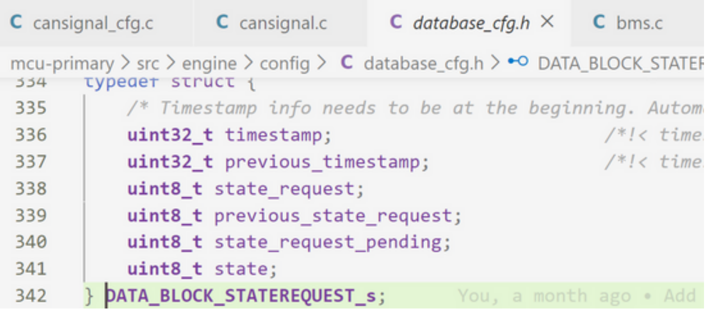
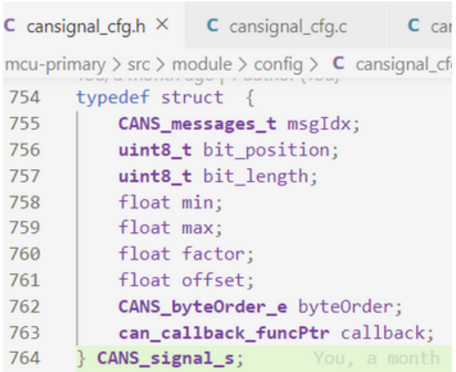
```
/**
 *
 * @copyright &copy; 2010 - 2020, Fraunhofer-Gesellschaft zur Foerderung der
 *  angewandten Forschung e.V. All rights reserved.
 *
 * BSD 3-Clause License
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * 1.  Redistributions of source code must retain the above copyright notice,
 *     this list of conditions and the following disclaimer.
 * 2.  Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 * 3.  Neither the name of the copyright holder nor the names of its
 *     contributors may be used to endorse or promote products derived from
 *     this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * We kindly request you to use one or more of the following phrases to refer
 * to foxBMS in your hardware, software, documentation or advertising
 * materials:
 *
 * &Prime;This product uses parts of foxBMS&reg;&Prime;
 *
 * &Prime;This product includes parts of foxBMS&reg;&Prime;
 *
 * &Prime;This product is derived from foxBMS&reg;&Prime;
 *
 */

/**
 * @file    cansignal.c
 * @author  foxBMS Team
 * @date    01.10.2015 (date of creation)
 * @ingroup DRIVERS
 * @prefix  CANS
 *
 * @brief   Messages and signal settings for the CAN driver
 *
 * generic conversion module of Can signals from CAN buffered reception to
 * DATA Manager and vice versa
```

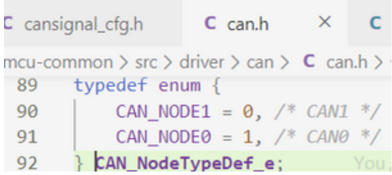
```c
 *
 */

/*================== Includes =============================================*/
#include "cansignal.h"

#include "database.h"
#include "diag.h"
#include "foxmath.h"
#include "os.h"

/*================== Macros and Definitions

/*================== Constant and Variable D
static CANS_STATE_s cans_state = {
        .periodic_enable = FALSE,
        .current_sensor_present = FALSE,
        .current_sensor_cc_present = FALSE,
    };

static DATA_BLOCK_STATEREQUEST_s canstatereq_tab;

/*================== Function Prototypes ==================================*/
```


cansignal_cfg.c
cansignal.c
database_cfg.h
bms.c
mcu-primary > src > engine > config > database_cfg.h > DATA_BLOCK_STATER

```c
typedef struct {
    /* Timestamp info needs to be at the beginning. Autom
    uint32_t timestamp;                          /*!< time
    uint32_t previous_timestamp;                 /*!< time
    uint8_t state_request;
    uint8_t previous_state_request;
    uint8_t state_request_pending;
    uint8_t state;
} DATA_BLOCK_STATEREQUEST_s;     You, a month ago • Add
```



```c
typedef struct  {
    CANS_messages_t msgIdx;
    uint8_t bit_position;
    uint8_t bit_length;
    float min;
    float max;
    float factor;
    float offset;
    CANS_byteOrder_e byteOrder;
    can_callback_funcPtr callback;
} CANS_signal_s;     You, a month
```



```c
typedef enum {
    CAN_NODE1 = 0, /* CAN1 */
    CAN_NODE0 = 1, /* CAN0 */
} CAN_NodeTypeDef_e;     You
```

```c
static STD_RETURN_TYPE_e CANS_PeriodicTransmit(void);
static STD_RETURN_TYPE_e CANS_PeriodicReceive(void);
```

