

EV RTOS Project

V1

Generated by Doxygen 1.12.0

Chapter 1

Deprecated List

Global `rbCheckBlackHeight (rbtree *rbt)`

Leftovers from laptop unit tests.

Global `rbCheckOrder (rbtree *rbt, void *min, void *max)`

Leftovers from laptop unit tests.

Global `rbPrint (rbtree *rbt, void(*print_func)(void *))`

Leftovers from laptop unit tests. Sorta useless, cause like what are we gonna print to?

Global `setup_extern_devices (void *argument)`

I really dunno why this still exists, but this gets called somewhere so Im leaving it. I think we just pass it NULL.

Global `uvForceDefaultReversionUponDeviceReset ()`

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

State Engine	??
State Engine API	??
State Engine Internals	??
UVFR Utilities	??
Utility Macros	??
UVFR Vehicle Commands	??
UVFR CANbus API	??
CMSIS	??
Stm32f4xx_system	??
STM32F4xx_System_Private_Includes	??
STM32F4xx_System_Private_TypesDefinitions	??
STM32F4xx_System_Private_Defines	??
STM32F4xx_System_Private_Macros	??
STM32F4xx_System_Private_Variables	??
STM32F4xx_System_Private_FunctionPrototypes	??
STM32F4xx_System_Private_Functions	??

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

abstract_conifer_channel	??
access_control_info	??
bms_settings_t	??
CAN_Callback	??
conifer_ch_setting	??
conifer_hw_channel	??
conifer_settings	??
conifer_state	??
daq_child_task	??
daq_loop_args	??
This struct holds info of what needs to be logged	??
daq_msg	??
daq_param_list_node	??
driving_loop_args	??
drivingLoopArgs	??
Arguments for the driving loop. The reason this is a struct passed in as an argument, rather than a bunch of global variables or constants is to allow the code to take settings from flash memory, therefore allowing the team to meet it's goal of having an actual GUI to change vehicle settings	??
drivingMode	??
This is where the driving mode and the drivingModeParams are at	??
drivingModeParams	??
This struct is designed to hold information about each drivingmode's map params	??
exp_torque_map_args	??
Struct to hold parameters used in an exponential torque map	??
helper_task_args	??
linear_torque_map_args	??
motor_controller_settings	??
output_channel	??
output_channel_settings	??
p_status	??
rbnode	??
Node of a Red-Black binary search tree	??
rbtree	??
Struct representing a binary search tree	??
s_curve_torque_map_args	??
Struct for s-curve parameters for torque	??

setting_helper_task	??
state_change_daemon_args	??
task_management_info		
Struct to contain data about a parent task	??
task_status_block		
Information about the task	??
tx_all_settings_args	??
tx_journal_args	??
uv19_pdu_settings	??
uv_binary_semaphore_info	??
uv_CAN_msg		
Representative of a CAN message	??
uv_imd_settings	??
uv_init_struct	??
uv_init_task_args		
Struct designed to act like the uv_task_info struct, but for the initialisation tasks. As a result it takes fewer arguments	??
uv_init_task_response		
Struct representing the response of one of the initialization tasks	??
uv_internal_params		
Data used by the uvfr_utils library to do what it needs to do :)	??
uv_mutex_info	??
uv_os_settings		
Settings that dictate state engine behavior	??
uv_persistent_data_frame	??
uv_scd_response	??
uv_semaphore_info	??
uv_task_info		
This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine	??
uv_task_msg_t		
Struct containing a message between two tasks	??
uv_vehicle_settings	??
veh_gen_info	??
vehicle_log_entry	??

