indication */

/* CAN bus state */
#define CAN_DL_BUSSTATE_ERR_ACTIVE              (0)
#define CAN_DL_BUSSTATE_ERR_WARNING             (1)
#define CAN_DL_BUSSTATE_ERR_PASSIVE             (2)
#define CAN_DL_BUSSTATE_BUS_OFF                 (3)

/* Masks CMD event indication */
#define MSK_CAN_DL_BUS_STATE                     0x00000003U
#define MSK_CAN_DL_EVENT_BUS_STATE_CHANGE        0x00000004U
#define MSK_CAN_DL_EVENT_RX_OVERRUN              0x00000008U
#define MSK_CAN_DL_EVENT_TX_OVERRUN              0x00000010U
#define MSK_CAN_DL_IO_INPUT_STATE                0x00030000U
#define MSK_CAN_DL_IO_0_INPUT_STATE              0x00010000U
#define MSK_CAN_DL_IO_1_INPUT_STATE              0x00020000U
#define MSK_CAN_DL_EVENT_IO_CHANGED              0x00040000U

typedef struct CAN_DL_SDU_EVENT_IND_Ttag {
  TLR_UINT32              ulEvents;
} CAN_DL_SDU_EVENT_IND_T;

typedef struct CAN_DL_PACKET_CAN_EVENT_IND_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_EVENT_IND_T        tData;        /* Packet Data      */
} CAN_DL_PACKET_EVENT_IND_T;
```

Packet Description

Structure Information CAN_DL_PACKET_EVENT_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A10	CAN_DL_CMD_EVENT_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_EVENT_IND_T			
ulEvents	UINT32	Bit mask	Event indication mask

Table 38: CAN_DL_PACKET_EVENT_IND_T – Event Indication

Packet Structure Reference

```
/* Event response */

/* Masks CMD event indication */
#define MSK_CAN_DL_BUS_STATE                      0x00000003U
#define MSK_CAN_DL_EVENT_BUS_STATE_CHANGE         0x00000004U
#define MSK_CAN_DL_EVENT_RX_OVERRUN               0x00000008U
#define MSK_CAN_DL_EVENT_TX_OVERRUN               0x00000010U
#define MSK_CAN_DL_IO_INPUT_STATE                 0x00030000U
#define MSK_CAN_DL_IO_0_INPUT_STATE               0x00010000U
#define MSK_CAN_DL_IO_1_INPUT_STATE               0x00020000U
#define MSK_CAN_DL_EVENT_IO_CHANGED               0x00040000U

typedef struct CAN_DL_SDU_EVENT_RES_Ttag {
  TLR_UINT32             ulEvents;
} CAN_DL_SDU_EVENT_RES_T;

typedef struct CAN_DL_PACKET_EVENT_RES_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_EVENT_RES_T        tData;        /* Packet Data      */
} CAN_DL_PACKET_EVENT_RES_T;
```

Packet Description

Structure Information CAN_DL_PACKET_EVENT_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A11	CAN_DL_CMD_EVENT_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_EVENT_RES_T			
ulEvents	UINT32	Bit mask	Event indication mask

Table 39: CAN_DL_PACKET_EVENT_RES_T – Response to Event Indication

4.1.9 CAN_DL_CMD_ENABLE_RXID_REQ/CNF – Enable Identifier Request/ Confirmation

This request allows to specify information in order to enable or disable certain CAN Identifiers for the receiving channel in the CAN DL Task.

The request packet has one parameter, the structure CAN_DL_ENABLE_RXID_T. This structure looks as follows:

Variable	Type	Range	Explanation
ulControlBits	UINT32	0..7 (Bit mask)	Control bits
ulIdentifier	UINT32		Identifier to be enabled or disabled
ulUniqueId	UINT32		Unique ID

Table 40: Structure CAN_DL_ENABLE_RXID_T

The control bits have the following meaning:

Bit 0	**MSK_CAN_DL_CTRL_RXID_ENABLE**	**Enable/disable bit**
=0	If cleared, the following identifier has to be disabled.	
=1	If set, the following identifier has to enabled.	
Bit 1	**MSK_CAN_DL_CTRL_RXID_RANGE_START**	**Range start bit**
=0	If cleared, the following identifier is a regular CAN identifier	
=1	If set, the following identifier marks the beginning of a consecutive range of identifiers	
Bit 2	**MSK_CAN_DL_CTRL_RXID_RET_USR_UUID**	**Return User UUID bit.**
=1	If set, the following unique ID will be returned with the frame.	

Table 41: Meaning of Control Bits

The CAN DL Task returns a reply packet to the Fieldbus Task to inform about success or failure of the request. Each CAN identifier is marked whether OK or not OK (NOK) in the reply packet. Undefined/unused bits and bytes are reserved and set to 0 (zero).

Packet Structure Reference

