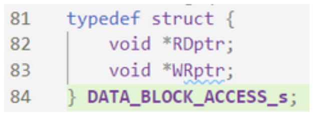
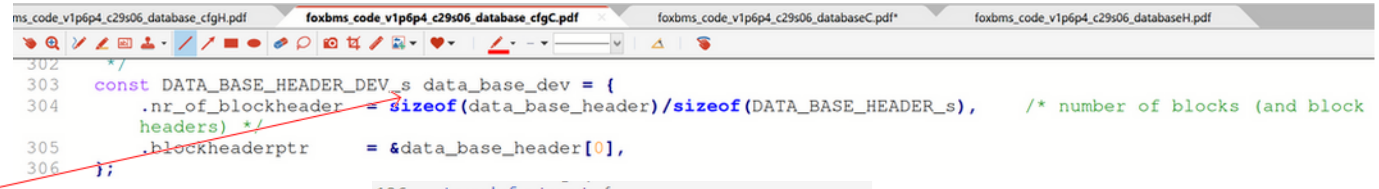
```c
/**
 *
 * @copyright &copy; 2010 - 2020, Fraunhofer-Gesellschaft zur Foerderung der
 *  angewandten Forschung e.V. All rights reserved.
 *
 * BSD 3-Clause License
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions 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    database.c
 * @author  foxBMS Team
 * @date    18.08.2015 (date of creation)
 * @ingroup ENGINE
 * @prefix  DATA
 *
 * @brief   Database module implementation
 *
 * Implementation of database module
 */
```

```c

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

#include "diag.h"
#include <string.h>

/*================== Macros and Definitions ===================================*/
/**
 * Maximum queue timeout time in milliseconds
 */
#define DATA_QUEUE_TIMEOUT_MS   (10u)

/**
 * @brief Length of data Queue
 */
#define DATA_QUEUE_LENGTH       (1u)

/**
 * @brief Size of data Queue item
 */
#define DATA_QUEUE_ITEM_SIZE    sizeof(DATA_QUEUE_MESSAGE_s)

/*================== Static Constant and Variable Definitions ===============*/
/* FIXME Some uninitialized variables */
static DATA_BLOCK_ACCESS_s data_block_access[DATA_MAX_BLOCK_NR];
QueueHandle_t data_queue;


/**
 * @brief size of Queue storage
 *
 * The array to use as the queue's storage area.
 * This must be at least #DATA_QUEUE_LENGTH * #DATA_QUEUE_ITEM_SIZE
 */
static uint8_t dataQueueStorageArea[ DATA_QUEUE_LENGTH * DATA_QUEUE_ITEM_SIZE ];

/**
 * @brief structure for static data queue
 */
static StaticQueue_t dataQueueStructure;

/*================== Extern Constant and Variable Definitions ===============*/

/*================== Static

/*================== Static

/*================== Extern
void DATA_Init(void) {
    if (sizeof(data_base_dev) == 0) {
        /* todo fatal error! */
```

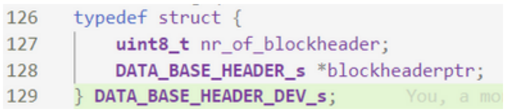


```c
typedef struct {
    void *RDptr;
    void *WRptr;
} DATA_BLOCK_ACCESS_s;
```



ms_code_v1p6p4_c29s06_database_cfgH.pdf
foxbms_code_v1p6p4_c29s06_database_cfgC.pdf
foxbms_code_v1p6p4_c29s06_databaseC.pdf*
foxbms_code_v1p6p4_c29s06_databaseH.pdf

```c
    */
const DATA_BASE_HEADER_DEV_s data_base_dev = {
    .nr_of_blockheader    = sizeof(data_base_header)/sizeof(DATA_BASE_HEADER_s),    /* number of blocks (and block
    headers) */
    .blockheaderptr       = &data_base_header[0],
};
```



```c
typedef struct {
    uint8_t nr_of_blockheader;
    DATA_BASE_HEADER_s *blockheaderptr;
} DATA_BASE_HEADER_DEV_s;         You, a mo
```

