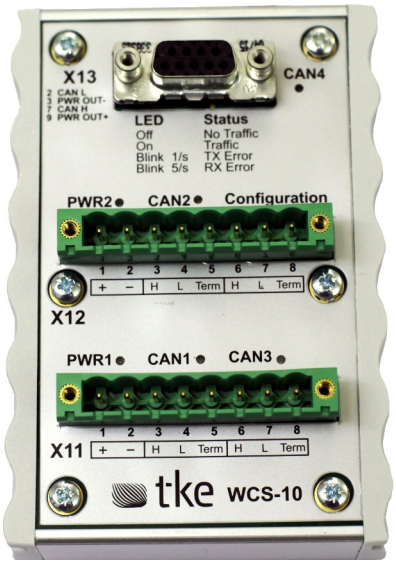
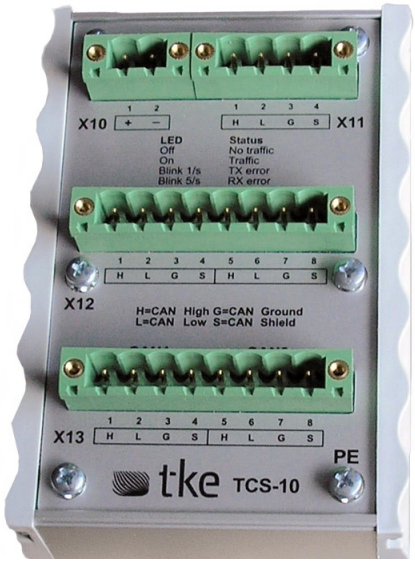
CAN Switch Firmware User Guide





The firmware in the TKE CAN Switch (WCS-10, TCS-10) copies CAN messages between the four routing CAN ports of the CAN Switch and provides configuration and monitoring services, through CANopen, on the additional, fifth, CAN port.

Table of Contents

Introduction

This document describes how to use the CAN switch and all features. Several constraints have been met with the standard CANGW-firmware, when used as the CAN network switch.

Technical Data

Buffer size: 15 CAN message transmit buffer on routing ports.

Transmit order: Messages are sent in FIFO order.

Product code: 0xFCB.

Factory Settings

The very first time when the CAN switch is started up it has hardcode factory settings in routing table and CAN-interfaces. The factory settings are as follow:

- **Acceptance Filter:** All CAN-interfaces has filter 0xFFFFFFFF. Additional information referenced to /2/ Chapter 13.4.3
- **Identifier Mask Register:** All CAN-interfaces has mask 0xFFFFFFFF. Additional information referenced to /2/ Chapter 13.3.2.18
- **Acceptance Control Register:** All CAN-interfaces has Control Register (IDAM0-IDAM1) value 0x00. Additional information referenced to /2/ Chapter 13.3.2.12 NOTE: IDHIT0-IDHIT2 are never changed.
- **Bit rate:** CAN1-4 bitrates are defined to 500kbit/s, CANA initial bit rate is 500kbit/s.
- **Route table:** All CAN IDs are forwarded to all ports except the port where it was received.
- **CANopen node ID:** 0x7F
- **Heartbeat time:** 1000
- **Operational mode:** The CANopen stack starts in Operational mode.
- **EMCY Inhibit time:** The EMCY inhibit time factor is set to 1000 (0x3EB). The factor is multiples of 100micro seconds. Ref /3/ This yields an EMCY inhibit time of 100ms.

Note: The factory configuration for acceptance filter, identifier mask register and acceptance control register behave as “pass all”.

Device Configuration

While the CAN switch is CAN networking device, configuration is object-dictionary oriented to enable use of standard CANopen configuration tools, EDS- and DCF-files.

Object Dictionary Overview

Table 1: Communication profile area

Index	Sub-idx	Type		Description
0x1000	0x00	U32		Device type (0x0000 012D)
0x100B	0x00	U8		Node ID
0x1010		Record		Store parameters
	0x00	U8		Largest sub-index supported
	0x01	U32		Store all parameters (“EVAS” = 0x65 76 61 73)
	0x02	U32		Store communication parameters (0x1000..0x1FFF)
	0x03	U32		Store application parameters (0x2000..0x9FFF)
0x1015	0x00	U16		Inhibit time EMCY (multiple of 100μs)
0x1017	0x00	U16		Producer heartbeat time (multiple of 1ms)
0x1018		Identity		Identity object
	0x00	U8		Largest sub-index supported
	0x01	U32		Vendor-ID
	0x02	U32		Product code
	0x03	U32		Revision number
	0x04	U32		Serial number

Table 2: Manufacturer-specific area

Index	Sub-idx	Type		Description
0x2000	0x00	U8		Heartbeat Enable/Disable
0x5010				CAN1 bit rate
	0x00	U8		Largest sub-index supported
	0x01	U8		CAN1 register BTR0
	0x02	U8		CAN1 register BTR1
0x5011	0x00	U32		CAN1 acceptance filter, bank 0
0x5012	0x00	U32		CAN1 acceptance filter, bank 1
0x5015	0x00	U32		CAN1 acceptance mask, bank 0
0x5016	0x00	U32		CAN1 acceptance mask, bank 1
0x5019	0x00	U8		CAN1 filter configuration
0x5020				CAN2 bit rate
	0x00	U8		Largest sub-index supported
	0x01	U8		CAN2 register BTR0
	0x02	U8		CAN2 register BTR1
0x5021	0x00	U32		CAN2 acceptance filter, bank 0
0x5022	0x00	U32		CAN2 acceptance filter, bank 1
0x5025	0x00	U32		CAN2 acceptance mask, bank 0
0x5026	0x00	U32		CAN2 acceptance mask, bank 1
0x5029	0x00	U8		CAN2 filter configuration
0x5030				CAN3 bit rate
	0x00	U8		Largest sub-index supported

	0x01	U8		CAN3 register BTR0
	0x02	U8		CAN3 register BTR1
0x5031	0x00	U32		CAN3 acceptance filter, bank 0
0x5031	0x00	U32		CAN3 acceptance filter, bank 1
0x5035	0x00	U32		CAN3 acceptance mask, bank 0
0x5036	0x00	U32		CAN3 acceptance mask, bank 1
0x5039	0x00	U8		CAN3 filter configuration
0x5040				CAN4 bit rate
	0x00	U8		Largest sub-index supported
	0x01	U8		CAN4 register BTR0
	0x02	U8		CAN4 register BTR1
0x5041	0x00	U32		CAN4 acceptance filter, bank 0
0x5042	0x00	U32		CAN4 acceptance filter, bank 1
0x5045	0x00	U32		CAN4 acceptance mask, bank 0
0x5046	0x00	U32		CAN4 acceptance mask, bank 1
0x5049	0x00	U8		CAN4 filter configuration
0x5050				CANA Configuration port bit rate
	0x00	U8		Largest sub-index supported
	0x01	U8		CANA register BTR0
	0x02	U8		CANA register BRT1

Table 3: Device-profile specific area

Index	**Sub-idx**	**Type**	**Bits**	**Description**
0x6800	0x01	U32		Universal ID route settings (Ignored)
	0x02	U16		Array index 0x000 routing
			0..3	Destinations for message received from port 1
			4..7	Destinations for message received from port 2
			8..11	Destinations for message received from port 3
			12..15	Destinations for message received from port 4
0x6801	0x01	U32		CAN-ID x
	0x02	U16		Array index 0x001 routing
			0..3	Destinations for message received from port 1
			4..7	Destinations for message received from port 2
			8..11	Destinations for message received from port 3
			12..15	Destinations for message received from port 4
⋮				
0x6864	0x01	U32		CAN-ID x
	0x02	U16		Array index 0x64 routing
			0..3	Destinations for message received from port 1
			4..7	Destinations for message received from port 2
			8..11	Destinations for message received from port 3
			12..15	Destinations for message received from port 4

Global Parameters

Device Type (0x1000)

Device type indicates rough category of the device. CAN switch equals pure CANopen-device without any device-profile specific functionality or application programmability. No additional information is provided.