```
/* Enable identifier request */

#define CAN_DL_MAX_ENABLE_RXID_PER_PACKET       (8)

/* Masks CMD enable identifier request */
#define MSK_CAN_DL_CTRL_RXID_ENABLE              0x00000001U
#define MSK_CAN_DL_CTRL_RXID_RANGE_START         0x00000002U
#define MSK_CAN_DL_CTRL_RXID_RET_USR_UUID        0x00000004U

/* Enable CAN identifier */
typedef struct CAN_DL_ENABLE_RXID_Ttag {
  TLR_UINT32  ulControlBits;
  TLR_UINT32  ulIdentifier;
  TLR_UINT32  ulUniqueId;
} CAN_DL_ENABLE_RXID_T;

typedef struct CAN_DL_SDU_ENABLE_RXID_REQ_Ttag {
  CAN_DL_ENABLE_RXID_T    atId[CAN_DL_MAX_ENABLE_RXID_PER_PACKET];
} CAN_DL_SDU_ENABLE_RXID_REQ_T;

typedef struct CAN_DL_PACKET_ENABLE_RXID_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_ENABLE_RXID_REQ_T  tData;        /* Packet Data Unit */
} CAN_DL_PACKET_ENABLE_RXID_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_ENABLE_RXID_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	12, 24, 36, 48, 60, 72, 84, 96	Packet Data Length in bytes (depending on number of members of structure `atId[]`
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A12	CAN_DL_CMD_ENABLE_RXID_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_ENABLE_RXID_REQ_T			
atId[]	CAN_DL_ENABLE_RXID_T[8]		Structure containing identifiers or ranges of identifiers to be enabled or disabled.

Table 42: `CAN_DL_PACKET_ENABLE_RXID_REQ_T` – Enable Identifier Request

Packet Structure Reference

```
/* Enable identifier confirmation */

typedef struct CAN_DL_PACKET_ENABLE_RXID_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_ENABLE_RXID_CNF_T;};
```

Packet Description

Structure Information CAN_DL_PACKET_ENABLE_RXID_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A13	CAN_DL_CMD_ENABLE_RXID_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 43: CAN_DL_PACKET_ENABLE_RXID_CNF_T – Confirmation to Enable Identifier Request

4.1.10 CAN_DL_CMD_EVENT_ACK_REQ/CNF – Event Acknowledge Request/ Confirmation

All indicated events have to be acknowledged by the application task. No new event of the same kind will be reported, until the indicated events has been confirmed.