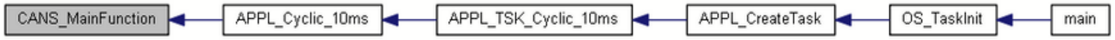
Line 143
Line 199

```c
static void CANS_SetSignalData(CANS_signal_s signal, uint64_t value, uint8_t *dataPtr);
static void CANS_GetSignalData(uint64_t *dst, CANS_signal_s signal, uint8_t *dataPtr);
static void CANS_ComposeMessage(CAN_NodeTypeDef_e canNode, CANS_messagesTx_e msgIdx, uint8_t dataptr[]);
static void CANS_ParseMessage(CAN_NodeTypeDef_e canNode, CANS_messagesRx_e msgIdx, uint8_t dataptr[]);
```

They define CAN message ID indexes.

```c
static uint8_t CANS_CheckCanTiming(void);
static void CANS_SetCurrentSensorPresent(uint8_t command);
static void CANS_SetCurrentSensorCCPresent(uint8_t command);
/*================== Function Implementations =============================*/

/*================== Public functions =====================================*/
void CANS_Init(void) {
    /* custom initialization could be made here. right now no need for any init */
}

void CANS_MainFunction(void) {
```

Called every 10 ms.



```c
    (void)CANS_PeriodicReceive();
    CANS_CheckCanTiming();
    if (cans_state.periodic_enable == TRUE) {
        (void)CANS_PeriodicTransmit();
    }
    DIAG_SysMonNotify(DIAG_SYSMON_CANS_ID, 0);  /* task is running, state = ok */
}
```

(to CAN buffer)

```c
STD_RETURN_TYPE_e CANS_AddMessage(CAN_NodeTypeDef_e canNode, uint32_t msgID, uint8_t* ptrMsgData,
        uint32_t msgLength, uint32_t RTR) {
    STD_RETURN_TYPE_e retVal = E_NOT_OK;
    OS_TaskEnter_Critical();
    /* Function should not be interrupted by the OS during the execution */
    retVal = CAN_Send(canNode, msgID, ptrMsgData, msgLength, RTR);
    OS_TaskExit_Critical();
    return retVal;
}
```

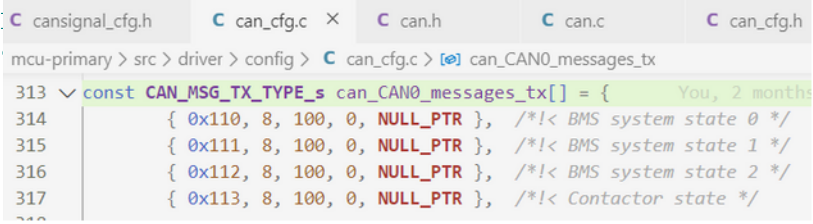
This is called as a callback function from the CAN Tx mailbox interrupt handler.

```c
STD_RETURN_TYPE_e CANS_TransmitBuffer(CAN_NodeTypeDef_e canNode) {
    STD_RETURN_TYPE_e retVal = E_NOT_OK;
    OS_TaskEnter_Critical();
    /* Function should not be interrupted by the OS during the execution */
    retVal = CAN_TxMsgBuffer(canNode);
    OS_TaskExit_Critical();
    return retVal;
}
```

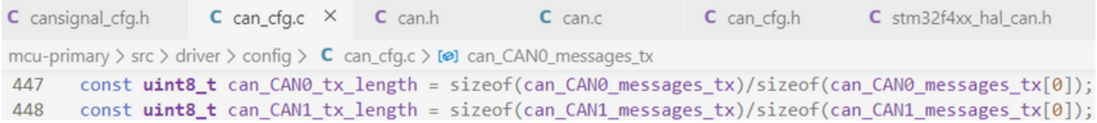
(directly)

```c
STD_RETURN_TYPE_e CANS_TransmitMessage(CAN_NodeTypeDef_e canNode, uint32_t msgID, uint8_t* ptrMsgData,
        uint32_t msgLength, uint32_t RTR) {
    STD_RETURN_TYPE_e retVal = E_NOT_OK;
    retVal = CAN_TxMsg(canNode, msgID, ptrMsgData, msgLength, RTR);
    return retVal;
}



/*================== Static functions ====================================*/
/**
 * handles the processing of messages that are meant to be transmitted.
 *
 * This function looks for the repetition times and the repetition phase of
 * messages that are intended to be sent periodically. If a comparison with
 * an internal counter (i.e., the counter how often this function has been called)
 * states that a transmit is pending, the message is composed i
 * and transfered to the buffer of the CAN module. If a callba
 * is declared in configuration, this callback is called after
 *
 * @return E_OK if a successful transfer to CAN buffer occured
 */
static STD_RETURN_TYPE_e CANS_PeriodicTransmit(void) {
    static uint32_t counter_ticks = 0;
    uint32_t i = 0;
    STD_RETURN_TYPE_e result = E_NOT_OK;

#if CAN_USE_CAN_NODE0 == TRUE
    for (i = 0; i < can_CAN0_tx_length; i++) {
        if (((counter_ticks * CANS_TICK_MS) % (can_CAN0_messages_tx[i].repetition_time)) ==
        can_CAN0_messages_tx[i].repetition_phase) {
            Can_PduType PduToSend = { {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, 0x0, 8 };
            CANS_ComposeMessage(CAN_NODE0, (CANS_messagesTx_e)(i), PduToSend.sdu);
            PduToSend.id = can_CAN0_messages_tx[i].ID;

            result = CANS_AddMessage(CAN_NODE0, PduToSend.id, PduToSend.sdu, PduToSend.dlc, 0);
```