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Core/Inc/adc.h		
This file contains all the function prototypes for the adc.c file	??
Core/Inc/bms.h	??
Core/Inc/can.h		
This file contains all the function prototypes for the can.c file	??
Core/Inc/constants.h	??
Core/Inc/daq.h	??
Core/Inc/dash.h	??
Core/Inc/dma.h		
This file contains all the function prototypes for the dma.c file	??
Core/Inc/driving_loop.h	??
Core/Inc/errorLUT.h	??
Core/Inc/gpio.h		
This file contains all the function prototypes for the gpio.c file	??
Core/Inc/imd.h	??
Core/Inc/main.h		
: Header for main.c file. This file contains the common defines of the application	??
Core/Inc/motor_controller.h	??
Core/Inc/odometer.h	??
Core/Inc/pdu.h	??
Core/Inc/rb_tree.h	??
Core/Inc/stm32f4xx_hal_conf.h		
HAL configuration template file. This file should be copied to the application folder and renamed to stm32f4xx_hal_conf.h	??
Core/Inc/stm32f4xx_it.h		
This file contains the headers of the interrupt handlers	??
Core/Inc/temp_monitoring.h	??
Core/Inc/tim.h		
This file contains all the function prototypes for the tim.c file	??
Core/Inc/uvfr_conifer.h	??
Core/Inc/uvfr_diagnostics.h	??
Core/Inc/uvfr_global_config.h	??
Core/Inc/uvfr_settings.h	??
Core/Inc/uvfr_state_engine.h	??
Core/Inc/uvfr_utils.h	??

Core/Inc/uvfr_vehicle_commands.h	??
Core/Inc/uvfr_vehicle_logger.h	??
Core/Src/adc.c	
This file provides code for the configuration of the ADC instances	??
Core/Src/bms.c	??
Core/Src/can.c	
This file provides code for the configuration of the CAN instances	??
Core/Src/constants.c	??
Core/Src/daq.c	??
Core/Src/dash.c	??
Core/Src/dma.c	
This file provides code for the configuration of all the requested memory to memory DMA transfers	??
Core/Src/driving_loop.c	
File containing the meat and potatoes driving loop thread, and all supporting functions	??
Core/Src/gpio.c	
This file provides code for the configuration of all used GPIO pins	??
Core/Src/imd.c	??
Core/Src/main.c	
: Main program body	??
Core/Src/motor_controller.c	??
Core/Src/odometer.c	??
Core/Src/pdu.c	??
Core/Src/rb_tree.c	??
Core/Src/ready_to_drive.c	??
Core/Src/stm32f4xx_hal_msp.c	
This file provides code for the MSP Initialization and de-Initialization codes	??
Core/Src/stm32f4xx_hal_timebase_tim.c	
HAL time base based on the hardware TIM	??
Core/Src/stm32f4xx_it.c	
Interrupt Service Routines	??
Core/Src/syscalls.c	
STM32CubeIDE Minimal System calls file	??
Core/Src/sysmem.c	
STM32CubeIDE System Memory calls file	??
Core/Src/system_stm32f4xx.c	
CMSIS Cortex-M4 Device Peripheral Access Layer System Source File	??
Core/Src/temp_monitoring.c	??
Core/Src/tim.c	
This file provides code for the configuration of the TIM instances	??
Core/Src/uvfr_conifer.c	??
Core/Src/uvfr_diagnostics.c	??
Core/Src/uvfr_settings.c	??
Core/Src/uvfr_state_engine.c	??
Core/Src/uvfr_utils.c	??
Core/Src/uvfr_vehicle_commands.c	??
Core/Src/uvfr_vehicle_logger.c	??

Chapter 5

Topic Documentation

5.1 State Engine

Module containing all of the functions needed for the vehicle state machine to work.

Collaboration diagram for State Engine:

Topics

- [State Engine API](#)
Provides publically available API for controlling vehicle state and error handling.
- [State Engine Internals](#)

Data Structures

- struct [state_change_daemon_args](#)

Macros

- #define [MAX_NUM_MANAGED_TASKS](#) 16

Typedefs

- typedef struct state_change_daemon_args [state_change_daemon_args](#)

Functions

- [uv_status killEmAll \(\)](#)
The name should be pretty self explanatory.
- void [uvSVCTaskManager](#) (void *args)
oversees all of the service tasks, and makes sure that theyre alright
- void [uvTaskManager](#) (void *args) PRIVILEGED_FUNCTION
The big papa task that deals with handling all of the others.
- int [compareTaskByName](#) ([uv_task_info](#) *t1, [uv_task_info](#) *t2)
- void [uvTaskPeriodEnd](#) ([uv_task_info](#) *t)
Function called at the end of the task period.

