```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    diag.h
 * @author  foxBMS Team
 * @date    09.11.2015 (date of creation)
 * @ingroup ENGINE
 * @prefix  DIAG
 *
 * @brief   Diagnosis driver header
 *
 */

```

```c
#ifndef DIAG_H_
#define DIAG_H_

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

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

/** diagnosis handler return types */
typedef enum {
    DIAG_HANDLER_RETURN_OK,                /*!<  error not occurred or occurred but threshold not reached */
    DIAG_HANDLER_RETURN_ERR_OCCURRED,      /*!<  error occurred and enabled */
    DIAG_HANDLER_RETURN_WARNING_OCCURRED, /*!<  warning occurred (error occurred but not enabled) */
    DIAG_HANDLER_RETURN_WRONG_ID,          /*!<  wrong diagnosis id */
    DIAG_HANDLER_RETURN_UNKNOWN,           /*!<  unknown return type */
    DIAG_HANDLER_INVALID_TYPE,             /*!<  invalid diagnosis type, error in configuration */
    DIAG_HANDLER_INVALID_DATA,             /*!<  invalid data, dependent of the diagHandler */
    DIAG_HANDLER_RETURN_NOT_READY,         /*!<  diagnosis handler not ready */
} DIAG_RETURNTYPE_e;

/**
 * state of diagnosis module
 */
typedef enum {
    DIAG_STATE_UNINITIALIZED,   /*!< diagnosis module not initialized              */
    DIAG_STATE_INITIALIZED,   /*!< diagnosis module initialized (ready for use)   */
} DIAG_STATE_e;

/**
 * structure of failure entry record
 */
typedef struct {
    uint8_t YY;
    uint8_t MM;
    uint8_t DD;
    uint8_t hh;
    uint8_t mm;
    uint8_t ss;
    DIAG_EVENT_e event;
    DIAG_CH_ID_e event_id;
    uint32_t item;
    uint32_t dummy1;
    uint32_t Val0;
    uint32_t Val1;
    uint32_t Val2;
    uint32_t Val3;
} DIAG_ERROR_ENTRY_s;

/* FIXME maybe short explanation why there is separate Error entry for contactor in a few words */
/**
 * structure of failure code entry record for contactor
 */
```

```c
typedef struct {
    uint8_t YY;
    uint8_t MM;
    uint8_t DD;
    uint8_t hh;
    uint8_t mm;
    uint8_t ss;
/*     DIAG_EVENT_e event; */
/*     DIAG_CH_ID_e event_id; */
    uint8_t contactor;
    float openingCurrent;
} DIAG_CONTACTOR_ERROR_ENTRY_s;

/**
 * structure contains number of switching actions for each contactor
 */
typedef struct {
    uint16_t cont_switch_closed[BS_NR_OF_CONTACTORS];
    uint16_t cont_switch_opened[BS_NR_OF_CONTACTORS];
    uint16_t cont_switch_opened_hard_at_current[BS_NR_OF_CONTACTORS];
    uint16_t errcntreported;         /*!<  number of hard switches occurred since last call of DIAG_PrintContactorInfo */
    /*            sizeof(struct) - (memory of contactors) - errcntreported - chksum */
    uint8_t reserved[0x40 - (3*BS_NR_OF_CONTACTORS*2) - 2 - 4];            /*!< reserved for future use */
} DIAG_CONTACTOR_s;

/* FIXME doxygen comment missing */
typedef struct {
    uint32_t Val0;
    uint32_t Val1;
    uint32_t Val2;
    uint32_t Val3;
} DIAG_FAILURECODE_s;

typedef struct {
    DIAG_STATE_e state;                            /*!< actual state of diagnosis module */
    uint16_t errcnttotal;                          /*!< total counts of diagnosis entry records*/
    uint16_t errcntreported;                       /*!< reported error counts to external tool*/
    uint32_t entry_event[DIAG_ID_MAX];             /*!< last detected entry event*/
    uint8_t entry_cnt[DIAG_ID_MAX];                /*!< reported event counter used for limitation  */
    uint16_t occurrence_cnt[DIAG_ID_MAX];          /*!< */
    uint8_t id2ch[DIAG_ID_MAX];                    /*!< diagnosis-id to configuration channel selector*/
    uint8_t nr_of_ch;                              /*!< number of configured channels*/
    uint32_t errflag[(DIAG_ID_MAX+31)/32];         /*!< detected error   flags (bit_nr = diag_id) */
    uint32_t warnflag[(DIAG_ID_MAX+31)/32];        /*!< detected warning flags (bit_nr = diag_id) */
    uint32_t err_enableflag[(DIAG_ID_MAX+31)/32];   /*!< enabled error flags (bit_nr = diag_id)    */
} DIAG_s;

/*================== Constant and Variable Definitions ====================*/
/* FIXME doxygen comment missing */
extern DIAG_FAILURECODE_s diag_fc;