The `ulEvents` parameter contains the information on the type of event to be acknowledged. It is coded as follows:

Coding of parameter `ulEvents`		
Bits	**Mask name**	**Explanation**
0..1	MSK_CAN_DL_BUS_STATE	Current bus state
2	MSK_CAN_DL_EVENT_BUS_STATE_CHANGE	Change of bus state
3	MSK_CAN_DL_EVENT_RX_OVERRUN	Receive overrun
4	MSK_CAN_DL_EVENT_TX_OVERRUN	Transmit overrun
16..17	MSK_CAN_DL_IO_INPUT_STATE	IO input state
16	MSK_CAN_DL_IO_0_INPUT_STATE	IO 0 input state
17	MSK_CAN_DL_IO_1_INPUT_STATE	IO 1 input state
18	MSK_CAN_DL_EVENT_IO_CHANGED	IO changed

Table 44: Coding of parameter `ulEvents`

The other bits are currently unused.

Packet Structure Reference

```
/* Event Acknowledge  request */

/* Masks CMD event indication */
#define MSK_CAN_DL_BUS_STATE                    0x00000003U
#define MSK_CAN_DL_EVENT_BUS_STATE_CHANGE       0x00000004U
#define MSK_CAN_DL_EVENT_RX_OVERRUN             0x00000008U
#define MSK_CAN_DL_EVENT_TX_OVERRUN             0x00000010U
#define MSK_CAN_DL_IO_INPUT_STATE               0x00030000U
#define MSK_CAN_DL_IO_0_INPUT_STATE             0x00010000U
#define MSK_CAN_DL_IO_1_INPUT_STATE             0x00020000U
#define MSK_CAN_DL_EVENT_IO_CHANGED             0x00040000U

typedef struct CAN_DL_SDU_EVENT_ACK_REQ_Ttag {
  TLR_UINT32              ulEvents;
} CAN_DL_SDU_EVENT_ACK_REQ_T;

typedef struct CAN_DL_PACKET_CAN_EVENT_ACK_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_EVENT_ACK_REQ_T    tData;        /* Packet Data      */
} CAN_DL_PACKET_EVENT_ACK_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_EVENT_ACK_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A14	CAN_DL_CMD_EVENT_ACK_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_EVENT_ACK_REQ_T			
ulEvents	UINT32	Bit mask	Event indication mask

Table 45: CAN_DL_PACKET_EVENT_ACK_REQ_T – Event Acknowledge Request

Packet Structure Reference

```
/* Event Acknowledge confirmation */

typedef struct CAN_DL_SDU_EVENT_ACK_CNF_Ttag {
  TLR_UINT32              ulEvents;
} CAN_DL_SDU_EVENT_ACK_CNF_T;

typedef struct CAN_DL_PACKET_EVENT_ACK_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_EVENT_ACK_CNF_T    tData;        /* Packet Data      */
} CAN_DL_PACKET_EVENT_ACK_CNF_T;
```

Packet Description

Structure Information CAN_DL_PACKET_EVENT_ACK_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A15	CAN_DL_CMD_EVENT_ACK_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_EVENT_ACK_CNF_T			
ulEvents	UINT32	Bit mask	Event indication mask

Table 46: CAN_DL_PACKET_EVENT_ACK_CNF_T – Confirmation to Event Acknowledge Request

4.1.11 CAN_DL_CMD_DIAG_REQ/CNF – Diagnosis Request/ Confirmation

This packet can be used to request diagnostic data. The kind of data to be delivered can be selected by setting parameter `ulDiagType` of the request packet to the values 0, 1 or 2.

1. Setting `ulDiagType` of the request packet to `CAN_DL_DIAG_TYPE_CMD1 = 0` will cause a complete diagnostic report to be returned in the confirmation packet. This report contains a lot of counters, see *Table 5: Diagnostic Report*.

2. Setting `ulDiagType` of the request packet to `CAN_DL_DIAG_TYPE_CMD2 = 1` will return the command statistics in the confirmation packet. The command statistics is structured as described in *Table 6: Command Statistics*.

3. Setting `ulDiagType` of the request packet to `CAN_DL_DIAG_TYPE_STATUS = 2` will return the CAN Status List in the confirmation packet. For detailed information about the CAN Status List see *Table 7: CAN Status List*.

The confirmation packet has two parameters namely the selected `ulDiagType` and the union `unDiag` which contains the requested diagnostic data.

Undefined or unused bits and bytes are reserved and set to 0.

Packet Structure Reference