Variables

- HeapStats_t [xHeapStats](#)
- uint8_t * [task_tardiness](#)
- TaskHandle_t * [scd_handle_ptr](#)
- rbtree * [task_name_lut](#) = NULL
- uint32_t [error_bitfield](#) [4]
- enum [uv_vehicle_state_t](#) [vehicle_state](#) = [UV_BOOT](#)
- enum [uv_vehicle_state_t](#) [previous_state](#) = [UV_BOOT](#)
- [uv_task_info](#) * [task_manager](#) = NULL
- [uv_task_info](#) * [svc_task_manager](#) = NULL
- rbtree * [task_name_tree](#)
- [uv_os_settings](#) * [os_settings](#) = NULL
- [uv_os_settings](#) [default_os_settings](#)

5.1.1 Detailed Description

Module containing all of the functions needed for the vehicle state machine to work.

The state-engine is mission critical code for doing the following:

- Providing a state machine for the vehicle
- Providing infrastructure neccessary for the vehicle to change state, and behaving as a parent to all the RTOS tasks
- Providing an API to hide the nitty-gritty of interfacing with the operating system, mitigating race conditions, etc...

5.1.2 Macro Definition Documentation

5.1.2.1 MAX_NUM_MANAGED_TASKS

```
#define MAX_NUM_MANAGED_TASKS 16
```

Definition at line 11 of file [uvfr_state_engine.c](#).

Referenced by [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvInitStateEngine\(\)](#), and [uvTaskManager\(\)](#).

5.1.3 Typedef Documentation

5.1.3.1 state_change_daemon_args

```
typedef struct state_change_daemon_args state_change_daemon_args
```

5.1.4 Function Documentation

5.1.4.1 compareTaskByName()

```
int compareTaskByName (
    uv_task_info * t1,
    uv_task_info * t2)
```

Definition at line 1412 of file [uvfr_state_engine.c](#).

References [uv_task_info::task_name](#).

5.1.4.2 killEmAll()

```
uv_status killEmAll ()
```

The name should be pretty self explanatory.

Definition at line 473 of file [uvfr_state_engine.c](#).

References [_BV_32](#), [UV_ERROR](#), [UV_OK](#), and [uvDeleteTask\(\)](#).

Referenced by [uvDeInitStateEngine\(\)](#).

Here is the call graph for this function:

5.1.4.3 uvSVCTaskManager()

```
void uvSVCTaskManager (
    void * args)
```

oversees all of the service tasks, and makes sure that theyre alright

Start all of the service tasks. This involves allocating neccessary memory, setting the appropriate task parameters, and saying "fuck it we ball" and adding the tasks to the central task tracking data structure.

Now we deinitialize the svcTaskManager. This is done by doing the following:

- actually shut down the svc tasks
- double check that the tasks have acually shut down
- if any svc tasks are resisting nature's call, they will be shut down forcibly
- deallocate data structures specific to `uvSVCTaskManager`

Lovely times for all

Definition at line 1365 of file [uvfr_state_engine.c](#).

References [__uvInitPanic\(\)](#), and [task_management_info::task_handle](#).

Referenced by [uvStartStateMachine\(\)](#).

Here is the call graph for this function:

5.1.4.4 uvTaskManager()

```
void uvTaskManager (
    void * args)
```

The big papa task that deals with handling all of the others.

The responsibilities of this task are as follows:

- Monitor tasks to ensure they are on schedule
- Setup inter-task communication channels?
- Invoke SCD if necessary
- Track mem usage if needed

This task is one of the most important ones in the system. Lovely times for all. Therefore it us of utmost importance that this one DOES NOT CRASH. EVER.

Definition at line 1102 of file [uvfr_state_engine.c](#).

References [uv_task_info::cmd_data](#), [killSelf\(\)](#), [uv_task_info::last_execution_time](#), [MAX_NUM_MANAGED_TASKS](#), [os_settings](#), [task_management_info::parent_msg_queue](#), [suspendSelf\(\)](#), [uv_task_info::task_flags](#), [task_management_info::task_handle](#), [uv_task_info::task_handle](#), [uv_os_settings::task_overshoot_margin_crit](#), [uv_os_settings::task_overshoot_margin_noncrit](#), [uv_task_info::task_period](#), [uv_task_info::task_state](#), [task_tardiness](#), [uv_task_info::tmi](#), [UV_INVALID_MSG](#), [UV_KILL_CMD](#), [UV_SUSPEND_CMD](#), [UV_TASK_MANAGER_MASK](#), [UV_TASK_MISSION_CRITICAL](#), [UV_TASK_RUNNING](#), [UV_TASK_VEHICLE_APPLICATION](#), [uvLateTaskHandler\(\)](#), and [uvTaskPeriodEnd\(\)](#).

