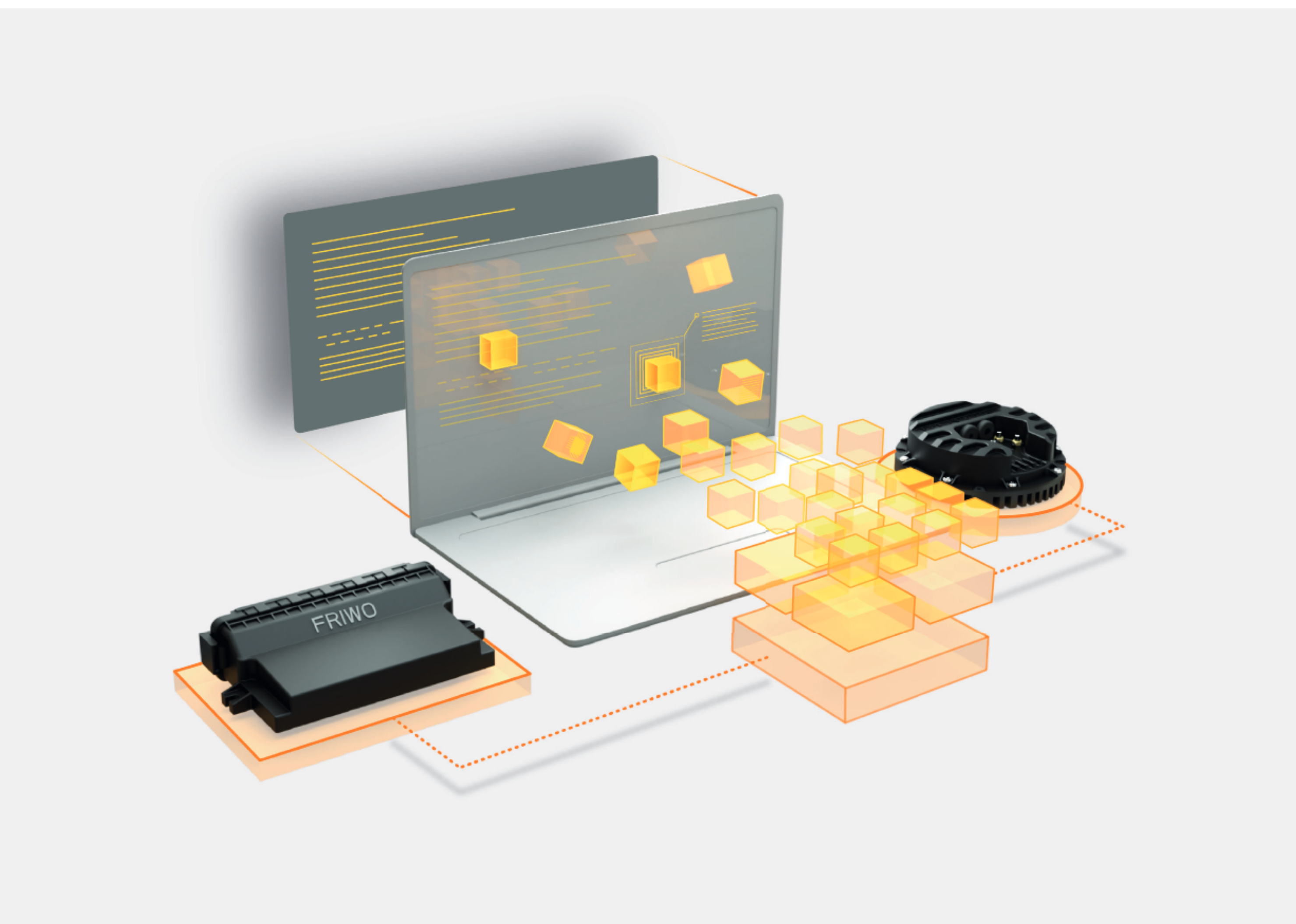
FRIWO | SDK

SOFTWARE DEVELOPMENT KIT

End-to-end Development Environment Setup Solution

MODULE DESCRIPTION



OVERVIEW

1. **Desired Torque (TRQ_DES) Module**
   Function to calculate and define the desired or requested torque from the user to the motor.

2. **CAN Module**
   Communication module that provides customized CAN-Bus Messaging. CAN Bus is industry standard the leading automotive real-time communication protocol.