Total number of blockheaders (blocks of data in the database)

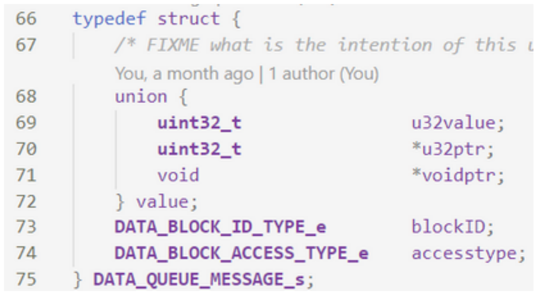
```c
        while (1) {
            /* No database defined - this should not have happened! */
        }
    }

```

This number is the same as DATA_MAX_BLOCK_NR

```c
    /* Iterate over database and set respective read/write pointer for each database entry */
    for (uint16_t i = 0; i < data_base_dev.nr_of_blockheader; i++) {
        /* Set write pointer to database entry */
        data_block_access[i].WRptr = (void*)*(uint32_t*)(data_base_dev.blockheaderptr + i);
        /* Set read pointer: read = write pointer */
        data_block_access[i].RDptr = data_block_access[i].WRptr;
```

pointing to the ith pair of data_base_header, which is the pointer of the ith data block.

```c

        /* Initialize database entry with 0, set read and write pointer in case double
         * buffering is used for database entries */
        uint8_t * startDatabaseEntryWR = (uint8_t *)data_block_access[i].WRptr;
        uint8_t * startDatabaseEntryRD = (uint8_t *)data_block_access[i].RDptr;

        for (uint16_t j = 0; j < (data_base_dev.blockheaderptr + i)->datalength; j++) {
            /* Set write pointer database entry to 0 */
            *startDatabaseEntryWR = 0;
            startDatabaseEntryWR++;

            /* Set read pointer database entry to 0 - identical to write pointer
             * if database entry is SINGLE_BUFFERED */
            *startDatabaseEntryRD = 0;
            startDatabaseEntryRD++;
        }
    }

    /* Create queue to transfer data to/from database */

    /* Create a queue capable of containing a pointer of type DATA_QUEUE_MESSAGE_s
    Data of Messages are passed by pointer as they contain a lot of data. */
    data_queue = xQueueCreateStatic(DATA_QUEUE_LENGTH, DATA_QUEUE_ITEM_SIZE, dataQueueStorageArea,
    &dataQueueStructure);

    if (data_queue == NULL_PTR) {
        /* Failed to create the queue */
        /* @ TODO Error Handling */
        while (1) {
            /* TODO: explain why infinite loop */
        }
    }
}


void DB_WriteBlock(void *dataptrfromSender, DATA_BLOCK_ID_TYPE_e  blockID) {
    /* dataptrfromSender is a pointer to data of caller function
       dataptr_toptr_fromSender is a pointer to this pointer
       this is used for passing message variable by reference
       note: xQueueSend() always takes message variable by value */
    DATA_QUEUE_MESSAGE_s data_send_msg;
```



```c
typedef struct {
    /* FIXME what is the intention of this
    You, a month ago | 1 author (You)
    union {
        uint32_t            u32value;
        uint32_t            *u32ptr;
        void                *voidptr;
    } value;
    DATA_BLOCK_ID_TYPE_e        blockID;
    DATA_BLOCK_ACCESS_TYPE_e    accesstype;
} DATA_QUEUE_MESSAGE_s;
```

```
    TickType_t queuetimeout;

    queuetimeout = DATA_QUEUE_TIMEOUT_MS / portTICK_RATE_MS;
    if (queuetimeout == 0) {
        queuetimeout = 1;
    }

    /* prepare send message with attributes of data block */
    data_send_msg.blockID = blockID;
    data_send_msg.value.voidptr = dataptrfromSender;
    data_send_msg.accesstype = WRITE_ACCESS;
    /* Send a pointer to a message object and
       maximum block time: queuetimeout */
    xQueueSend(data_queue, (void *) &data_send_msg, queuetimeout);    The actual data writing happens in DATA_Task()
}


void DATA_Task(void) {
    DATA_QUEUE_MESSAGE_s receive_msg;      Better to be called queue_message
    void *srcdataptr;          Better to be called queue_dataptr
    ~~void *dstdataptr;~~
    DATA_BLOCK_ID_TYPE_e blockID;
    DATA_BLOCK_ACCESS_TYPE_e    accesstype; /* read or write access type */
    uint16_t datalength;

    if (data_queue != NULL_PTR) {
        if (xQueueReceive(data_queue, (&receive_msg), (TickType_t) 1)) {  /* scan queue and wait for a message up to
        a maximum amount of 1ms (block time) */
            /* ptrrcvmessage now points to message of sender which contains data pointer and data block ID */
            blockID = receive_msg.blockID;
            srcdataptr = receive_msg.value.voidptr;
            accesstype = receive_msg.accesstype;
            if ((blockID < DATA_MAX_BLOCK_NR) && (srcdataptr != NULL_PTR)) {  /* plausibility check */
                /* get entries of blockheader and write pointer */
                if (accesstype == WRITE_ACCESS) {
                    /* write access to data blocks */
                    datalength = (data_base_dev.blockheaderptr + blockID)->datalength;
           void * dstdataptr = data_block_access[blockID].WRptr;

                        uint32_t *previousTimestampptr = NULL_PTR;
                        uint32_t *timestampptr = NULL_PTR;

                        /* Set timestamp pointer */
                        timestampptr = (uint32_t *)dstdataptr;
                        /* Set previous timestampptr */
                        previousTimestampptr = (uint32_t *)srcdataptr;
                        previousTimestampptr++;

                        /* Write previous timestamp */
                        *previousTimestampptr = *timestampptr;
                        /* Write timestamp */
                        *(uint32_t *)srcdataptr = OS_getOSSysTick();
```