Referenced by [uvStartStateMachine\(\)](#).

Here is the call graph for this function:

5.1.4.5 uvTaskPeriodEnd()

```
void uvTaskPeriodEnd (
    uv_task_info * t)
```

Function called at the end of the task period.

@

Definition at line 1469 of file [uvfr_state_engine.c](#).

References [uv_task_info::last_execution_time](#), [uv_task_info::task_id](#), [uv_task_info::task_period](#), [task_tardiness](#), [uvTaskResetDelayBit](#), and [uvTaskSetDelayBit](#).

Referenced by [uvTaskManager\(\)](#).

5.1.5 Variable Documentation

5.1.5.1 default_os_settings

```
uv_os_settings default_os_settings
```

Initial value:

```
= {  
    .svc_task_manager_period = 50,  
    .task_manager_period = 50,  
    .max_svc_task_period = 250,  
    .max_task_period = 500,  
    .min_task_period = 3,  
    .task_overshoot_margin_noncrit = 1.5F,  
    .task_overshoot_margin_crit = 1.1F  
}
```

Definition at line 48 of file [uvfr_state_engine.c](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvResetFlashToDefault\(\)](#).

5.1.5.2 error_bitfield

```
uint32_t error_bitfield[4]
```

Definition at line 36 of file [uvfr_state_engine.c](#).

5.1.5.3 os_settings

```
uv_os_settings* os_settings = NULL
```

Definition at line 46 of file [uvfr_state_engine.c](#).

Referenced by [uvStartStateMachine\(\)](#), and [uvTaskManager\(\)](#).

5.1.5.4 previous_state

```
enum uv_vehicle_state_t previous_state = UV_BOOT
```

Definition at line 39 of file [uvfr_state_engine.c](#).

Referenced by [changeVehicleState\(\)](#), and [uvStartStateMachine\(\)](#).

5.1.5.5 scd_handle_ptr

```
TaskHandle_t* scd_handle_ptr
```

Definition at line 26 of file [uvfr_state_engine.c](#).

5.1.5.6 `svc_task_manager`

```
uv_task_info* svc_task_manager = NULL
```

Definition at line 42 of file [uvfr_state_engine.c](#).

Referenced by [uvInitStateEngine\(\)](#), and [uvStartStateMachine\(\)](#).

5.1.5.7 `task_manager`

```
uv_task_info* task_manager = NULL
```

Definition at line 41 of file [uvfr_state_engine.c](#).

Referenced by [uvInitStateEngine\(\)](#), and [uvStartStateMachine\(\)](#).

5.1.5.8 `task_name_lut`

```
rbtree* task_name_lut = NULL
```

Definition at line 34 of file [uvfr_state_engine.c](#).

5.1.5.9 `task_name_tree`

```
rbtree* task_name_tree
```

Definition at line 44 of file [uvfr_state_engine.c](#).

5.1.5.10 `task_tardiness`

```
uint8_t* task_tardiness
```

Definition at line 21 of file [uvfr_state_engine.c](#).

Referenced by [uvLateTaskHandler\(\)](#), [uvTaskManager\(\)](#), and [uvTaskPeriodEnd\(\)](#).

5.1.5.11 `vehicle_state`

```
enum uv_vehicle_state_t vehicle_state = UV_BOOT
```

Definition at line 38 of file [uvfr_state_engine.c](#).

Referenced by [_stateChangeDaemon\(\)](#), [changeVehicleState\(\)](#), [handleIncomingLaptopMsg\(\)](#), [logVehicleFault\(\)](#), [StartDrivingLoop\(\)](#), [testfunc\(\)](#), [uvInitStateEngine\(\)](#), and [uvStartStateMachine\(\)](#).

5.1.5.12 xHeapStats

HeapStats_t xHeapStats

Definition at line 13 of file [uvfr_state_engine.c](#).

Referenced by [uvInitStateEngine\(\)](#).

5.1.6 State Engine API

Provides publically available API for controlling vehicle state and error handling.

Collaboration diagram for State Engine API:

Data Structures

- struct [uv_scd_response](#)
- struct [task_management_info](#)

Struct to contain data about a parent task.
- struct [task_status_block](#)

Information about the task.
- struct [uv_os_settings](#)

Settings that dictate state engine behavior.
- struct [uv_task_info](#)

This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine.