```
/* Diag request */

/* Task diagnostic */
#define CAN_DL_DIAG_TYPE_CMD1         (0)
#define CAN_DL_DIAG_TYPE_CMD2         (1)
#define CAN_DL_DIAG_TYPE_STATUS       (2)

typedef struct CAN_DL_SDU_DIAG_REQ_Ttag {
  TLR_UINT32              ulDiagType;
} CAN_DL_SDU_DIAG_REQ_T;

typedef struct CAN_DL_PACKET_CAN_DIAG_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_DIAG_REQ_T         tData;        /* Packet Data      */
} CAN_DL_PACKET_DIAG_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_DIAG_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A16	CAN_DL_CMD_DIAG_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure XXX_CNF_DATA_T			
ulDiagType	UINT32	0-2	Type of diagnosis to be requested

Table 47: CAN_DL_SDU_DIAG_REQ_T – Diagnosis Request

Packet Structure Reference

```
/* Diag confirmation */

typedef  struct CAN_DL_DIAG_CMD2_Ttag {
  /* Command statistic */

  TLR_UINT32               ulCyclicEvent;
  TLR_UINT32               ulDrvCanRxInd;
  TLR_UINT32               ulDrvCanEveInd;

  TLR_UINT32               ulCmdDiagReq;
  TLR_UINT32               ulCmdDiagCnfPos;
  TLR_UINT32               ulCmdDiagCnfNeg;

} CAN_DL_DIAG_CMD2_T;

typedef struct CAN_DL_DIAG_STATUS_Ttag{
/* CAN Status List */

  TLR_UINT32 ulStatus;
  TLR_UINT32 ulTxFrameSucceed;
  TLR_UINT32 ulTxErrorSummary;
  TLR_UINT32 ulRxFrameSucceed;
  TLR_UINT32 ulRxErrorSummary;
  TLR_UINT32 ulTxErrCnt;
  TLR_UINT32 ulRxErrCnt;
  TLR_UINT32 ulArbitrationLost;
  TLR_UINT32 ulIndDroppedDueFifoFull;
  TLR_UINT32 ulConDroppedDueFifoFull;
  TLR_UINT32 ulRxStdFramesFilterd;
  TLR_UINT32 ulRxExtFramesFilterd;
  TLR_UINT32 ulRxStdFramesPassed;
  TLR_UINT32 ulRxExtFramesPassed;

} CAN_DL_DIAG_STATUS_T;

typedef union CAN_DL_DIAG_Ttag {
  CAN_DL_DIAG_CMD1             tCmd1;
  CAN_DL_DIAG_CMD2_T           tCmd2;
  CAN_DL_DIAG_STATUS_T         tStatus;
} CAN_DL_DIAG_T;

typedef struct CAN_DL_SDU_DIAG_CNF_Ttag {
  TLR_UINT32              ulDiagType;
  CAN_DL_DIAG_T           unDiag;
} CAN_DL_SDU_DIAG_CNF_T;

typedef struct CAN_DL_PACKET_DIAG_CNF_Ttag {
  TLR_PACKET_HEADER_T          tHead;        /* Packet Header    */
  CAN_DL_SDU_DIAG_CNF_T        tData;        /* Packet Data      */
} CAN_DL_PACKET_DIAG_CNF_T;
```

Packet Description

Structure Information CAN_DL_PACKET_DIAG_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	152	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A17	CAN_DL_CMD_DIAG_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_DIAG_CNF_T			
ulDiagType	UINT32	0-2	Type of requested diagnosis
unDiag	CAN_DL_DIAG_T		Diagnostic data according to requested type

Table 48: `CAN_DL_PACKET_DIAG_CNF_T` – Confirmation to Diagnosis Request

4.1.12 CAN_DL_CMD_TX_ABORT_REQ/CNF – Transmission Abort

This request can be used by the application to abort a running transmission. Depending on the transmission abort mode, you may choose whether to abort the transmission of all CAN frames or only the transmission of the currently transmitted CAN frame by choosing the value 0 or 1 for this parameter.