Increment the pointer to point to the previous time stamp.

The code block to the left is an example of bad code. For time update, which should be saved in the database only, we can use the following code, for better readability:

```
dstdataptr->previous_timestap = dstdataptr->timestamp;
dstdataptr->timestamp = OS_getOSSysTick();
```

```
                                            queue_dataptr
                        memcpy(dstdataptr, srcdataptr, datalength);

                } else if (accesstype == READ_ACCESS) {
                    /* Read access to data blocks */
                    datalength = (data_base_dev.blockheaderptr + blockID)->datalength;
                    ~~dstdataptr = srcdataptr;~~

               void * srcdataptr = data_block_access[blockID].RDptr;
                    if (srcdataptr != NULL_PTR) {
                        memcpy(dstdataptr, srcdataptr, datalength);
                    }          queue dataptr,
                } else {
            /* TODO: explain why empty else */   Should report error and exit.
                }
            }
        }
        DIAG_SysMonNotify(DIAG_SYSMON_DATABASE_ID, 0);        /* task is running, state = ok */
    }
}


STD_RETURN_TYPE_e DB_ReadBlock(void *dataptrtoReceiver, DATA_BLOCK_ID_TYPE_e  blockID) {
    DATA_QUEUE_MESSAGE_s data_send_msg;         should be data_access_msg
    TickType_t queuetimeout;

    queuetimeout = DATA_QUEUE_TIMEOUT_MS / portTICK_RATE_MS;
    if (queuetimeout  ==  0) {
        queuetimeout = 1;
    }

    /* prepare send message with attributes of data block */
    data_send_msg.blockID = blockID;
    data_send_msg.value.voidptr = dataptrtoReceiver;
    data_send_msg.accesstype = READ_ACCESS;

    /* Send a pointer to a message object and */
    /* maximum block time: queuetimeout */
    xQueueSend(data_queue, (void *) &data_send_msg, queuetimeout);

    return E_OK;
}

/*================== Static functions =====================================*/
```

NOTE: There should be two versions of DB_ReadBlock and DB_WriteBolck. The first is the current one, and the second is a new one which can read/write directly without using the Queue which blocks the caller of the function. This is especially useful for those data blocks that are short.