Macros

- #define [UV_TASK_VEHICLE_APPLICATION](#) 0x0001U<<(0)
- #define [UV_TASK_PERIODIC_SVC](#) 0x0001U<<(1)
- #define [UV_TASK_DORMANT_SVC](#) 0b0000000000000000011
- #define [UV_TASK_GENERIC_SVC](#) 0x0001U<<(2)
- #define [UV_TASK_MANAGER_MASK](#) 0b0000000000000000011
- #define [UV_TASK_LOG_START_STOP_TIME](#) 0x0001U<<(2)
- #define [UV_TASK_LOG_MEM_USAGE](#) 0x0001U<<(3)
- #define [UV_TASK_SCD_IGNORE](#) 0x0001U<<(4)
- #define [UV_TASK_IS_PARENT](#) 0x0001U<<(5)
- #define [UV_TASK_IS_CHILD](#) 0x0001U<<(6)
- #define [UV_TASK_IS_ORPHAN](#) 0x0001U<<(7)
- #define [UV_TASK_ERR_IN_CHILD](#) 0x0001U<<(8)
- #define [UV_TASK_AWAITING_DELETION](#) 0x0001U<<(9)
- #define [UV_TASK_DEFER_DELETION](#) 0x0001U<<(10)
- #define [UV_TASK_DEADLINE_NOT_ENFORCED](#) 0x00
- #define [UV_TASK_PRIO_INCREMENTATION](#) 0x0001U<<(11)
- #define [UV_TASK_DEADLINE_FIRM](#) 0x0001U<<(12)
- #define [UV_TASK_DEADLINE_HARD](#) (0x0001U<<(11)|0x0001U<<(12))
- #define [UV_TASK_DEADLINE_MASK](#) (0x0001U<<(11)|0x0001U<<(12))
- #define [UV_TASK_MISSION_CRITICAL](#) 0x0001U<<(13)
- #define [UV_TASK_DELAYING](#) 0x0001U<<(14)
- #define [uvTaskSetDeletionBit](#)(t)
- #define [uvTaskResetDeletionBit](#)(t)
- #define [uvTaskSetDelayBit](#)(t)
- #define [uvTaskResetDelayBit](#)(t)
- #define [uvTaskIsDelaying](#)(t)
- #define [uvTaskDelay](#)(x, t)

State engine aware vTaskDelay wrapper.
- #define [uvTaskDelayUntil](#)(x, lasttim, per)

State engine aware vTaskDelayUntil wrapper.

Typedefs

- **typedef enum uv_vehicle_state_t uv_vehicle_state**
Type representing the overall state and operating mode of the vehicle.
- **typedef enum uv_task_cmd_e uv_task_cmd**
Special commands used to start and shutdown tasks.
- **typedef struct uv_scd_response uv_scd_response**
- **typedef enum uv_task_state_t uv_task_status**
Enum representing the state of a managed task.
- **typedef enum task_priority task_priority**
Priority of a managed task. Maps directly to OS priority.
- **typedef struct task_management_info task_management_info**
Struct to contain data about a parent task.
- **typedef struct task_status_block task_status_block**
Information about the task.
- **typedef struct uv_os_settings uv_os_settings**
Settings that dictate state engine behavior.
- **typedef struct uv_task_info uv_task_info**
This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine.