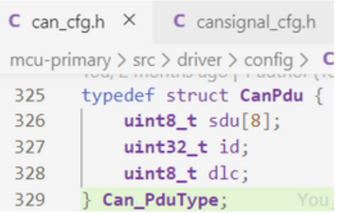
This is the total number of different CAN message IDs.


cansignal_cfg.h
can_cfg.c
can.h
can.c
can_cfg.h
mcu-primary > src > driver > config > can_cfg.c > can_CAN0_messages_tx

```c
const CAN_MSG_TX_TYPE_s can_CAN0_messages_tx[] = {      You, 2 months
        { 0x110, 8, 100, 0, NULL_PTR },  /*!< BMS system state 0 */
        { 0x111, 8, 100, 0, NULL_PTR },  /*!< BMS system state 1 */
        { 0x112, 8, 100, 0, NULL_PTR },  /*!< BMS system state 2 */
        { 0x113, 8, 100, 0, NULL_PTR },  /*!< Contactor state */
```


cansignal_cfg.h
can_cfg.c
can.h
can.c
can_cfg.h
stm32f4xx_hal_can.h
mcu-primary > src > driver > config > can_cfg.c > can_CAN0_messages_tx

```c
const uint8_t can_CAN0_tx_length = sizeof(can_CAN0_messages_tx)/sizeof(can_CAN0_messages_tx[0]);
const uint8_t can_CAN1_tx_length = sizeof(can_CAN1_messages_tx)/sizeof(can_CAN1_messages_tx[0]);
```



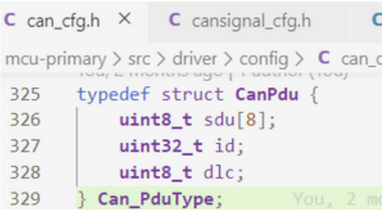
```c
typedef struct CanPdu {
    uint8_t sdu[8];
    uint32_t id;
    uint8_t dlc;
} Can_PduType;      You
```

```c
            DIAG_checkEvent(result, DIAG_CH_CANS_CAN_MOD_FAILURE, 1);

            if (can_CAN0_messages_tx[i].cbk_func != NULL_PTR && result == E_OK) {
                can_CAN0_messages_tx[i].cbk_func(i, NULL_PTR);
            }
        }
    }
#endif

#if CAN_USE_CAN_NODE1 == TRUE
    for (i = 0; i < can_CAN1_tx_length; i++) {
        if (((counter_ticks * CANS_TICK_MS) % (can_CAN1_messages_tx[i].repetition_time)) ==
        can_CAN1_messages_tx[i].repetition_phase) {
            Can_PduType PduToSend = { {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, 0x0, 8 };
```

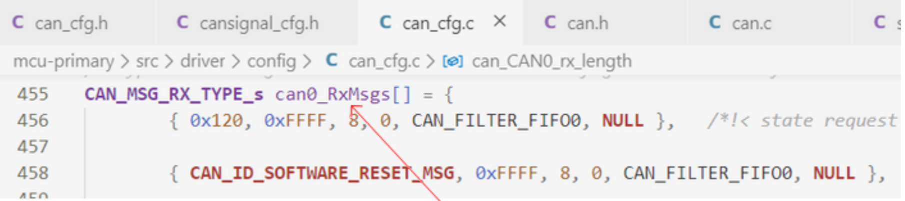
Line 325

```c
            CANS_ComposeMessage(CAN_NODE1, (CANS_messagesTx_e)i + can_CAN0_tx_length, PduToSend.sdu);
            PduToSend.id = can_CAN1_messages_tx[i].ID;

            result = CANS_AddMessage(CAN_NODE1, PduToSend.id, PduToSend.sdu, PduToSend.dlc, 0);
            DIAG_checkEvent(result, DIAG_CH_CANS_CAN_MOD_FAILURE, 0);

            if (can_CAN1_messages_tx[i].cbk_func != NULL_PTR && result == E_OK) {
                can_CAN1_messages_tx[i].cbk_func(i, NULL_PTR);
            }
        }
    }
#endif

    result = E_NOT_OK;

    counter_ticks++;
    return TRUE;
}
```