1. DESIRED TORQUE (TRQ_DES)

1.1 DESCRIPTION

This module is the first stage of torque calculation. It utilizes the mapped input signals of throttle and brake and calculates the torque request which is desired by the driver. In the easiest case, this can be done by simply adding the signals up in order to get a resulting torque, depending on the operation mode. For special driving maneuvers, e.g. hill assistance, additional motor related information such as the rotor speed can be used to prioritize either the throttle or the brake input.

Additionally, different maximum torque gradients could be represented for each ride mode, which evoke the driver's feeling to be more comfy or rather sporty in acceleration. After the desired torque has been calculated, the output of this module is directly handed over to the torque limitation (TRQ LIM) and torque strategy (TRQ STR) modules until the torque will finally be generated by the current controllers.

1.2 CONFIGURATION

The following table lists all configuration parameters for TRQ DES module:

Function	Datatype	Min	Max	Description
SDK_C_TRQDES_Custom_Module_Enable	UInt8	0	1	Switch between FRIWO default TRQ_DES module and custom module. 0: default 1: custom **Note:** Switching is only possible, if motor connected to MCU is in standstill. Default value: 0

If the custom module has been successfully implemented in firmware with FRIWO SDK, the module can be executed by setting the configuration parameter SDK_C_TRQDES_Custom_Module_Enable to 1 with FRIWO Enable Tool Application. When setting this configuration parameter to 0, the default module is executed. Because of security reasons, the switch between default and custom module is only possible during standstill of the motor connected to the control unit.

1.3 IMPORTANT API FUNCTIONS

Function	Description
trqdesApi_Get_*VariableName*	Get data from variable *VariableName* used in firmware, e.g. trqdesApi_Get_*APP_Disp_Ride_Mode* See Variable Description for a full list of available Get-variables. https://friwo.link/md/variables
trqdesApi_Set_*VariableName*	Set firmware variable *VariableName* with values from custom module, e.g. canApi_Set_*TRQ_DES_Trq_Req_Rel* See Variable Description for a full list of available Set-variables.

2. CAN

2.1 DESCRIPTION

This module is the interface between the motor controller firmware and the CAN bus. It processes the incoming CAN messages and provides the received information to the firmware. Information from the firmware can be cyclically placed on the bus in the form of CAN messages. With this module, the developer can specify the structure of the messages on the bus themselves and use their own messages and protocols.

The API of this module offers buffers for sending and receiving messages. These buffers can be read and written by the developer using the associated API functions. The base firmware independently takes care of the CAN peripheral.

Further API function and parameter descriptions can be found in the *canApi.h* header file.

2.2 IMPORTANT API FUNCTIONS /1

Function	Description
canApi_UserInitCallBack	User callback, which is called when the base firmware initializes the CAN peripheral. In this callback, the CAN buffer must be initialized and should be cleared. All required CAN filters for incoming messages should be set in this function. The developer can initialize his module code here.
canApi_UserPeriodicCallBack	This callback is called by the base firmware each millisecond. The developer must read the input buffer and forward received data to the firmware using the API functions. The complexity of this function should be kept as low as possible, as it runs in a high priority interrupt.
canApi_ClearTransmitBuffer	Remove all entries from the transceive buffer.
canApi_ClearReceiveBuffer	Remove all entries from the receive buffer.
canApi_SendMessage	Put a custom CAN frame into the transceive buffer. The base firmware will handle the transmission.
canApi_ReceiveMessage	Receive an incoming message.
canApi_Filter....	Filter functions to setup a filter bank for incoming messages.
canApi_FilterDeactivateFilterBank	Deactivate an active setting on a specific filter bank.

2.2 IMPORTANT API FUNCTIONS /2

Function	Description
canApi_Get_*VariableName*	Get data from variable VariableName used in firmware to be transceived via CAN bus, e.g. canApi_Get_*INFO_Voltage_DC* See Variable Description for a full list of available Get-variables. https://friwo.link/md/variables
canApi_Set_*VariableName*	Set firmware variable VariableName with data of received CAN frame, e.g. canApi_Set_*CAN_EXT_Ride_Mode* See Variable Description for a full list of available Set-variables.
canApi_Set_*VariableName***_Timeout**	Set timeout flag for corresponding firmware variable VariableName to handle CAN signal timeouts, e.g. canApi_Set_*CAN_EXT_Ride_Mode_Timeout* See Variable Description for a full list of available Timeout-flags.