Table 4: 0x1000 object description

Attribute	Value
Index	0x1000
Name	Device type
Object code	Variable
Data type	U32
Category	Mandatory

Table 5: 0x1000 entry description

Attribute	Value
Sub-index	0x00
Access	Constant
PDO mapping	No
Value range	U32
Default value	0x0000 012D (DS-301)

Node-ID (0x100B)

Table 6: 0x100B object description

Attribute	Value
Index	0x100B
Name	Node-ID
Object code	Variable
Data type	U8
Category	Optional

Table 7: 0x100B entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	0x01..0x7F
Default value	0x7F

Node-ID applies only to the configuration port (port CAN A).

Store Parameters (0x1010)

When storing parameters, the device stores the configuration and routing-table in non-volatile memory (flash). The device does not support automated storing. When storing data, the device follows the standard Ref /3/. The value to send is ASCII value for “SAVE”.

Table 8: 0x1010 object description

Attribute	Value
Index	0x1010
Name	Store parameters
Object code	Array
Data type	U32
Category	Optional

Table 9: 0x1010 entry description

Attribute	Value
Sub-index	0x00
Description	Highest sub-index supported
Access	RO
PDO mapping	No
Value range	0x01..0x7F
Default value	0x03
Sub-index	0x01
Description	Save all parameters
Access	RW
PDO mapping	No
Value range	See Table 10
Default value	profile- or manufacturer-specific
Sub-index	0x02
Description	Save communication parameters
Access	RW
PDO mapping	No
Value range	See Table 10
Default value	profile- or manufacturer-specific
Sub-index	0x03
Description	Save application parameters
Access	RW
PDO mapping	No
Value range	See Table 10
Default value	profile- or manufacturer-specific

Table 10: Storage write access signature

Byte3 (MSB)	Byte2	Byte1	Byte0 (LSB)
“E”	“V”	“A”	“S”
0x65	0x76	0x61	0x73

Inhibit Time of EMCY (0x1015)

Table 11: 0x1015 object description

Attribute	Value
Index	0x1015
Name	Inhibit time EMCY
Object code	Variable
Data type	U16
Category	Optional

Table 12: 0x1015 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U16 (multiple of 100μs)
Default value	0x0000

Inhibit time is a minimum time between two consecutive emergency objects produced by node.

Producer Heartbeat Time (0x1017)

Table 13: 0x1017 object description

Attribute	Value
Index	0x1017
Name	Producer heartbeat time
Object code	Variable
Data type	U16
Category	Mandatory

Table 14: 0x1017 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U16 (multiple of 1ms)
Default value	0x03E8 (1000ms)

The value for heartbeat time is given in multiples of 1ms. The value 0 disables the producer heartbeat. Dynamic heartbeat time is affected instantly. But in order to keep configuration after reboot a save is needed. Node-ID is affected after reboot.

Device Identity (0x1018)

Table 15: 0x1018 object description

Attribute	Value
Index	0x1018
Name	Identity object
Object code	Record
Data type	Identity
Category	Mandatory

Table 16: 0x1018 entry description

Attribute	Value
Sub-index	0x00
Description	Highest sub-index supported
Access	RO
PDO mapping	No
Value range	0x01..0x04
Default value	0x04
Sub-index	0x01
Description	Vendor-ID
Access	RO
PDO mapping	No
Value range	U32
Default value	0x00000209 (TK Engineering Oy)
Sub-index	0x02
Description	Product code
Access	RO

PDO mapping	No
Value range	U32
Default value	0x00000FCB
Sub-index	0x03
Description	Revision number
Access	RO
PDO mapping	No
Value range	U32
Default value	0x0001002E (Depends on firmware version)
Sub-index	0x04
Description	Serial number
Access	RO
PDO mapping	No
Value range	U32
Default value	No

Table 17: Revision number structure

Byte3	Byte2	Byte1	Byte0
Major revision MSB	Major revision LSB	Minor revision MSB	Minor revision LSB

Hardware Configuration

Disable/Enable Heartbeat

The CAN switch has a configuration port that is using the CANopen stack specifications. This port uses to configure the node, and it also transmits heartbeat and EMCY messages. This port can disable and enable the heartbeat by writing to SDO object 0x2000. Writing 0 disables the heartbeat, writing 1 enables, which is default.

Table 18: 0x2000 object description

Attribute	**Value**
Index	0x2000
Name	Disable/Enable Heartbeat
Object code	Variable
Data type	U8
Category	Mandatory

Table 19: 0x2000 entry description

Attribute	**Value**
Sub-index	0x00
Access	rw
PDO mapping	No
Value range	U8
Default value	1 = Enabled (Default), 0 = Disabled

Bit Rates (0x50x0)

Table 20: 0x5010..0x5050 object description

Attribute	Value
Index	0x5010 (CAN1), 0x5020 (CAN2), 0x5030 (CAN3), 0x5040 (CAN4), 0x5050 (CANA)
Name	Port [1..4, A] Bit rate
Object code	Variable
Data type	U8
Category	Optional

Table 21: 0x5010..0x5050 sub 0x01 BTR0 entry description

Attribute	Value
Sub-index	0x01
Access	RW
PDO mapping	No
Value range	See Table 24 to Table 30
Default value	0x01 (500 kbps)

Table 22: 0x5010..0x5040 sub 0x02 BTR1 entry description

Attribute	Value
Sub-index	0x02
Access	RW
PDO mapping	No
Value range	See Table 24 to Table 30
Default value	0x3A (500 kbps)

Table 23: 0x5050 sub 0x02 BTR1 entry description

Attribute	Value
Sub-index	0x02
Access	RW
PDO mapping	No
Value range	See Table 24 to Table 30
Default value	0x1C (500 kbps)

These tables below about bit rates are examples and pre-calculated bit rates for the MSCAN controller and with oscillator clock 16 MHz. If you want some other sampling point and bit rates please then contact TK Engineering for re-calculation.