Enumerations

- **enum uv_vehicle_state_t {**
UV_INIT = 0x0001 , **UV_READY** = 0x0002 , **PROGRAMMING** = 0x0004 , **UV_DRIVING** = 0x0008 ,
UV_SUSPENDED = 0x0010 , **UV_LAUNCH_CONTROL** = 0x0020 , **UV_ERROR_STATE** = 0x0040 ,
UV_BOOT = 0x0080 ,
UV_HALT = 0x0100 **}**
Type representing the overall state and operating mode of the vehicle.
- **enum uv_task_cmd_e {** **UV_NO_CMD** , **UV_KILL_CMD** , **UV_SUSPEND_CMD** , **UV_TASK_START_CMD** **}**
Special commands used to start and shutdown tasks.
- **enum uv_scd_response_e {**
UV_SUCCESSFUL_DELETION , **UV_SUCCESSFUL_SUSPENSION** , **UV_COULDNT_DELETE** ,
UV_COULDNT_SUSPEND ,
UV_UNSAFE_STATE **}**
Response from a task confirming it has been either deleted or suspended.
- **enum uv_task_state_t {** **UV_TASK_NOT_STARTED** , **UV_TASK_DELETED** , **UV_TASK_RUNNING** ,
UV_TASK_SUSPENDED **}**
Enum representing the state of a managed task.
- **enum task_priority {**
IDLE_TASK_PRIORITY , **LOW_PRIORITY** , **BELOW_NORMAL** , **MEDIUM_PRIORITY** ,
ABOVE_NORMAL , **HIGH_PRIORITY** , **REALTIME_PRIORITY** **}**
Priority of a managed task. Maps directly to OS priority.
- **enum os_flag {** **UV_OS_LOG_MEM** = 0x01 , **UV_OS_LOG_TASK_END_TIME** = 0x02 , **UV_OS_ATTEMPT_RESTART_NC_TASK** = 0x04 , **UV_OS_ENABLE_NONCRIT_TASK_THROTTLE** = 0x08 **}**

Functions

- `void uvTaskPeriodEnd (uv_task_info *t)`
Function called at the end of the task period.
- `uv_status changeVehicleState (uint16_t state)`
Function for changing the state of the vehicle, as well as the list of active + inactive tasks.
- `uv_status initRTDtask (void *args)`
- `uv_status uvInitStateEngine ()`
Function that prepares the state engine to do its thing.
- `void uvCrashIntoWall ()`
- `uv_status uvStartStateMachine ()`
Actually starts up the state engine to do state engine things.
- `uv_status uvDeInitStateEngine ()`
Stops and frees all resources used by uvfr_state_engine.
- `uv_task_info * uvCreateTask ()`
This function gets called when you want to create a task, and register it with the task register. Theres some gnarlyness here, but not unacceptable levels. Pray this thing doesn't hang itself.

5.1.6.1 Detailed Description

Provides publically available API for controlling vehicle state and error handling.

The functions defined in this group are publicly accessible and can be called from either application or service tasks. These are not necessarily interrupt safe, and therefore should not be called from them, unless they end with FromISR

5.1.6.2 Macro Definition Documentation

5.1.6.2.1 UV_TASK_AWAITING_DELETION

```
#define UV_TASK_AWAITING_DELETION 0x0001U<<(9)
```

Definition at line 211 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#), and [uvScheduleTaskDeletion\(\)](#).

5.1.6.2.2 UV_TASK_DEADLINE_FIRM

```
#define UV_TASK_DEADLINE_FIRM 0x0001U<<(12)
```

Definition at line 215 of file [uvfr_state_engine.h](#).

5.1.6.2.3 UV_TASK_DEADLINE_HARD

```
#define UV_TASK_DEADLINE_HARD (0x0001U<<(11)|0x0001U<<(12))
```

Definition at line 216 of file [uvfr_state_engine.h](#).

5.1.6.2.4 UV_TASK_DEADLINE_MASK

```
#define UV_TASK_DEADLINE_MASK (0x0001U<<(11)|0x0001U<<(12))
```

Definition at line 217 of file [uvfr_state_engine.h](#).

5.1.6.2.5 UV_TASK_DEADLINE_NOT_ENFORCED

```
#define UV_TASK_DEADLINE_NOT_ENFORCED 0x00
```

Definition at line 213 of file [uvfr_state_engine.h](#).

5.1.6.2.6 UV_TASK_DEFER_DELETION

```
#define UV_TASK_DEFER_DELETION 0x0001U<<(10)
```

Definition at line 212 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#).

5.1.6.2.7 UV_TASK_DELAYING

```
#define UV_TASK_DELAYING 0x0001U<<(14)
```

Definition at line 219 of file [uvfr_state_engine.h](#).

5.1.6.2.8 UV_TASK_DORMANT_SVC

```
#define UV_TASK_DORMANT_SVC 0b0000000000000001
```

Definition at line 201 of file [uvfr_state_engine.h](#).

5.1.6.2.9 UV_TASK_ERR_IN_CHILD

```
#define UV_TASK_ERR_IN_CHILD 0x0001U<<(8)
```

Definition at line 210 of file [uvfr_state_engine.h](#).