2.3 BUFFER MODEL DESCRIPTION

The CAN module has configurable input and output buffers. The developer can write any CAN frames into the output buffer and read incoming messages from the input buffer. For this purpose, corresponding API functions are made available via the canApi. The base firmware takes care of sending and receiving the frames via the CAN peripheral. When starting the firmware, the developer must configure the buffers via the buffer setup function.

With the function parameters of the setup function, the developer can configure the behaviour of the buffers. The table below shows the available configurations:

Buffer Type	Description
Ringbuffer	Standard ringbuffer FIFO implementation. This implementation is strongly recommended if the developer intents to use a higher protocol with segmented block transfer (e.g. ISO 15765-2)
Prioritybuffer Queue	Every message has a *Priority* property. The message with the highest priority in the buffer is transmitted first. The priority of all messages that could not be sent in a cycle is increased by ‚1' at the end of each cycle. With this implementation, the user can prioritize important messages in his system. Incoming messages are not affected by prioritization (always priority ‚'1').
Prioritybuffer Replace	This implementation works just like the „Priority Queue" option with the difference that an existing message in a buffer is replaced if the new message for the buffer has the same CAN identifier. This is used to update existing data if it gets irrelevant if a new one is available (e.g. sensor data like temperatures). The priority value of the existing message is kept if it's priority is higher.

2.4 CAN BUS FILTER FUNCTIONALITY

The CAN periphery supports the filtering of incoming CAN frames in hardware. For this purpose, 27 filter banks are available to the developer. With these filters, either individual identifiers can be selectively allowed through (ListMode) or entire ranges can be permitted (MaskMode).

Each individual filter bank can hold up to four individual standard identifiers in ListMode and two identifiers with their corresponding masks in MaskMode. Consequently, a filter bank can hold two extended identifiers in List-Mode and one extended identifier in MaskMode.

In MaskMode, every incoming CAN frame is accepted whose identifier at the masked positions (logic 1 on corresponding bit position) is identical to the identifier of the filter.

The use of filters is strongly recommended to avoid overloading the CAN periphery and the microcontroller with irrelevant messages. A brief description for each filter setting function can be found in the *canApi.h* header file.

2.5 PROCEDURE FOR THE DEVELOPMENT OF AN INDIVIDUAL CAN BUS IMPLEMENTATION

1. Implement *canApi_UserInitCallBack()* function

- 1.1. Init buffers with *canApi_SetupBuffer(...)* function
- 1.2. Clear both buffers
- 1.3. Setup required CAN filters (see filter function descriptions)

2. Implement *canApi_UserPeriodicCallBack()* function

- 2.1. Read incoming messages from the input buffer with *canApi_ReceiveMessage(...)* function
- 2.2. Handle received messages (parse data and forward to firmware using *canApi_Set_* functions
- 2.3. Handle message timeouts → set timeouts using the corresponding *canApi_Set_..._Timeout* functions
- 2.4. Send periodic messages in desired interval, fill payload with data from the firmware using the *canApi_Get_* functions

3. Define configurable Parameters to control the behaviour of your implementation using the *Enable Tool.* (optional)

- 3.1. E.g. create parameters to control message sending interval and timeout values
- 3.2. E.g. use CAN bus and *Enable Tool* to create a logging functionality for external bus participants

2.6 TIMEOUT HANDLING

The timeout flags of the corresponding signals can be set by the function canApi_Set_*VariableName*_Timeout. All timeouts critical for firmware execution are merged to a 32 bit-codeword within the errorhandler. A list of these timeout flags with their corresponding 32 bit decimal and hex value as codeword implementation is shown in the table below:

Timeout flag	Codeword		Usage
	Decimal	Hex	
CAN_EXT_State_Request_Timeout	1	0x00000001	default
CAN_EXT_Torque_Request_Timeout	2	0x00000002	default
CAN_EXT_Reverse_Gear_Signal_Channel_Timeout	4	0x00000004	default
CAN_EXT_Alive_Counter_Timeout	8	0x00000008	default
CAN_EXT_ROC_Start_Timeout	16	0x00000010	default
CAN_EXT_Rotor_Speed_Max_Timeout	32	0x00000020	default
CAN_EXT_Skip_Signal_Checks_Timeout	64	0x00000040	default
CAN_EXT_Ride_Mode_Timeout	128	0x00000080	default
CAN_Immo_Unlock_Request_Timeout	256	0x00000100	default
CAN_BMS_SOC_Timeout	512	0x00000200	default
CAN_BMS_Fullcharge_Capacity_Timeout	1024	0x00000400	default
CAN_BMS_Max_Charge_Timeout	2048	0x00000800	default
CAN_BMS_Max_Discharge_Timeout	4096	0x00001000	default
CAN_BMS_Max_Voltage_Timeout	8192	0x00002000	default
CAN_BMS_Min_Voltage_Timeout	16384	0x00004000	default
CAN_BMS_PushButton_SuperLongPress_Ongoing_Timeout	32768	0x00008000	default
CAN_BMS_Pending_Bordnet_Shutdown_Timeout	65536	0x00010000	default
CAN_BMS_Pending_HV_Shutdown_Timeout	131072	0x00020000	default
CAN_Custom_Timeout_Bit27	2^{27}	0x08000000	individual
CAN_Custom_Timeout_Bit28	2^{28}	0x10000000	individual
CAN_Custom_Timeout_Bit29	2^{29}	0x20000000	individual
CAN_Custom_Timeout_Bit30	2^{30}	0x40000000	individual
CAN_Custom_Timeout_Bit31	2^{31}	0x80000000	individual

The last bits 27 to 31 of the timeout codeword can be set individually, while the rest is already reserved for default timeout flags.

The 32 bit-codeword of timeouts is masked bitwise by an errorcode-filter and a warningcode-filter each resulting in an errorcode (ERR_CAN_Timeout_Errorcode) and a warningcode (ERR_CAN_Timeout_Warningcode). The difference between both codes is that a non-zero errorcode leads to a system error and powerstage shutdown, if the parameter ERR_C_CAN_Timeout_Enable is set to 1. In contrast, the warningcode only displays timeouts of specific signals and has no further impact.

Note: Per default, the 32 bit-codeword of timeouts is masked by the constant decimal value of 262140 (0x3FFFC) for each the errorcode ERR_CAN_Timeout_Errorcode and the warningcode ERR_CAN_Timeout_Warningcode. Thus, only the two flags CAN_EXT_State_Request_Timeout and CAN_EXT_Torque_Request_Timeout are handled. To switch to custom timeout handling, taking care also of other timeouts, the parameter SDK_C_CAN_custom_Timeout_Enable has to be set to 1.

The following table shows the configuration parameters for CAN timeout handling:

Function	Datatype	Min	Max	Description
SDK_C_CAN_Custom_Timeout_Enable	UInt8	0	1	Switch between FRIWO default timeout handling and custom timeout handling. 0: default, 1: custom Default value: 0
ERR_C_CAN_Timeout_Enable	Float32	0	1	If set to 1, a non-zero timeout errorcode ERR_CAN_Timeout_Errorcode directly leads to a system error and powerstage shutdown. Default value: 1
ERR_C_SDK_CAN_Timeout_Errorcode_Filter	UInt32	0	2^{32}-1	This parameter serves as bitwise filter mask for the 32bit-codeword of timeouts, resulting in the errorcode ERR_CAN_Timeout_Errorcode. If a filter-bit is 0, the codeword-bit gets passed unchanged. Filter value is applied if SDK_C_CAN_custom_Timeout_Enable is set to 1. Default value: 0
ERR_C_SDK_CAN_Timeout_Warningcode_Filter	UInt32	0	2^{32}-1	This parameter serves as bitwise filter mask for the 32bit-codeword of timeouts, resulting in the warningcode. If a filter-bit is 0, the codeword-bit gets passed unchanged. Filter value is applied if SDK_C_CAN_custom_Timeout_Enable is set to 1. Default value: 0
ERR_E_CAN_Timeout	Float32	0	1	Error flag for system shutdown. Is only set, if ERR_C_CAN_Timeout_Enable is set to 1 and ERR_CAN_Timeout_Errorcode is non-zero.

Function	Datatype	Min	Max	Description
ERR_W_CAN_Timeout	Float32	0	1	Warning flag; is only set, if ERR_CAN_Timeout_Warningcode is non-zero.
ERR_CAN_Timeout_Errorcode	UInt32	0	$2^{32}-1$	Errorcode of timeouts after bitwise filtering by ERR_C_SDK_CAN_Timeout_Errorcode_Filter
ERR_CAN_Timeout_Warningcode	UInt32	0	$2^{32}-1$	Warningcode of timeouts after bitwise filtering by ERR_C_SDK_CAN_Timeout_Warningcode_Filter

Feedback

We are working very hard to improve our products and therefore **feedback** is indispensable! Please send us your valuable feedback as contact form or via Mail to feedback@friwo.com

https://friwo.link/md/feedback