Table 24: Bit rate register enumeration for 1000kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BTR1
8	87,5	0x00	0x1C
8	75	0x00	0x3A
8	62,5	0x00	0x58

Table 25: Bit rate register enumeration for 800kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BRT1
20	85	0x00	0x2F
20	75	0x00	0x4D
20	65	0x00	0x6B

Table 26: Bit rate register enumeration for 500kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BRT1
16	87,5	0x01	0x1C
16	75	0x01	0x3A
16	62,5	0x01	0x58

Table 27: Bit rate register enumeration for 250kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BRT1
16	87,5	0x03	0x1C
16	75	0x03	0x3A
16	62,5	0x03	0x58

Table 28: Bit rate register enumeration for 125kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BRT1
16	87,5	0x07	0x1C
16	76	0x07	0x3A
16	62,5	0x07	0x58

Table 29: Bit rate register enumeration for 100kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BRT1
16	87,5	0x09	0x1C
16	75	0x09	0x3A
16	62,5	0x09	0x58

Table 30: Bit rate register enumeration for 50kbps

Nr Time quanta	Sample Point %	Register BTR0	Register BRT1
16	87,5	0x13	0x1C
16	75	0x13	0x3A
16	62,5	0x13	0x58

The new settings will apply after made a “Save configuration” request and when made a software/hardware reboot.

Acceptance Filter Configurations First Filter Bank (0x50x1)

Table 31: 0x50x1 object description

Attribute	**Value**
Index	0x5011 (CAN1), 0x5021 (CAN2), 0x5031 (CAN3), 0x5041 (CAN4)
Name	Port [1..4] acceptance filter configuration
Object code	Variable
Data type	U32
Category	Optional

Table 32: 0x50n1 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U32, see Table 33
Default value	0xFFFFFFFF

HW acceptance filter is unsigned 32. See ref /2/ chapter 13.3.2.17 data sheet for details. The values from the CANopen telegram are inserted into the register as follow:

Table 33: Acceptance register mapping

Data Byte	Acceptance Registers
Byte[0]	CANIDAR3
Byte[1]	CANIDAR2
Byte[2]	CANIDAR1
Byte[3]	CANIDAR0

NOTE: The hardware and software use both banks. The new settings will apply after made a “Save configuration” request and when made a software/hardware reboot.

Acceptance Filter Configurations Second Filter Bank (0x50x2)

Table 34: 0x50x2 object description

Attribute	Value
Index	0x5012 (CAN1), 0x5022 (CAN2), 0x5032 (CAN3), 0x5042 (CAN4)
Name	Port [1..4] acceptance filter configuration
Object code	Variable
Data type	U32
Category	Optional

Table 35: 0x50x2 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U32, see Table 36
Default value	0xFFFFFFFF

HW acceptance filter is unsigned 32. See ref /2/ chapter 13.3.2.17 data sheet for details. The values from the CANopen telegram are inserted into the register as follow:

Table 36: Acceptance register mapping

Data Byte	Acceptance Registers
Byte[0]	CANIDAR3
Byte[1]	CANIDAR2
Byte[2]	CANIDAR1
Byte[3]	CANIDAR0

NOTE: The hardware and software use both banks. The new settings will apply after made a “Save configuration” request and when made a software/hardware reboot.

Acceptance Mask Configurations First Filter Bank (0x50x5)

Table 37: 0x50x5 object description

Attribute	Value
Index	0x5015 (CAN1), 0x5025 (CAN2), 0x5035 (CAN3), 0x5045 (CAN4)
Name	Port [1..4] acceptance mask configuration
Object code	Variable
Data type	U32
Category	Optional

Table 38: 0x50x5 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U32, see Table 39: Acceptance mask register mapping
Default value	0xFFFFFFFF (Don’t care all)

HW acceptance mask is unsigned 32. See ref /2/ chapter 13.3.2.18 data sheet for details. The values from the CANopen telegram are inserted into the register as follow:

Table 39: Acceptance mask register mapping

Data Byte	Acceptance Registers
Byte[0]	CANIDMR3
Byte[1]	CANIDMR2
Byte[2]	CANIDMR1
Byte[3]	CANIDMR0

NOTE: The hardware and software use both banks, and the software is not programmed to work with extended identifiers. The new settings will apply after made a “Save configuration” request and when made a software/hardware reboot.

Acceptance Mask Configurations Second Filter Bank (0x50x6)

Table 40: 0x50x6 object description

Attribute	Value
Index	0x5016 (CAN1), 0x5026 (CAN2), 0x5036 (CAN3), 0x5046 (CAN4)
Name	Port [1..4] acceptance mask configuration
Object code	Variable
Data type	U32
Category	Optional

Table 41: 0x50x6 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U32, see Table 42
Default value	0xFFFFFFFF (Don’t care all)

HW acceptance mask is unsigned 32. See ref /2/ chapter 13.3.2.18 data sheet for details. The values from the CANopen telegram are inserted into the register as follow:

Table 42: Acceptance mask register mapping

Data Byte	Acceptance Registers
Byte[0]	CANIDMR3
Byte[1]	CANIDMR2
Byte[2]	CANIDMR1
Byte[3]	CANIDMR0

NOTE: The hardware and software use both banks. The new settings will apply after made a “Save configuration” request and when made a software/hardware reboot.

Hardware Filter Configuration (0x50x9)

There are 4 different filter configurations which the CAN switch supports. More detailed information is found in reference /2/ Chapter 13.3.2.12. Filter configuration changes register CANIDAC fields IDAM0-IDAM1.

Table 43: 0x50x9 object description

Attribute	Value
Index	0x5019 (CAN1), 0x5029 (CAN2), 0x5039 (CAN3), 0x5049 (CAN4)
Name3.5	Port [1..4] filter configuration register
Object code	Variable
Data type	U8
Category	Optional

Table 44: 0x50x9 entry description

Attribute	Value
Sub-index	0x00
Access	RW
PDO mapping	No
Value range	U8, see Table 45
Default value	0x00

Table 45: HW filter configuration description

Value	Description
0x00	Two 32-bit registers
0x10	Four 16-bit registers
0x20	Eight 8-bit registers
0x30	Filter closed

The new settings will apply after made a “Save configuration” request and when made a software/hardware reboot.

Identifier Acceptance Filter

By the nature of the CAN controller, filtering happens after messages received on the bus are read into the receive buffer. Since messages that are internally routed from another port are not received this way, filtering does not apply to such messages.

On reception, each message is checked to see whether it passes the identifier acceptance filter. The identifier acceptance registers define the acceptable patterns of the standard or extended identifier. If more than one hit occurs (two or more filters match), the lower hit has priority. The filter is programmable to operate in four different modes:

Two 32-bit filters, each to be applied to:

- The full 29 bits of the extended identifier, plus the RTR (Remote transmission request), IDE (Identifier extension) and SRR (Substitute remote request) bits.
- The 11 bits of the standard identifier, plus the RTR and IDE bits. It is however recommended to use the four or eight identifier acceptance filters for standard identifiers.

Four 16-bit filters, each to be applied to:

- The 14 most significant bits of the extended identifier, plus the SRR and IDE bits.
- The 11 bits of the standard identifier, plus the RTR and IDE bits.