If activated, this command forces the communication controller xMAC/xPEC to abort the last Send command immediately.

Packet Structure Reference

```
/* Tx Abort  request */

/* Value for transmit abort timer infinite */
#define   CAN_DL_TX_ABORT_INFINITE              (0)

/* TX abort modes */
#define CAN_DL_ABORT_ALL_CAN_FRAME              (0)
#define CAN_DL_ABORT_ACTUAL_CAN_FRAME           (1)

typedef struct CAN_DL_TX_ABORT_REQ_Ttag {
  TLR_UINT32              ulMode;
} CAN_DL_SDU_TX_ABORT_REQ_T;

typedef struct CAN_DL_PACKET_TX_ABORT_REQ_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_TX_ABORT_REQ_T     tData;        /* Packet Data      */
} CAN_DL_PACKET_TX_ABORT_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_TX_ABORT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A1A	CAN_DL_CMD_TX_ABORT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_TX_ABORT_REQ_T			
ulMode	UINT32	0,1	Transmission abort mode

Table 49: CAN_DL_PACKET_TX_ABORT_REQ_T – Transmission Abort Request

Packet Structure Reference

```
/* Tx Abort confirmation */

typedef struct CAN_DL_PACKET_TX_ABORT_CNF_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
} CAN_DL_PACKET_TX_ABORT_CNF_T; ,
```

Packet Description

Structure Information CAN_DL_PACKET_TX_ABORT_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002A1B	CAN_DL_CMD_TX_ABORT_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 50: CAN_DL_PACKET_TX_ABORT_CNF_T – Confirmation to Transmission Abort Request

4.1.13 CAN_DL_CMD_AUTO_BAUD_IND/RES – Auto Baud Detection Complete Indication

This indication signals the completion of the automatic baud rate detection process.

The following baud rates may be indicated and are available as parameter for the response packet:

Baudrate	Symbolic name	Value
1 MBit/s	CAN_DL_BAUDRATE_1000kB	1000000
800 kBit/s	CAN_DL_BAUDRATE_800kB	800000
500 kBit/s	CAN_DL_BAUDRATE_500kB	500000
250 kBit/s	CAN_DL_BAUDRATE_250kB	250000
125 kBit/s	CAN_DL_BAUDRATE_125kB	125000
100 kBit/s	CAN_DL_BAUDRATE_100kB	100000
50 kBit/s	CAN_DL_BAUDRATE_50kB	50000
20 kBit/s	CAN_DL_BAUDRATE_20kB	20000
12.5 kBit/s	CAN_DL_BAUDRATE_12_5kB	12500
10 kBit/s	CAN_DL_BAUDRATE_10kB	10000

Table 51: Possible Baud Rates in CAN_DL_CMD_AUTO_BAUD_IND/RES

Packet Structure Reference

```
/* Packet to indicate that auto baud detection was complete */

typedef struct CAN_DL_SDU_AUTO_BAUD_IND_Ttag {
  TLR_UINT32              ulBaudRate;
} CAN_DL_SDU_AUTO_BAUD_IND_T;

typedef struct CAN_DL_PACKET_AUTO_BAUD_IND_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_AUTO_BAUD_IND_T    tData;        /* Packet Data Unit */
} CAN_DL_PACKET_AUTO_BAUD_IND_T;
```

Packet Description

Structure Information CAN_DL_PACKET_AUTO_BAUD_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AE8	CAN_DL_CMD_AUTO_BAUD_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_AUTO_BAUD_IND_T			
ulBaudRate	UINT32	See above	Baud rate

Table 52: CAN_DL_PACKET_AUTO_BAUD_IND_T – Auto Baud Detection Complete Indication

Packet Structure Reference