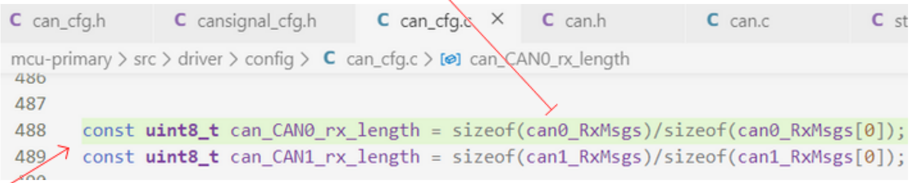
The return type should just be void!



```c
325 typedef struct CanPdu {
326     uint8_t sdu[8];
327     uint32_t id;
328     uint8_t dlc;
329 } Can_PduType;
```


C can_cfg.h
C cansignal_cfg.h
C can_cfg.c ×
C can.h
C can.c
mcu-primary > src > driver > config > C can_cfg.c > can_CAN0_rx_length

```c
CAN_MSG_RX_TYPE_s can0_RxMsgs[] = {
        { 0x120, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL },   /*!< state request

        { CAN_ID_SOFTWARE_RESET_MSG, 0xFFFF, 8, 0, CAN_FILTER_FIFO0, NULL },
```


C can_cfg.h
C cansignal_cfg.h
C can_cfg.c ×
C can.h
C can.c
C st
mcu-primary > src > driver > config > C can_cfg.c > can_CAN0_rx_length

```c
const uint8_t can_CAN0_rx_length = sizeof(can0_RxMsgs)/sizeof(can0_RxMsgs[0]);
const uint8_t can_CAN1_rx_length = sizeof(can1_RxMsgs)/sizeof(can1_RxMsgs[0]);
```

```c
/**
 * handles the processing of received CAN messages.
 *
 * This function gets the messages in the receive buffer
 * of the CAN module. If a message ID is
 * matching one of the IDs in the configuration of
 * CANS module, the signal processing is executed
 * by call to CANS_ParseMessage.
 *
 * @return E_OK, if a message has been received and parsed, E_NOT_OK otherwise
 */
static STD_RETURN_TYPE_e CANS_PeriodicReceive(void) {
    Can_PduType msg = {};
    STD_RETURN_TYPE_e result_node0 = E_NOT_OK, result_node1 = E_NOT_OK;
    uint32_t i = 0;

#if CAN_USE_CAN_NODE0 == TRUE
```

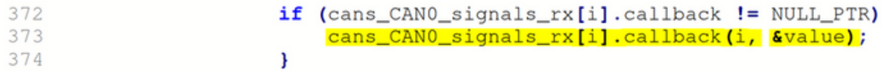
The entire message is saved in msg here.

```c
    while (CAN_ReceiveBuffer(CAN_NODE0, &msg)  ==  E_OK) {
        for (i = 0; i < can_CAN0_rx_length; i++) {
```

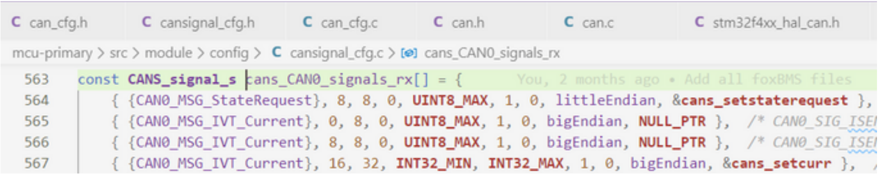
This should be named as CAN_CAN0_RxMsg_length to indicate clearly that this is for the number of CAN messages not CAN signals.

```
207                 if (msg.id  ==  can0_RxMsgs[i].ID) {
208                     CANS_ParseMessage(CAN_NODE0, (CANS_messagesRx_e)i, msg.sdu);
209                     result_node0 = E_OK;
210                 }
211             }
212         }
```

CAN messages are parsed to CAN signals in this ParseMessage function. See Line 362. The parsed data is saved in the cans_CAN0_signals_rx array, which is defined in consignal_cfg.c.



```
372                 if (cans_CAN0_signals_rx[i].callback != NULL_PTR) {
373                     cans_CAN0_signals_rx[i].callback(i, &value);
374                 }
```