Eight 8-bit filters, each to be applied to:

- The 8 most significant bits of the extended identifier.
- The 8 most significant bits of the standard identifier.

Closed filter:

- No CAN messages are seen by the application.

Filtering Examples

The next few sections provide examples of how to configure the hardware filters. Read the sections in sequence, as later examples expand on those introduced earlier.

Filtering Using Two 32-bit Filters and Standard IDs

The figure below shows the function of the first of the two per-channel 32-bit filters (filter banks) when using standard CAN IDs. For a filter to produce a “hit”, the ID in the CAN controller identifier register of the receive message buffer is masked with the value in the CAN controller acceptance mask register and compared to the value in the CAN controller acceptance register. A “1” in CAN controller acceptance mask register means “don’t care.”

The identifier registers for a standard format identifier consist of a total of 13 bits; ID[10:0], RTR, and IDE bits. The IDE bit in the ID register is always zero. Only the first two acceptance and mask registers are applied (numbered 0 and 1 in the figure). To receive standard identifiers in 32 bit filter mode, it is required to program the last three bits (AM[2:0]) in the mask register CANIDMR1 to “don’t care.”

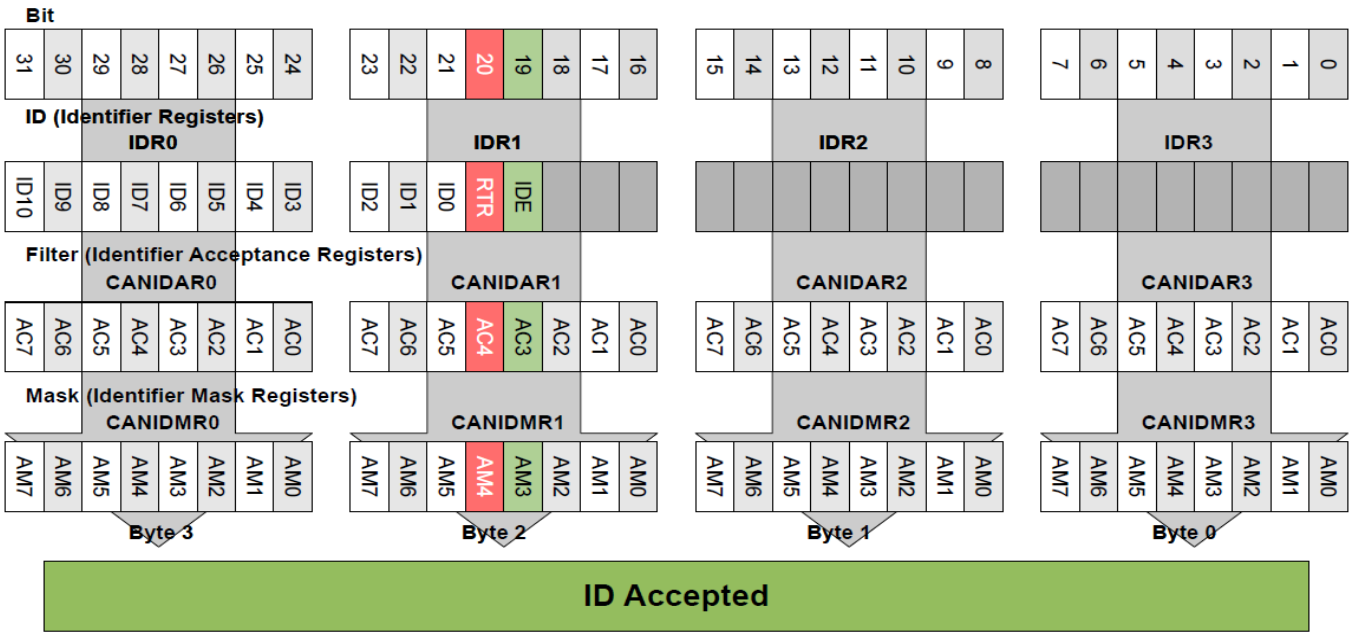


Figure 1: 32-bit filter using standard IDs

Example 1 – Accepting Standard IDs 0x1 and 0x2

Configuring a filter to accept standard ID 0x1 can be visualised as follows, where the ID has been left-shifted by 21 in order to align correctly with the ID register. As noted earlier, the **mask bits [18:16] must be set to 1 “don’t care”** when working with standard IDs and 32-bit filters. Note that the microcontroller data sheet does not explicitly state that bits [15:0] of the mask must be set to zero.

```
RTR (bit 20)
IDE (bit 19)

id (0x1<<21):   = 00000000 00100000 00000000 00000000 = 0x00200000
filter 1:       = 00000000 00100000 00000000 00000000 = 0x00200000
mask 1:         = 11111111 11000111 00000000 00000000 = 0xFFC70000
```

*For the mask, 0: must match, 1: don’t care

Since bit 21 of the mask is set to 0 (must match), bit 21 in the ID must match that of bit 21 in the filter in order for the ID to be accepted.

In this case however, bits [31:22] of the mask have been set to 1 (don’t care), meaning that any ID with a least significant bit set to 1 (i.e. every odd-numbered ID) will be accepted as well. Setting bits [31:22] to 0 instead would result in a filter that only accepts ID 0x1 and nothing else.

Note however that if filter 2 is left in its default state, IDs blocked by filter 1 will leak through filter 2 instead, so either apply the same configuration to both filters, or configure filter 2 to “block all”.

We can configure a CAN port to use two 32-bit filters, with filter 1 only accepting ID 0x1, and filter 2 only accepting ID 0x2 as follows:

```
0x01 << 21:     = 00000000 00100000 00000000 00000000 = 0x00200000
filter 1:       = 00000000 00100000 00000000 00000000 = 0x00200000
mask 1:         = 00000000 00000111 00000000 00000000 = 0x00070000

0x02 << 21:     = 00000000 01000000 00000000 00000000 = 0x00400000
filter 2:       = 00000000 01000000 00000000 00000000 = 0x00400000
mask 2:         = 00000000 00000111 00000000 00000000 = 0x00070000
```

This configuration is applied to port CAN1 by configuring the object dictionary as follows:

Table 46: 32-bit filter standard ID example

Index	Value
0x5011 (HW accept filter CAN1)	0x200000 (0x1 << 21)
0x5012 (HW accept filter2 CAN1)	0x400000 (0x2 << 21)
0x5015 (HW accept mask CAN1)	0x70000 (All bits must match)
0x5016 (HW accept mask2 CAN1)	0x70000 (All bits must match)
0x5019 (HW filter conf CAN1)	0x00 (Two 32-bit filters)

The settings are however more easily applied using the Custom GUI feature in CANopen ConfTool:

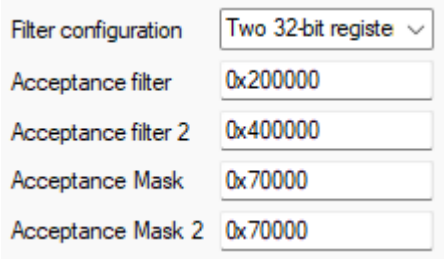


Figure 2: Applying filter settings in CANopen ConfTool

The new settings will take effect after making a “Save configuration” request followed by a software or hardware reboot.