```
/* Response Packet to indication that auto baud detection was complete */

typedef struct CAN_DL_SDU_AUTO_BAUD_RES_Ttag {
  TLR_UINT32              ulBaudRate;
} CAN_DL_SDU_AUTO_BAUD_RES_T;

typedef struct CAN_DL_PACKET_AUTO_BAUD_RES_Ttag {
  TLR_PACKET_HEADER_T           tHead;        /* Packet Header    */
  CAN_DL_SDU_AUTO_BAUD_RES_T    tData;        /* Packet Data Unit */
} CAN_DL_PACKET_AUTO_BAUD_RES_T;
```

Packet Description

Structure Information CAN_DL_PACKET_AUTO_BAUD_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AE9	CAN_DL_CMD_AUTO_BAUD_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_AUTO_BAUD_RES_T			
ulBaudRate	UINT32	See above	Baud rate which has been indicated

Table 53: CAN_DL_PACKET_AUTO_BAUD_RES_T – Response to Auto Baud Detection Complete Indication

4.1.14 CAN_DL_CMD_GET_HANDLE_REQ/CNF – Get Handle

Using this packet you can get two handles for further use with the function interface

Note: Use this packet only when working with the function interface. It is not necessary to use this packet when working with the packet interface..

Note: This packet is used by the AP-task only and will not be routed from the user application to the Fieldbus task.

Packet Structure Reference

```
typedef struct CAN_DL_PACKET_GET_HANDLE_REQ_Ttag {
  TLR_PACKET_HEADER_T            tHead;        /* Packet Header    */
} CAN_DL_PACKET_GET_HANDLE_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_GET_HANDLE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AEA	CAN_DL_CMD_GET_HANDLE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 54: CAN_DL_PACKET_GET_HANDLE_REQ_T – Get Handle Request

Packet Structure Reference

```
typedef struct CAN_DL_SDU_GET_HANDLE_CNF_Ttag {
  TLR_HANDLE              hRscHandle;
  TLR_HANDLE              hIdentHandle;
} CAN_DL_SDU_GET_HANDLE_CNF_T;

typedef struct CAN_DL_PACKET_GET_HANDLE_CNF_Ttag {
  TLR_PACKET_HEADER_T            tHead;        /* Packet Header    */
  CAN_DL_SDU_GET_HANDLE_CNF_T    tData;        /* Packet Data Unit */
} CAN_DL_PACKET_GET_HANDLE_CNF_T };
```

Packet Description

Structure Information CAN_DL_PACKET_GET_HANDLE_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	8	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AEB	CAN_DL_CMD_GET_HANDLE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_GET_HANDLE_CNF_T			
hRscHandle	TLR_HANDLE		Resource handle
hIdentHandle	TLR_HANDLE		Ident handle

Table 55: CAN_DL_PACKET_GET_HANDLE_CNF_T – Confirmation to Get Handle Request

4.1.15 CAN_DL_CMD_SET_AUTOBAUD_FLAGS_REQ/CNF – Set Autobaud Flags

Fieldbus stacks which use auto baud detection have to set baud rates which should be scanned while detection process.. This can be done using this packet.

There is only one parameter in the request packet, namely `ulAutoBaudFlags` containing flags. Set the autobaud flags as follows:

Variable	Value	Bits	Baudrate
MSK_CAN_DL_BAUDRATE_1000kB	0x00000001	0	1 MBit/s
MSK_CAN_DL_BAUDRATE_800kB	0x00000002	1	800 kBit/s
MSK_CAN_DL_BAUDRATE_500kB	0x00000004	2	500 kBit/s
MSK_CAN_DL_BAUDRATE_250kB	0x00000008	3	250 kBit/s
MSK_CAN_DL_BAUDRATE_125kB	0x00000010	4	125 kBit/s
MSK_CAN_DL_BAUDRATE_100kB	0x00000020	5	100 kBit/s
MSK_CAN_DL_BAUDRATE_50kB	0x00000040	6	50 kBit/s
MSK_CAN_DL_BAUDRATE_20kB	0x00000080	7	20 kBit/s
MSK_CAN_DL_BAUDRATE_12_5kB	0x00000100	8	12.5 kBit/s
MSK_CAN_DL_BAUDRATE_10kB	0x00000200	9	10 kBit/s

Table 56: Autobaud Flags

If the respective bit is set, the corresponding baudrate will be used during the automatic baudrate detection process. If it is not set, the corresponding baudrate will be omitted.

Packet Structure Reference

```
/* Set auto baud flags request */

typedef  struct CAN_DL_SDU_SET_AUTOBAUD_FLAGS_REQ_Ttag {
  TLR_UINT32      ulAutoBaudFlags;
} CAN_DL_SDU_SET_AUTOBAUD_FLAGS_REQ_T;

typedef struct CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_REQ_Ttag {
  TLR_PACKET_HEADER_T                       tHead;        /* Packet Header    */
  CAN_DL_SDU_SET_AUTOBAUD_FLAGS_REQ_T       tData;        /* Packet Data Unit */
} CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_REQ_T;
```

Packet Description

Structure Information CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AEC	CAN_DL_CMD_SET_AUTOBAUD_FLAGS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_SET_AUTOBAUD_FLAGS_REQ_T			
ulAutoBaudFlags	UINT32	0..1023	Auto Baud Flags

Table 57: CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_REQ_T – Set Autobaud Flags Request

Packet Structure Reference

```
/* Set auto baud flags confirmation */

typedef struct CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_CNF_Ttag {
  TLR_PACKET_HEADER_T                        tHead;        /* Packet Header    */
} CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_CNF_T; ^
```

Packet Description

Structure Information CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AED	CAN_DL_CMD_SET_AUTOBAUD_FLAGS_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 58: CAN_DL_PACKET_SET_AUTOBAUD_FLAGS_CNF_T – Confirmation to Set Autobaud Flags Request

4.1.16 CAN_DL_CMD_SET_EVENTS_TO_INDICATE_REQ/CNF – Register Events

Every application of the CAN DL task can register for certain events.

The "*Set parameter*" packet provides also this functionality with parameter type option `CAN_DL_PRM_TYPE_EVENTS_TO_INDICATE`

Possible error situations which might occur are:

- Configuration locked (Error code: `TLR_E_CAN_DL_CONF_LOCKED`)
- Invalid packet length (Error code: `TLR_E_CAN_DL_CMD_LENGTH_MISMATCH`)
- Set events requested failed

  (Error code: `TLR_E_CAN_DL_SET_EVENTS_REQUESTED_FAILED`)

Packet Structure Reference

```
/* Separate command to register certain events */
typedef  struct CAN_DL_SDU_SET_EVENTS_TO_INDICATE_REQ_Ttag {
  TLR_UINT32      ulEventsIndicated;
} CAN_DL_SDU_SET_EVENTS_TO_INDICATE_REQ_T;

typedef struct CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_REQ_Ttag {
  TLR_PACKET_HEADER_T                         tHead;        /* Packet Header    */
  CAN_DL_SDU_SET_EVENTS_TO_INDICATE_REQ_T     tData;        /* Packet Data Unit */
} CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_REQ_T;
```

Packet Description

Structure Information **CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_REQ_T**			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue-Handle of CAN-DL-Task Process Queue
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle of AP-Task Process Queue
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AEE	CAN_DL_CMD_SET_EVENTS_TO_INDICATE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
Structure CAN_DL_SDU_SET_EVENTS_TO_INDICATE_REQ_T			
ulEventsIndicated	UINT32		Indicated events