C can_cfg.h
C cansignal_cfg.h
C can_cfg.c
C can.h
C can.c
C stm32f4xx_hal_can.h
mcu-primary > src > module > config > C cansignal_cfg.c > cans_CAN0_signals_rx
You, 2 months ago • Add all foxBMS files

```
563  const CANS_signal_s cans_CAN0_signals_rx[] = {
564      { {CAN0_MSG_StateRequest}, 8, 8, 0, UINT8_MAX, 1, 0, littleEndian, &cans_setstaterequest },
565      { {CAN0_MSG_IVT_Current}, 0, 8, 0, UINT8_MAX, 1, 0, bigEndian, NULL_PTR },  /* CAN0_SIG_ISEI
566      { {CAN0_MSG_IVT_Current}, 8, 8, 0, UINT8_MAX, 1, 0, bigEndian, NULL_PTR },  /* CAN0_SIG_ISEI
567      { {CAN0_MSG_IVT_Current}, 16, 32, INT32_MIN, INT32_MAX, 1, 0, bigEndian, &cans_setcurr },  /
```

```
213 #else
214     result_node0 = E_OK;
215 #endif
216 
217 #if CAN_USE_CAN_NODE1 == TRUE
218     while (CAN_ReceiveBuffer(CAN_NODE1, &msg) == E_OK) {
219         for (i = 0; i < can_CAN1_rx_length; i++) {
220             if (msg.id == can1_RxMsgs[i].ID) {
221                 CANS_ParseMessage(CAN_NODE1, (CANS_messagesRx_e)i + can_CAN0_rx_length -
                    CAN0_BUFFER_BYPASS_NUMBER_OF_IDs, msg.sdu);
222                 result_node1 = E_OK;
223             }
224         }
225     }
226 #else
227     result_node1 = E_OK;
228 #endif
229 
230     return result_node0 && result_node1;
231 }
232 /**
233  * @brief   generates bitfield, which masks the bits where the actual signal (defined by its bitlength) is located
234  *
235  * @param   bitlength   length of the signal in bits
236  *
237  * @return  bitmask     bitfield mask
238  */
239 static uint64_t CANS_GetBitmask(uint8_t bitlength) {
240     uint64_t bitmask = 0x00000000;
241     uint8_t i = 0;
242     for (i = 0; i < bitlength; i++) {
243         bitmask = bitmask << 1;
244         bitmask = bitmask | 0x00000001;
245     }
246     return bitmask;
247 }
```

bitmask = (1 << bitlength) - 1;

```
248 
249 /**
250  * extracts signal data from CAN message data
251  *
252  * @param[out] dst       pointer where the signal data should be copied to
253  * @param[in]  signal    signal identifier
254  * @param[in]  dataPtr   CAN message data, from which signal data is extracted
255  */
256 static void CANS_GetSignalData(uint64_t *dst, CANS_signal_s signal, uint8_t *dataPtr) {
257     uint64_t bitmask = 0x00000000;
```

```c
    uint64_t *dataPtr64 = (uint64_t *)dataPtr;
    /* Get signal data */
    bitmask = CANS_GetBitmask(signal.bit_length);
    *dst = (((*dataPtr64) >> signal.bit_position) & bitmask);
    /* Swap byte order if necessary */
    if (signal.byteOrder == littleEndian) {
        /* No need to switch byte order as native MCU endianness is little-endian (intel) */
    } else if (signal.byteOrder == bigEndian) {
        if (signal.bit_length <= 8) {
            /* No need to switch byte order as signal length is smaller than one byte */
        } else if (signal.bit_length <= 16) {
            /* Swap byte order */
            *dst = (uint64_t)MATH_swapBytes_uint16_t((uint16_t)*dst);
        } else if (signal.bit_length <= 32) {
            /* Swap byte order */
            *dst = (uint64_t)MATH_swapBytes_uint32_t((uint32_t)*dst);
        } else {  /* (signal.bit_length <= 64) */
            /* Swap byte order */
            *dst = MATH_swapBytes_uint64_t(*dst);
        }
    }
}


/**                        into
 * assembles signal data in CAN message data
 *
 * @param signal    signal identifier
 * @param value     signal value data
 * @param dataPtr   CAN message data, in which the signal data is inserted
 */
static void CANS_SetSignalData(CANS_signal_s signal, uint64_t value, uint8_t *dataPtr) {
    uint64_t bitmask = 0x0000000000000000;
    uint64_t *dataPtr64 = (uint64_t *)dataPtr;

    /* Swap byte order if necessary */
    if (signal.byteOrder == littleEndian) {
        /* No need to switch byte order as native MCU endianness is little-endian (intel) */
    } else if (signal.byteOrder == bigEndian) {
        if (signal.bit_length <= 8) {
            /* No need to switch byte order as signal length is smaller than one byte */
        } else if (signal.bit_length <= 16) {
            /* Swap byte order */
            value = (uint64_t)MATH_swapBytes_uint16_t((uint16_t)value);
        } else if (signal.bit_length <= 32) {
            /* Swap byte order */
            value = (uint64_t)MATH_swapBytes_uint32_t((uint32_t)value);
        } else {  /* (signal.bit_length <= 64) */
            /* Swap byte order */
            value = MATH_swapBytes_uint64_t(value);
        }
    }

    /* Set can data according to configuration */
    bitmask = CANS_GetBitmask(signal.bit_length);
    dataPtr64[0] &= ~(((uint64_t)bitmask) << signal.bit_position);
    dataPtr64[0] |= ((((uint64_t)value) & bitmask) << signal.bit_position);
}

/**
 * composes message data from all signals associated with this msgIdx
 *
 * signal data is received by callback getter functions
 *
 * @param[in] msgIdx   message index for which the data should be composed
 * @param[out] dataptr  pointer where the message data should be stored to
 */
static void CANS_ComposeMessage(CAN_NodeTypeDef_e canNode, CANS_messagesTx_e msgIdx, uint8_t dataptr[]) {
    uint32_t i = 0;
    uint32_t nrTxSignals = 0;
    /* find multiplexor if multiplexed signal */

    CANS_signal_s *cans_signals_tx;
```