Example 2 – Blocking Standard ID 0x1

Blocking ID 0x1 can be done by requiring that the bit in position 21 is zero by setting the filter and mask accordingly.

```
0x01 << 21:   = 00000000 00100000 00000000 00000000 = 0x00200000
filter 1:     = 00000000 00000000 00000000 00000000 = 0x00000000
mask 1:       = 11111111 11000111 00000000 00000000 = 0xFFC70000
```

*For the mask, 0: must match, 1: don’t care

The drawback is that now any ID with a least significant bit set to 1 (i.e. every odd-numbered ID) gets blocked, which is probably not what was intended.

In case all the IDs that will be present on the bus are known, it may be worth checking for similarities in the bit patterns of these IDs. As an example, assume that we want to block ID 0x1 but let the following IDs pass:

```
0x70B << 21: = 11100001 01100000 00000000 00000000 = 0xE1600000
0x115 << 21: = 00100010 10100000 00000000 00000000 = 0x22A00000
0x116 << 21: = 00100010 11000000 00000000 00000000 = 0x22C00000
0x702 << 21: = 11100000 01000000 00000000 00000000 = 0xE0400000
0x701 << 21: = 11100000 00100000 00000000 00000000 = 0xE0200000
0x194 << 21: = 00110010 10000000 00000000 00000000 = 0x32800000
0x8B  << 21: = 00010001 01100000 00000000 00000000 = 0x11600000
```

In this case we can see that bit 29 (the third bit from the left) is set for the first six IDs, whereas bit 28 is set for the seventh ID. This means we can configure the first filter to require that bit 29 is set, and the second filter to require that bit 28 is set. Both filters will block ID 0x1, as well as any other ID that does not fulfil either requirement.

```
filter 1:     = 00100000 00000000 00000000 00000000 = 0x20000000
mask 1:       = 11011111 11100111 00000000 00000000 = 0xDFE70000

filter 2:     = 00010000 00000000 00000000 00000000 = 0x10000000
mask 2:       = 11101111 11100111 00000000 00000000 = 0xEFE70000
```

If more than two such bit pattern matches are needed, consider using four 16-bit filters instead.

Example 3 – Using four 16-bit filters

When using 32-bit filtering for standard IDs, only two out of four filter and mask registers (CANIDAR0/1, CANIDMR0/1) are applied. With 16-bit filtering, the unused registers (CANIDAR2/3, CANIDMR2/3) are put to work, doubling the amount of IDs that can be matched. The figure below shows the first filter bank (i.e. two 16-bit filters).

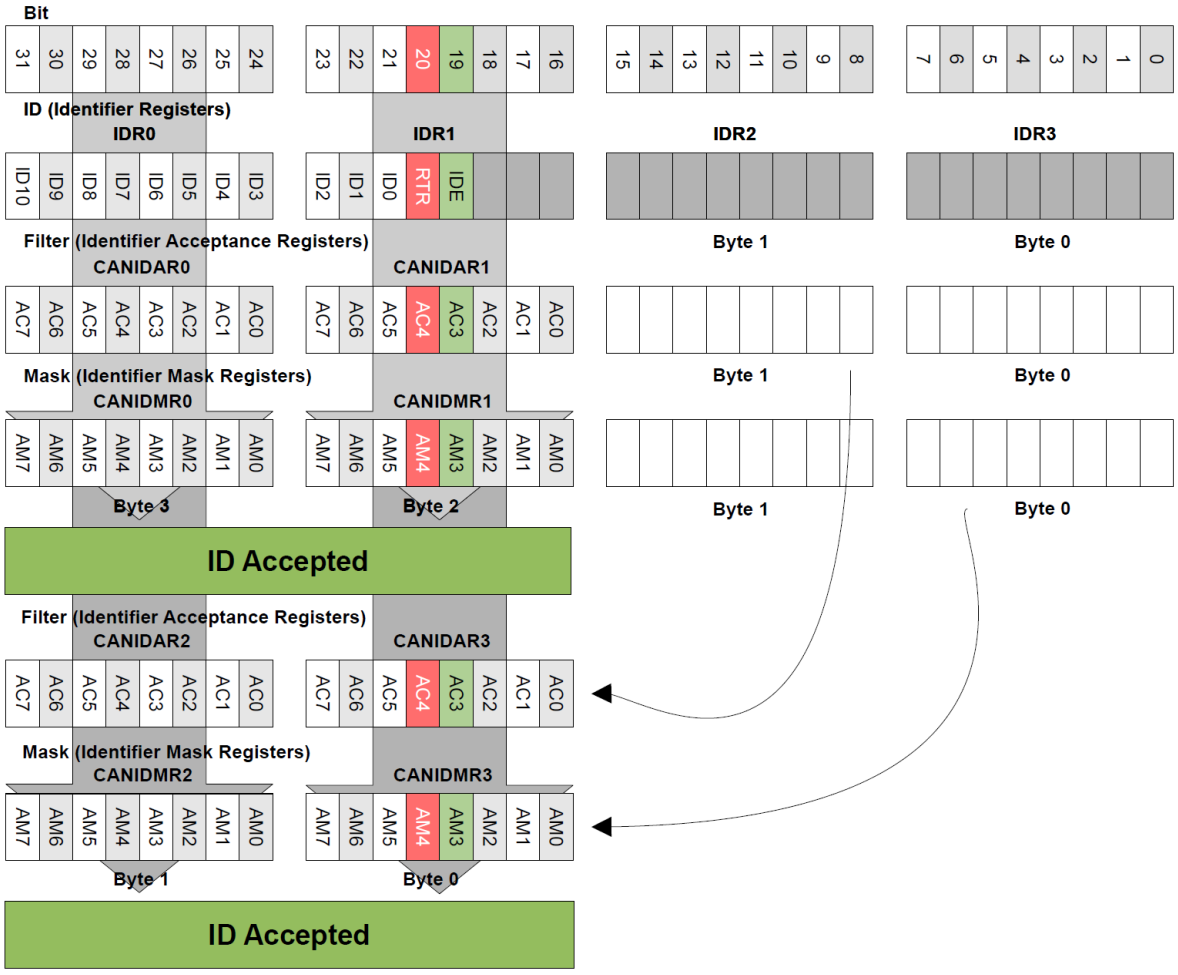


Figure 3: Using for 16-bit filters for standard identifiers

Continuing from the previous example, it will probably be easier to reason about the filter settings if we drop the last two bytes from the ID register, and left-shift the ID by 5 instead of 21. Let's also add two more IDs that we want passing the filter; 0x2C and 0x7, at the end to have a reason for using 16-bit filtering, while still blocking ID 0x1.

```
RTR (bit 20)
IDE (bit 19)

0x70B << 5:  = 11100001 01100000 = 0xE160
0x115 << 5:  = 00100010 10100000 = 0x22A0
0x116 << 5:  = 00100010 11000000 = 0x22C0
0x702 << 5:  = 11100000 01000000 = 0xE040
0x701 << 5:  = 11100000 00100000 = 0xE020
0x194 << 5:  = 00110010 10000000 = 0x3280
0x8B  << 5:  = 00010001 01100000 = 0x1160
0x2C  << 5:  = 00000101 10000000 = 0x0580
0x7   << 5:  = 00000000 11100000 = 0x00E0
```