Table 59: `CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_REQ_T` – Register Events Request

Packet Structure Reference

```
typedef struct CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_CNF_Ttag {
  TLR_PACKET_HEADER_T                          tHead;        /* Packet Header    */
} CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_CNF_T;
```

Packet Description

Structure Information **CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_CNF_T**			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle, untouched
ulSrc	UINT32		Source Queue-Handle, untouched
ulDestId	UINT32		Destination End Point Identifier, untouched
ulSrcId	UINT32		Source End Point Identifier, untouched
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section *Status/Error Codes Overview* on page *68*
ulCmd	UINT32	0x00002AEF	CAN_DL_CMD_SET_EVENTS_TO_INDICATE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 60: CAN_DL_PACKET_SET_EVENTS_TO_INDICATE_CNF_T – Confirmation to Register Events Request

5 Status/Error Codes Overview

Value	Definition/Description
0x00000000	TLR_S_OK Status ok
0xC03F0001	TLR_E_CAN_DL_COMMAND_INVALID Invalid command.
0xC03F0002	TLR_E_CAN_DL_CMD_LENGTH_MISMATCH The length code of the command is invalid.
0xC03F0003	TLR_E_CAN_DL_UNKNOWN_PARAMETER_TYPE The parameter type of the command "Set Parameter" is invalid.
0xC03F0004	TLR_E_CAN_DL_SET_MODE_FAILED Within the command "Set Parameter" the function set "CAN Mode" failed.
0xC03F0005	TLR_E_CAN_DL_SET_BAUDRATE_FAILED Within the command "Set Parameter" the function set "Baudrate" failed.
0xC03F0006	TLR_E_CAN_DL_SET_TXABORT_TIME_FAILED Within the command "Set Parameter" the function set "Transmission Abort Timer" failed.
0xC03F0007	TLR_E_CAN_DL_SET_EVENTS_REQUESTED_FAILED Within the command "Set Parameter" the function set "Requetsed Events" failed.
0xC03F0008	TLR_E_CAN_DL_SET_FILTER_FAILED Within the command "Set Parameter" or "Set Filter the function set "CAN Filter" failed.
0xC03F0009	TLR_E_CAN_DL_SET_ENABLE_DISABLE_RXID_FAILED Within the command Enable or Diasble of receive identifiers an error occurred.
0xC03F000A	TLR_E_CAN_DL_TX_FRAME_FAILED At least one CAN frame could not be sent. Normally because the send process was aborted by the transmission abort timer.
0xC03F000B	TLR_E_CAN_DL_TX_BUFFER_OVERRUN The send request of CAN frames was rejected because the internal buffer for send requests is full.
0xC03F000C	TLR_E_CAN_DL_UNKNOWN_DIAG_TYPE The diagnostic type of the command "Get Diag" is invalid.
0xC03F000D	TRL_E_CAN_DL_TX_ABORT_ALREADY_IN_REQUEST T he command "Transmission Abort" is already requested.
0xC03F000E	TRL_E_CAN_DL_TX_ABORT The send process of can frames was aborted by "Transmission Abort" command.
0xC03F000F	TRL_E_CAN_DL_UNKNOWN_APPLICATION The application makes access, is not registered at CAN_DL task.
0xC03F0010	TLR_E_CAN_DL_AP_ALREADY_REGISTERED The application is already registered.
0xC03F0011	TLR_E_CAN_DL_CONF_LOCK_FAIL The configuration lock failed.
0xC03F0012	TLR_E_CAN_DL_CONF_LOCKED The configuration is locked.

Table 61: Status/Error Codes Overview

6 Appendix

6.1 List of Tables

6.2 List of Figures

6.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
New Delhi - 110 025
Phone: +91 11 40515640
E-Mail: info@hilscher.in

Italy
Hilscher Italia srl
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Suwon, 443-734
Phone: +82 (0) 31-695-5515
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com