5.1.6.2.10 UV_TASK_GENERIC_SVC

```
#define UV_TASK_GENERIC_SVC 0x0001U<<(2)
```

Definition at line 202 of file [uvfr_state_engine.h](#).

Referenced by [uvCreateServiceTask\(\)](#), and [uvStartSVCTask\(\)](#).

5.1.6.2.11 UV_TASK_IS_CHILD

```
#define UV_TASK_IS_CHILD 0x0001U<<(6)
```

Definition at line 208 of file [uvfr_state_engine.h](#).

5.1.6.2.12 UV_TASK_IS_ORPHAN

```
#define UV_TASK_IS_ORPHAN 0x0001U<<(7)
```

Definition at line 209 of file [uvfr_state_engine.h](#).

5.1.6.2.13 UV_TASK_IS_PARENT

```
#define UV_TASK_IS_PARENT 0x0001U<<(5)
```

Definition at line 207 of file [uvfr_state_engine.h](#).

5.1.6.2.14 UV_TASK_LOG_MEM_USAGE

```
#define UV_TASK_LOG_MEM_USAGE 0x0001U<<(3)
```

Definition at line 205 of file [uvfr_state_engine.h](#).

5.1.6.2.15 UV_TASK_LOG_START_STOP_TIME

```
#define UV_TASK_LOG_START_STOP_TIME 0x0001U<<(2)
```

Definition at line 204 of file [uvfr_state_engine.h](#).

5.1.6.2.16 UV_TASK_MANAGER_MASK

```
#define UV_TASK_MANAGER_MASK 0b0000000000000001
```

Definition at line 203 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskManager\(\)](#).

5.1.6.2.17 UV_TASK_MISISON_CRITICAL

```
#define UV_TASK_MISISON_CRITICAL 0x0001U<<(13)
```

Definition at line 218 of file [uvfr_state_engine.h](#).

Referenced by [uvLateTaskHandler\(\)](#), [uvStartStateMachine\(\)](#), [uvTaskCrashHandler\(\)](#), and [uvTaskManager\(\)](#).

5.1.6.2.18 UV_TASK_PERIODIC_SVC

```
#define UV_TASK_PERIODIC_SVC 0x0001U<<(1)
```

Definition at line 200 of file [uvfr_state_engine.h](#).

5.1.6.2.19 UV_TASK_PRIO_INCREMENTATION

```
#define UV_TASK_PRIO_INCREMENTATION 0x0001U<<(11)
```

Definition at line 214 of file [uvfr_state_engine.h](#).

5.1.6.2.20 UV_TASK_SCD_IGNORE

```
#define UV_TASK_SCD_IGNORE 0x0001U<<(4)
```

Definition at line 206 of file [uvfr_state_engine.h](#).

Referenced by [uvCreateServiceTask\(\)](#), and [uvStartStateMachine\(\)](#).

5.1.6.2.21 UV_TASK_VEHICLE_APPLICATION

```
#define UV_TASK_VEHICLE_APPLICATION 0x0001U<<(0)
```

Definition at line 199 of file [uvfr_state_engine.h](#).

Referenced by [uvCreateTask\(\)](#), and [uvTaskManager\(\)](#).

5.1.6.2.22 uvTaskDelay

```
#define uvTaskDelay(
    x,
    t)
```

Value:

```
uvTaskSetDelayBit(x); \
vTaskDelay(t); \
uvTaskResetDelayBit(x)
```

State engine aware vTaskDelay wrapper.

Parameters

x	
t	is how long to delay in ticks

Definition at line 295 of file [uvfr_state_engine.h](#).

Referenced by [daqMasterTask\(\)](#).

5.1.6.2.23 uvTaskDelayUntil

```
#define uvTaskDelayUntil(
    x,
    lasttim,
    per)
```

Value:

```
uvTaskSetDelayBit(x); \
vTaskDelayUntil(&lasttim,per); \
uvTaskResetDelayBit(x)
```

State engine aware vTaskDelayUntil wrapper.

Parameters

<i>x</i>	
<i>lasttim</i>	is the variable storing the last delay time.
<i>per</i>	is the period.

This will cause the task to wait until the last time + the period.

Definition at line 316 of file [uvfr_state_engine.h](#).

Referenced by [tempMonitorTask\(\)](#).