Filter 1 can now also do what filter 2 did in the previous example, and filter 2 can instead take care of the two new IDs.

```
filter 1.1:  = 00100000 00000000 = 0x2000
filter 1.2:  = 00010000 00000000 = 0x1000
mask 1.1:    = 11011111 11100111 = 0xDFE7
mask 1.2:    = 11101111 11100111 = 0xEFE7

filter 2.1:  = 00000101 10000000 = 0x0580
filter 2.2:  = 00000000 11100000 = 0x00E0
mask 2.1:    = 00000000 00000111 = 0x0007
mask 2.2:    = 00000000 00000111 = 0x0007
```

*For the mask, 0: must match, 1: don't care

Combining these 16-bit filters and masks into 32-bit register values, we get:

```
filter 1:    = 00100000 00000000 00010000 00000000 = 0x20001000
mask 1:      = 11011111 11100111 11101111 11100111 = 0xDFE7EFE7

filter 2:    = 00000101 10000000 00000000 11100000 = 0x058000E0
mask 2:      = 00000000 00000111 00000000 00000111 = 0x00070007
```

This configuration is applied to port CAN1 by configuring the object dictionary as follows:

Table 47: 16-bit filter standard ID example

Index	Value
0x5011 (HW accept filter CAN1)	0x20001000
0x5012 (HW accept filter2 CAN1)	0x058000E0
0x5015 (HW accept mask CAN1)	0xDFE7EFE7
0x5016 (HW accept mask2 CAN1)	0x00070007 (All bits must match)
0x5019 (HW filter conf CAN1)	0x10 (Four 16-bit registers)

The settings are however more easily applied using the Custom GUI feature in CANopen ConfTool:

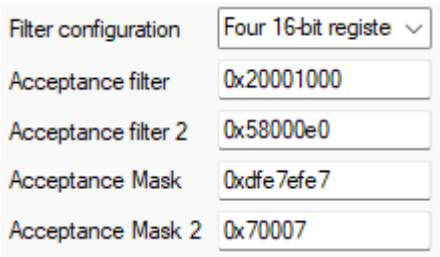


Figure 4: Applying filter settings in CANopen ConfTool

The new settings will take effect after making a “Save configuration” request followed by a software or hardware reboot.

Example 4 – Using four 16-bit filters for filtering ranges

Let us assume that we want to allow the following IDs: 0x700-0x707, 0x70C-70F, 0x501-0x5F1, and 0x5FF. Note the limitations of this method due to how binary numbers work: we cannot allow a range 0x70A-0x70F as this would only allow 0x70A, 0x70B, 0x70E and 0x70F, while skipping 0x70C and 0x70D.

```
0x700-0x707:   = 11100000 XXX00000
filter 1.1:    = 11100000 00000000 = 0xE000
mask 1.1:      = 00000000 11100111 = 0x00E7

0x70C-0x70F:   = 11100001 1XX00000
filter 1.2:    = 11100001 10000000 = 0xE180
mask 1.2:      = 00000000 01100111 = 0x0067

0x501-0x5F1:   = 101XXXX0 00100000
filter 2.1:    = 10100000 00100000 = 0xA020
mask 2.1:      = 00011110 00000111 = 0x1E07

0x5FF:         = 10111111 11100000
filter 2.2:    = 10111111 11100000 = 0xBFE0
mask 2.2:      = 00000000 00000111 = 0x0007


filter 1:     = 11100000 00000000 11100001 10000000 = 0xE000E180
mask 1:       = 00000000 11100111 00000000 01100111 = 0x00E70067
filter 2:     = 10100000 00100000 10111111 11100000 = 0xA020BFE0
mask 2:       = 00011110 00000111 00000000 00000111 = 0x1E070007
```

Table 48: 16-bit filter standard ID example

Index	Value
0x5011 (HW accept filter CAN1)	0xE000E180
0x5012 (HW accept filter2 CAN1)	0xA020BFE0
0x5015 (HW accept mask CAN1)	0x00E70067
0x5016 (HW accept mask2 CAN1)	0x1E070007
0x5019 (HW filter conf CAN1)	0x10 (Four 16-bit registers)

Filtering Using Two 32-bit Filters and Extended IDs

The identifier registers for an extended format identifier consist of a total of 32 bits; ID[28:0], SRR, IDE, and RTR bits. For extended identifiers, the SRR and IDE bits in the ID register are always set.

The figure below shows the function of the first of the two per-channel 32-bit filters when using extended CAN IDs. For a filter to produce a “hit”, the ID in the CAN controller identifier register of the receive message buffer is masked with the value in the CAN controller acceptance mask register and compared to the value in the CAN controller acceptance register. A “1” in CAN controller acceptance mask register means “don’t care.”