```

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

/**
 * @brief   DIAG_Handler provides generic error handling, based on diagnosis group.
   @ingroup API_DIAG

 * This function calls the handler functions depending on the diagnosis group of call.
 * It needs to get called in every function which wants to apply some kind of diagnosis handling.
 * According to its return value further treatment is either left to the calling module itself, or
 * can be done in the callback function defined in diag_cfg.c
 *
 *
 * @param   diag_ch_id: event ID of the event that has occurred
 * @param   event:      event that occurred (OK, NOK, RESET)
 * @param   item_nr:    item nr of event, to distinguish between different calling locations of the event
 *
 * @return  DIAG_HANDLER_RETURN_UNKNOWN if invalid DIAG_TYPE_e, otherwise return value of
 *          DIAG_GeneralHandler or DIAG_ContHandler
 */
extern DIAG_RETURNTYPE_e DIAG_Handler(DIAG_CH_ID_e diag_ch_id,
                                      DIAG_EVENT_e event,
                                      uint32_t item_nr);


/**
 * @brief   DIAG_checkEvent provides a simple interface to check an event for E_OK
 *
 * @details DIAG_checkEvent is a wrapper function for DIAG_Handler. In simple cases where a return value
 *          that is not E_OK (or a 0 casted to E_OK) should increase the error counter in a diagnosis
 *          channel, this function should be used instead of directly calling the DIAG_Handler().
 *
 * @param   cond:       condition
 * @param   diag_ch_id: event ID of the event that has occurred
 * @param   item_nr:    item nr of event, to distinguish between different calling locations of the event
 *
 * @return  E_OK if ok, E_NOK if not ok
 */
extern STD_RETURN_TYPE_e DIAG_checkEvent(STD_RETURN_TYPE_e cond, DIAG_CH_ID_e diag_ch_id, uint32_t item_nr);


/**
 * @brief   DIAG_Init initializes all needed structures/buffers.
 *
 * This function provides initialization of the diagnose module.
 * In case of miss behaviour it calls Reset and adds an entry into database
 * to ensure data validity/report back malfunction
 *
 * @param   diag_dev_pointer
 */
extern STD_RETURN_TYPE_e DIAG_Init(DIAG_DEV_s *diag_dev_pointer, STD_RETURN_TYPE_e bkpramValid);

#if BUILD_MODULE_ENABLE_CONTACTOR == 1
```

```c
DIAG_RETURNTYPE_e DIAG_ContHandler(DIAG_CH_ID_e eventID, uint32_t cont_nr, float* openingCur);
#endif
/**
 * @brief   trap of configuration errors derived by FreeRTOS configASSERT
 */
extern void DIAG_configASSERT(void);

/**
 * @brief   overall system monitoring
 *
 * checks notifications (state and timestamps) of all system-relevant tasks or functions
 * all checks should be customized corresponding to its timing and state requirements
 */
extern void DIAG_SysMon(void);

/**
 * @brief   DIAG_PrintErrors prints contents of the error buffer on user request.
 *
 * This function prints out complete error buffer using the UART interface.
 */
extern void DIAG_PrintErrors(void);

#if BUILD_MODULE_ENABLE_CONTACTOR == 1
/**
 * @brief   DIAG_PrintContactorInfo prints contents of the contactor switching buffer on user request.
 *
 * This function prints out complete contactor information using the UART interface.
 */
extern void DIAG_PrintContactorInfo(void);
#endif


/**
 * @brief   DIAG_SysMonNotify has to be called in every function using the system monitoring.
 *
 * @param   module_id:  module id to notify system monitoring
 * @param   state:      state of module
 */
extern void DIAG_SysMonNotify(DIAG_SYSMON_MODULE_ID_e module_id, uint32_t state);

/*================== Function Implementations ============================*/

#endif /* DIAG_H_ */
```