The way of composing CAN messages here is very inefficient. It actually searches in two loops. The first is over all CAN messages. The second is for each CAN message, it searches over all the CAN signals, which is more than CAN messages. A more efficient approach is to search on CAN signals first and put all those with the same CAN IDs into one CAN message.

Even if we keep the same order of iterations, we can still improve the efficiency by using helper variables. Need to implement this at a later time.

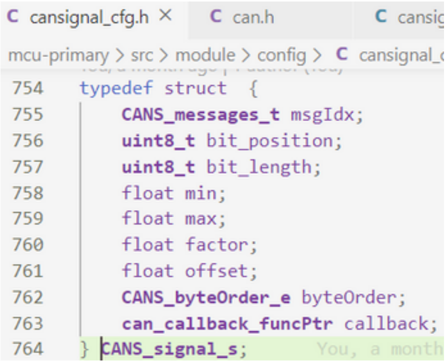
This is the pointer to CAN signals. Note that a CAN message can contain a couple of CAN signals.

```
    if (canNode == CAN_NODE0) {
        cans_signals_tx = (CANS_signal_s *)&cans_CAN0_signals_tx;
        nrTxSignals = cans_CAN0_signals_tx_length;
    } else if (canNode == CAN_NODE1) {
        cans_signals_tx = (CANS_signal_s *)&cans_CAN1_signals_tx;
        nrTxSignals = cans_CAN1_signals_tx_length;
    }
```

This is the total number of CAN signals

```
    for (i = 0; i < nrTxSignals; i++) {
        if (cans_signals_tx[i].msgIdx.Tx == msgIdx) {
            /* simple, not multiplexed signal */
            uint64_t value = 0;
            if (cans_signals_tx[i].callback != NULL_PTR) {
                cans_signals_tx[i].callback(i, &value);
            }
            CANS_SetSignalData(cans_signals_tx[i], value, dataptr);
        } else {
            /* TODO: explain why empty else */
        }
    }
}
```

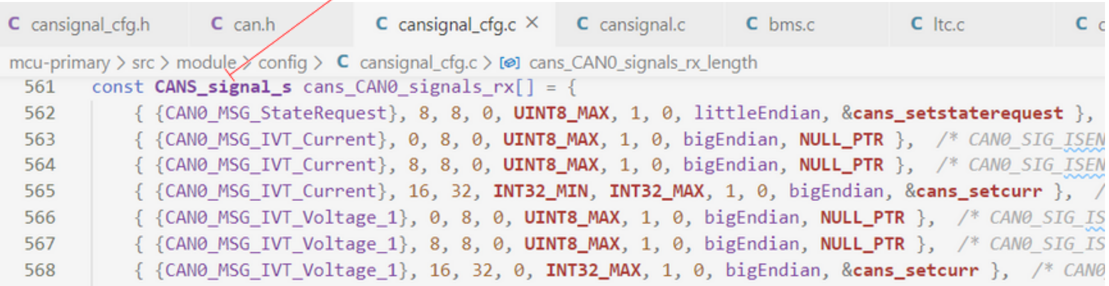
The entire CAN signal is assembled here from individual CAN signals, such as voltages of three cells.