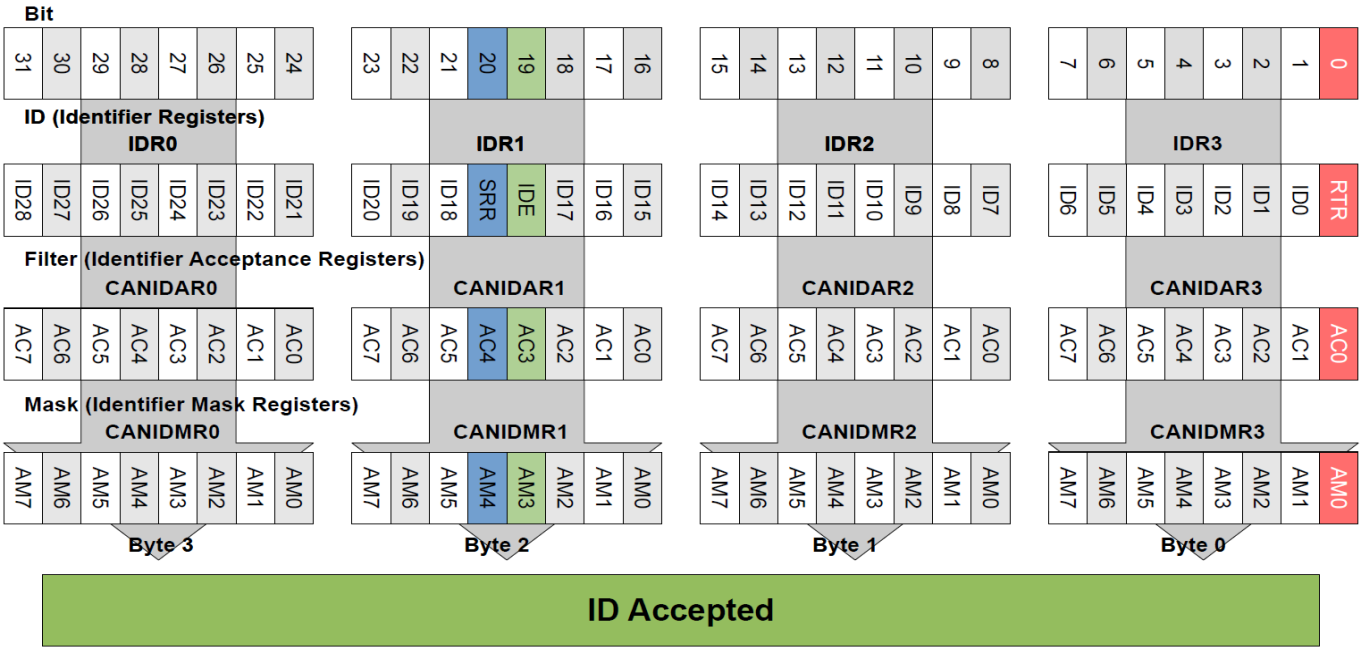


Figure 5: 32-bit filter using extended IDs

Example 1 – Accepting Extended IDs 0x1 and 0x2

Configuring a filter to accept extended ID 0x1 can be visualised as follows, where the ID has been left-shifted by 1 in order to align correctly with the ID register. Note that **the SRR and IDE bits are implicitly set for any received extended ID**, so the resulting ID is effectively (ID << 1) | 0x180000.

```
SRR (bit 20)
IDE (bit 19)
RTR (bit 0)

0x01: →        = 00000000 00011000 00000000 0000001 0 = 0x00180002
filter 1:      = 00000000 00011000 00000000 0000001 0 = 0x00180002
mask 1:        = 00000000 00000000 00000000 0000000 0 = 0x00000000

0x02: →        = 00000000 00011000 00000000 0000010 0 = 0x00180004
filter 2:      = 00000000 00011000 00000000 0000010 0 = 0x00180004
mask 2:        = 00000000 00000000 00000000 0000000 0 = 0x00000000
```

*For the mask, 0: must match, 1: don’t care

To configure port CAN1 to use two 32-bit filters, with filter one only accepting an extended ID with CAN ID 0x1, and filter two only accepting extended CAN ID 0x2, the following object dictionary values are needed:

Table 49: 32-bit filter extended ID example

Index	Value
0x5011	0x180002
0x5012	0x180004
0x5015	0x0 (All bits must match)
0x5016	0x0 (All bits must match)
0x5019	0x0 (Two 32-bit filters)

The new settings will take effect after making a “Save configuration” request followed by a software or hardware reboot.

Example 2 – Accepting J1939 Extended IDs

Configuring the filters to accept an ID such as those found in the J1939 protocol is complicated by the fact that the 29 ID bits [28:0] occupy bits [18:1] and [31:21] in the ID register.

As a quick example just to illustrate the concept, let us assume that all bits in the ID are 1, resulting in a value of 0x1FFFFFFF:

```
                    00011111 11111111 11111111 11111111 = 0x1FFFFFFF
```

We can take the first 18 LSBs and left-shift by 1 to skip over the RTR bit:

```
SRR (bit 20)
IDE (bit 19)
RTR (bit 0)

(0x1FFFFFFF & 0x3FFF) << 1
                  = 00000000 00000111 11111111 11111110 = 0x7FFFE
```

Next, we take the remaining bits 19 to 29 of the ID and left-shift by 1 + 2 to skip over the SRR and IDE bits:

```
(0x1FFFFFFF & 0x1FFC0000) << 3
                  = 11111111 11100000 00000000 00000000 = 0xFFE00000
```

Doing a bitwise-OR gets us:

```
0x7FFFE | 0xFFE00000
                  = 11111111 11100111 11111111 11111110 = 0xFFE7FFFE
```

Finally, we set the mandatory bits 19 and 20:

```
0xFFE7FFFE | 0x180000
                  = 11111111 11111111 11111111 11111110 = 0xFFFFFFFE
```

Assume that we want to configure the filters to allow IDs 0x18FEC1EE and 0x1CFECA03. Using the steps outlined above, we create a filter value for ID 0x18FEC1EE as follows:

```
                    00011000 11111110 11000001 11101110 = 0x18FEC1EE

(0x18FEC1EE & 0x3FFF) << 1
                  = 00000000 00000101 10000011 11011100 = 0x583DC
(0x18FEC1EE & 0x1FFC0000) << 3
                  = 11000111 11100000 00000000 00000000 = 0xC7E00000
0x583DC | 0xC7E00000 | 0x180000
                  = 11000111 11111101 10000011 11011100 = 0xC7FD83DC
```

Doing the corresponding calculations for ID 0x1CFECA03 results in:

```
0x59406 | 0xE7E00000 | 0x180000
                  = 11100111 11111101 10010100 00000110 = 0xE7FD9406
```

The filter configuration then becomes:

```
0x18FEC1EE: →   = 11000111 11111101 10000011 11011100 = 0xC7FD83DC
filter 1:       = 11000111 11111101 10000011 11011100 = 0xC7FD83DC
mask 1:         = 00000000 00000000 00000000 00000000 = 0x00000000
```

```
0x1CFECA03: →  = 11100111 11111101 10010100 0000011 0 = 0xE7FD9406
filter 2:      = 11100111 11111101 10010100 0000011 0 = 0xE7FD9406
mask 2:        = 00000000 000 00000 00000000 0000000 0 = 0x00000000
```

*For the mask, 0: must match, 1: don't care

This configuration is applied to port CAN1 by configuring the object dictionary as follows:

Table 50: 32-bit filter extended ID example

Index	Value
0x5011	0xC7FD83DC
0x5012	0xE7FD9406
0x5015	0x0 (All bits must match)
0x5016	0x0 (All bits must match)
0x5019	0x0 (Two 32-bit filters)

Again, these settings are most easily applied using the Custom GUI feature in CANopen ConfTool:

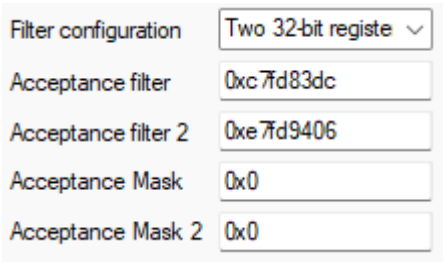


Figure 6: Applying filter settings in CANopen ConfTool

The new settings will take effect after making a "Save configuration" request followed by a software or hardware reboot.

Universal ID Routing Settings

This is used if you want all can-id to have the same routing settings. If value of routing settings is something else than 0x0000 then this routing settings will be used and the routing table 0x6801 to 0x6864 will be ignored.