C cansignal_cfg.h × C can.h C cansig

mcu-primary > src > module > config > C cansignal_

```
typedef struct  {
    CANS_messages_t msgIdx;
    uint8_t bit_position;
    uint8_t bit_length;
    float min;
    float max;
    float factor;
    float offset;
    CANS_byteOrder_e byteOrder;
    can_callback_funcPtr callback;
} CANS_signal_s;          You, a month
```



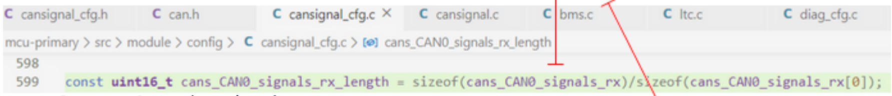
C cansignal_cfg.h C can.h C cansignal_cfg.c × C cansignal.c C bms.c C ltc.c C c

mcu-primary > src > module > config > C cansignal_cfg.c > cans_CAN0_signals_rx_length

```
const CANS_signal_s cans_CAN0_signals_rx[] = {
    { {CAN0_MSG_StateRequest}, 8, 8, 0, UINT8_MAX, 1, 0, littleEndian, &cans_setstaterequest },
    { {CAN0_MSG_IVT_Current}, 0, 8, 0, UINT8_MAX, 1, 0, bigEndian, NULL_PTR },  /* CAN0_SIG_ISEN
    { {CAN0_MSG_IVT_Current}, 8, 8, 0, UINT8_MAX, 1, 0, bigEndian, NULL_PTR },  /* CAN0_SIG_ISEN
    { {CAN0_MSG_IVT_Current}, 16, 32, INT32_MIN, INT32_MAX, 1, 0, bigEndian, &cans_setcurr },  /
    { {CAN0_MSG_IVT_Voltage_1}, 0, 8, 0, UINT8_MAX, 1, 0, bigEndian, NULL_PTR },  /* CAN0_SIG_IS
    { {CAN0_MSG_IVT_Voltage_1}, 8, 8, 0, UINT8_MAX, 1, 0, bigEndian, NULL_PTR },  /* CAN0_SIG_IS
    { {CAN0_MSG_IVT_Voltage_1}, 16, 32, 0, INT32_MAX, 1, 0, bigEndian, &cans_setcurr },  /* CAN0
```

```

/**
 * @brief   parses signal data from message associ
 *
 * signal data is received by callback setter func
 *
 * @param[in]   msgIdx   message index for which t
 * @param[in]   dataptr  pointer where the message
 */
```

```c
static void CANS_ParseMessage(CAN_NodeTypeDef_e canNode, CANS_messagesRx_e msgIdx, uint8_t dataptr[]) {
    uint32_t i = 0;

    if (canNode == CAN_NODE0) {
        for (i = 0; i < cans_CAN0_signals_rx_length; i++) {
            /* Iterate over CAN0 rx signals and find message */

            if (cans_CAN0_signals_rx[i].msgIdx.Rx  ==  msgIdx) {
                uint64_t value = 0;
                CANS_GetSignalData(&value, cans_CAN0_signals_rx[i], dataptr);
                if (cans_CAN0_signals_rx[i].callback != NULL_PTR) {
                    cans_CAN0_signals_rx[i].callback(i, &value);
```


C cansignal_cfg.h
C can.h
C cansignal_cfg.c ×
C cansignal.c
C bms.c
C ltc.c
C diag_cfg.c
mcu-primary > src > module > config > C cansignal_cfg.c > cans_CAN0_signals_rx_length

```c
const uint16_t cans_CAN0_signals_rx_length = sizeof(cans_CAN0_signals_rx)/sizeof(cans_CAN0_signals_rx[0]);
```

This is the index for the CAN message ID array, not the value of the CAN message ID.

Note that the parameters to the callback function are passed here.

```c
                }
            }
        }
    } else if (canNode == CAN_NODE1) {
        for (i = 0; i < cans_CAN1_signals_rx_length; i++) {
            /* Iterate over CAN1 rx signals and find message */

            if (cans_CAN1_signals_rx[i].msgIdx.Rx  ==  msgIdx) {
                uint64_t value = 0;
                CANS_GetSignalData(&value, cans_CAN1_signals_rx[i], dataptr);
                if (cans_CAN1_signals_rx[i].callback != NULL_PTR) {
                    cans_CAN1_signals_rx[i].callback(i, &value);
                }
            }
        }
    }
}

/**
 * @brief   Checks if the CAN messages come in the specified time window
 *
 * if actual time stamp- previous time stamp is > 96 and < 104 check is good
 * else the check is bad
 *
 * @return  TRUE if timing is in tolerance range, FLASE if not
 */

static uint8_t CANS_CheckCanTiming(void) {
```

Check CAN timing from State requests from the host.

```c
    uint8_t retVal = FALSE;

    uint32_t current_time;
    DATA_BLOCK_ERRORSTATE_s error_flags;
    DATA_BLOCK_CURRENT_SENSOR_s current_tab;


    current_time = OS_getOSSysTick();
    DB_ReadBlock(&canstatereq_tab, DATA_BLOCK_ID_STATEREQUEST);

    DB_ReadBlock(&error_flags, DATA_BLOCK_ID_ERRORSTATE);

```

We need to relax the checking thresholds to reduce unnecessary error CAN timing error message.

```
    /* Is the BMS still getting CAN messages? */    115
    if ((current_time-canstaterreq_tab.timestamp) <= 105) {                      85
        if (((canstatereq_tab.timestamp - canstatereq_tab.previous_timestamp) >= 95) && \
                ((canstatereq_tab.timestamp - canstatereq_tab.previous_timestamp) <= 105)) {
            retVal = TRUE;
            DIAG_Handler(DIAG_CH_CAN_TIMING, DIAG_EVENT_OK, 0);                  115
        } else {
            retVal = FALSE;
            DIAG_Handler(DIAG_CH_CAN_TIMING, DIAG_EVENT_NOK, 0);
        }
    } else {
        retVal = FALSE;
        DIAG_Handler(DIAG_CH_CAN_TIMING, DIAG_EVENT_NOK, 0);
    }
                                                  CAN timing error is not affected by the Current Sensor.
#if CURRENT_SENSOR_PRESENT == TRUE                The contents below need to be in another function with a descriptive name; or even two functions.
    /* check time stamps of current measurements */
    DB_ReadBlock(&current_tab, DATA_BLOCK_ID_CURRENT_SENSOR);    Need to increase this value as well.
    if (current_time-current_tab.timestamp > CURRENT_SENSOR_RESPONSE_TIMEOUT_MS) {
        DIAG_Handler(DIAG_CH_CURRENT_SENSOR_RESPONDING, DIAG_EVENT_NOK, 0);
    } else {
        DIAG_Handler(DIAG_CH_CURRENT_SENSOR_RESPONDING, DIAG_EVENT_OK, 0);
        if (cans_state.current_sensor_present == FALSE) {
            CANS_SetCurrentSensorPresent(TRUE);
        }
    }

    /* check time stamps of CC measurements */
    /* if timestamp_cc != 0, this means current sensor cc message has been received at least once */
    if (current_tab.timestamp_cc != 0) {
        if (current_time-current_tab.timestamp_cc > CURRENT_SENSOR_RESPONSE_TIMEOUT_MS) {
            DIAG_Handler(DIAG_CH_CAN_CC_RESPONDING, DIAG_EVENT_NOK, 0);
        } else {
            DIAG_Handler(DIAG_CH_CAN_CC_RESPONDING, DIAG_EVENT_OK, 0);
            if (cans_state.current_sensor_cc_present == FALSE) {
                CANS_SetCurrentSensorCCPresent(TRUE);
            }
        }
    }
#endif /* CURRENT_SENSOR_PRESENT == TRUE */

    return retVal;
}


/**
 * @brief   enable/disable the periodic transmit/receive.
 *
 * @return  none
 *
 */
extern void CANS_Enable_Periodic(uint8_t command) {
```

```c
    if (command == TRUE) {
        cans_state.periodic_enable = TRUE;
    } else {
        cans_state.periodic_enable = FALSE;
    }
}



/**
 * @brief   set flag for presence of current sensor.
 *
 * @return  none
 *
 */
static void CANS_SetCurrentSensorPresent(uint8_t command) {
    if (command == TRUE) {
        taskENTER_CRITICAL();
        cans_state.current_sensor_present = TRUE;
        taskEXIT_CRITICAL();
    } else {
        taskENTER_CRITICAL();
        cans_state.current_sensor_present = FALSE;
        taskEXIT_CRITICAL();
    }
}


/**
 * @brief   set flag for sending of C-C by current sensor.
 *
 * @return  none
 *
 */
static void CANS_SetCurrentSensorCCPresent(uint8_t command) {
    if (command == TRUE) {
        taskENTER_CRITICAL();
        cans_state.current_sensor_cc_present = TRUE;
        taskEXIT_CRITICAL();
    } else {
        taskENTER_CRITICAL();
        cans_state.current_sensor_cc_present = FALSE;
        taskEXIT_CRITICAL();
    }
}



/**
 * @brief   set flag for presence of current sensor.
 *
 * @return  retval  TRUE if a current sensor is present, FALSE otherwise
 *
 */
extern uint8_t CANS_IsCurrentSensorPresent(void) {
    uint8_t retval = FALSE;

    retval    = cans_state.current_sensor_present;

    return (retval);
}



/**
 * @brief   set flag for sending of C-C by current sensor.
 *
 * @return  retval  TRUE if C-C is being sent, FALSE otherwise
 *
 */
extern uint8_t CANS_IsCurrentSensorCCPresent(void) {
    uint8_t retval = FALSE;

    retval    = cans_state.current_sensor_cc_present;

    return (retval);
}

```