Table 51: Universal ID routing settings

Attribute	Value
Index	0x6800
Name	Universal ID routing
Object code	Variable
Data type	U16
Category	Optional

Table 52: 0x6800 sub 1 entry description CAN ID

Attribute	Value
Sub-index	0x01
Description	CAN id NOTE! This ID will be ignored
Access	RW
PDO mapping	No
Value range	U32
Default value	0x00000000

Table 53: 0x6800 sub2 entry description Route

Attribute	Value
Sub-index	0x02
Description	Route description
Access	RW
PDO mapping	No
Value range	U16, see Table 57
Default value	0x7BDE (for all entries)

Route Table Configuration (0x6801..0x6864)

If object dictionary index 0x6800 sub index 2 is equal to 0x0000 then this routing table of 100 CAN-ID will be used for routing.

Table 54: Route table Configuration

Attribute	Value
Index	0x6801..0x6864
Name	Route table
Object code	Variable
Data type	U16
Category	Optional

Table 55: 0x6801..0x6864 entry description CAN-ID

Attribute	Value
Sub-index	0x01
Description	CAN id
Access	RW
PDO mapping	No
Value range	U32
Default value	0x00000000

NOTE: Standard id 0x3f and extended id 0x3f are handled separately, to define an extended id, bit 30 in ID field must be set to 1. To define a remote id, bit 31 in ID field must be set.

Table 56: 0x6801..0x6864 sub2 entry description Route

Attribute	Value
Sub-index	0x02
Description	Route description
Access	RW
PDO mapping	No
Value range	U16, see Table 57
Default value	0x0000 (for all entries)

Table 57: Route descriptor structure

Bit15 (MSb)												Bit0 (LSb)			
Route descriptor entry for CAN-ID															
Nibble3				Nibble2				Nibble1				Nibble0			
Target flag set to message received from CAN4				Target flag set to message received from CAN3				Target flag set to message received from CAN2				Target flag set to message received from CAN1			
Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0: Not forwarded to CAN4 1: Forwarded to CAN4	0: Not forwarded to CAN3 1: Forwarded to CAN3	0: Not forwarded to CAN2 1: Forwarded to CAN2	0: Not forwarded to CAN1 1: Forwarded to CAN1	0: Not forwarded to CAN4 1: Forwarded to CAN4	0: Not forwarded to CAN3 1: Forwarded to CAN3	0: Not forwarded to CAN2 1: Forwarded to CAN2	0: Not forwarded to CAN1 1: Forwarded to CAN1	0: Not forwarded to CAN4 1: Forwarded to CAN4	0: Not forwarded to CAN3 1: Forwarded to CAN3	0: Not forwarded to CAN2 1: Forwarded to CAN2	0: Not forwarded to CAN1 1: Forwarded to CAN1	0: Not forwarded to CAN4 1: Forwarded to CAN4	0: Not forwarded to CAN3 1: Forwarded to CAN3	0: Not forwarded to CAN2 1: Forwarded to CAN2	0: Not forwarded to CAN1 1: Forwarded to CAN1

The switch software is capable of handle all telegrams having 11-bit (standard) or 29-bit (extended) CAN-IDs to be forwarded to one or more interfaces or to be discarded. Route table consists of a Word (16bit) array, length 100. Each Word correspond to a table index between 0 to 100.

Example 1:

0x6801:0x01 = 0x3f: (array index 0)

0x6801:0x02 = 0x0356: (array index 0)

Where the meaning of 0x0356 is:

- 0: Messages with ID 0x3f received from CAN4 are discarded.
- 3: Messages with ID 0x3f received from CAN3 are copied to CAN1 and CAN2.
- 5: Messages with ID 0x3f received from CAN2 are copied to CAN1 and CAN3.
- 6: Messages with ID 0x3f received from CAN1 are copied to CAN2 and CAN3.

Example 2:

0x6802:0x01 = 0x2000003f: (array index 0)

0x6802:0x02 = 0x7BDE: (array index 0)

Where the meaning of 0x7BDE is:

- 7: Messages with EXT ID 0x3f received from CAN4 are copied to CAN1, 2 and 3
- B: Messages with EXT ID 0x3f received from CAN3 are copied to CAN1, 2 and 4
- D: Messages with EXT ID 0x3f received from CAN2 are copied to CAN1, 3 and 4
- E: Messages with EXT ID 0x3f received from CAN1 are copied to CAN2, 3 and 4

Row 1 (CAN Channel) defines from which channel (CAN-Interface) the telegram is received. Row 2 defines to what channel(s) the telegram is to be forwarded. The forwarding method is always ASAP.

Route table configuration is done bye CANopen SDO. OD-index starts from 0x6801, which corresponds to array index 1.

The new settings will apply directly. In order to remain same route configuration after reboot, a “Save configuration” request is needed.

Standard id 0x3f and extended id 0x3f are handled separately, to define a extended id, bit 30 in ID field must be set. To define a remote id, bit 31 in ID field must be set.

Error Messages

The error messages (EMCY) are using CANopen protocol as template. Software is producing tree types of error messages for each CAN interface. The error messages are defined in table below.

Table 58: Error message descriptions

Error code	Error message	Description
0x8119	TX Overrun	This error message occurs when target CAN channel has dropped telegrams, because of overrun. Overrun is when 3-stage HW buffer and 16-stage SW buffer is full.
0x8111	RX Overrun	This error message occurs when the receiving CAN channel 5-stage HW buffer is full.
0x0000	Error-Active	Sent after a error state is cleared and transmission is working without errors.
0x8120	Error-Warning	RX or TX counter >96 and ≤ 127
0x8121	Error-Passive	RX or TX counter >127 and ≤ 255
0x8122	Bus-off	RX and TX counter > 255

In table below the bytes in CANopen telegram are defined.

Table 59: Error message structure

Byte	Description
0	Error code LSB
1	Error code MSB
2	Error-register object (0x1001)
3	Target CAN interface error occurred
4	Error counter, counter of how many times the error occurred LSB
5	Error counter, MSB
6	not used
7	not used

Frequently Asked Questions (FAQ)

Does the CAN Switch need to be configured before use?

The default configuration of the CAN Switch is to forward all CAN messages on CAN ports 1-4 using the bitrate 500kbit/s. If this is compatible with your network then no configuration is needed. See the chapter *Factory Settings* for the complete default configuration.

The CAN Switch can be ordered pre-configured for your network. In that case no further configuration is needed.

Legal Information

About This Manual

Trademarks

Revision History

The following revision history table summarizes changes contained in this document.

Date	Revision	Author	Description
2019-04-24	1.0	TO	Created document from TCS-10 and WCS-10 guide
2024-02-08	1.1	CS	Fixed default values for 0x5010...0x5040 sub 0x02
2025-10-13	1.2	CS	Added hardware filtering examples
2025-11-05	1.3	TR	In Factory Settings, fixed EMCY inhibit time value.

References

/2/ MC9S12XDP512 Data Sheet, Rev. 2.11, Freescale Semiconductor

/3/ CiA301, CiA Draft Standard Proposal 301, V4.1

/4/ Embedded Networking with CAN and CANopen, Pfeiffer, Ayre and Keydel

/5/ CiA302-1, CiA Addition application layer functions, V3.4.1

/6/ CiA Draft Recommendation 303-3 Indicator specification V1.2