5.1.6.2.24 uvTaskIsDelaying

```
#define uvTaskIsDelaying(
    t)
```

Value:

```
((t->task_flags&UV_TASK_DELAYING)==UV_TASK_DELAYING)
```

Definition at line 288 of file [uvfr_state_engine.h](#).

Referenced by [uvDeleteTask\(\)](#), and [uvSuspendTask\(\)](#).

5.1.6.2.25 uvTaskResetDelayBit

```
#define uvTaskResetDelayBit(
    t)
```

Value:

```
(t->task_flags&=(~UV_TASK_DELAYING))
```

Definition at line 286 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskPeriodEnd\(\)](#).

5.1.6.2.26 uvTaskResetDeletionBit

```
#define uvTaskResetDeletionBit(
    t)
```

Value:

```
(t->task_flags &= (~UV_TASK_AWAITING_DELETION))
```

Definition at line 282 of file [uvfr_state_engine.h](#).

5.1.6.2.27 uvTaskSetDelayBit

```
#define uvTaskSetDelayBit(  
    t)
```

Value:

```
(t->task_flags |=UV_TASK_DELAYING)
```

Definition at line 284 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskPeriodEnd\(\)](#).

5.1.6.2.28 uvTaskSetDeletionBit

```
#define uvTaskSetDeletionBit(  
    t)
```

Value:

```
(t->task_flags |=UV_TASK_AWAITING_DELETION)
```

Definition at line 281 of file [uvfr_state_engine.h](#).

5.1.6.3 Typedef Documentation

5.1.6.3.1 task_management_info

```
typedef struct task_management_info task_management_info
```

Struct to contain data about a parent task.

This contains the information required for the child task to communicate with it's parent.

This will be a queue, since one parent task can in theory have several child tasks

5.1.6.3.2 task_priority

```
typedef enum task_priority task_priority
```

Priority of a managed task. Maps directly to OS priority.

5.1.6.3.3 task_status_block

```
typedef struct task_status_block task_status_block
```

Information about the task.

5.1.6.3.4 uv_os_settings

```
typedef struct uv_os_settings uv_os_settings
```

Settings that dictate state engine behavior.

5.1.6.3.5 uv_scd_response

```
typedef struct uv_scd_response uv_scd_response
```

5.1.6.3.6 uv_task_cmd

```
typedef enum uv_task_cmd_e uv_task_cmd
```

Special commands used to start and shutdown tasks.

Definition at line 141 of file [uvfr_utils.h](#).

5.1.6.3.7 uv_task_info

```
typedef struct uv_task_info uv_task_info
```

This struct is designed to hold neccessary information about an RTOS task that will be managed by `uvfr_state_engine`.

Pay close attention, because this is one of the most cursed structs in the project, as well as one of the most important

5.1.6.3.8 uv_task_status

```
typedef enum uv_task_state_t uv_task_status
```

Enum representing the state of a managed task.

This is used as a flag to indicate whether or not the `state_engine` is aware of a task is running or not.

5.1.6.3.9 uv_vehicle_state

```
typedef enum uv_vehicle_state_t uv_vehicle_state
```

Type representing the overall state and operating mode of the vehicle.

Type made to represent the state of the vehicle, and the location in the state machine. The states are powers of two to make it easier to discern tasks that need to happen in multiple states

5.1.6.4 Enumeration Type Documentation

5.1.6.4.1 os_flag

```
enum os_flag
```

Enumerator

UV_OS_LOG_MEM
UV_OS_LOG_TASK_END_TIME
UV_OS_ATTEMPT_RESTART_NC_TASK
UV_OS_ENABLE_NONCRIT_TASK_THROTTLE

Definition at line 174 of file [uvfr_state_engine.h](#).

5.1.6.4.2 task_priority

```
enum task_priority
```

Priority of a managed task. Maps directly to OS priority.

Enumerator

IDLE_TASK_PRIORITY
LOW_PRIORITY
BELOW_NORMAL
MEDIUM_PRIORITY
ABOVE_NORMAL
HIGH_PRIORITY
REALTIME_PRIORITY

Definition at line 142 of file [uvfr_state_engine.h](#).

5.1.6.4.3 uv_scd_response_e

```
enum uv_scd_response_e
```

Response from a task confirming it has been either deleted or suspended.

Enumerator

UV_SUCCESSFUL_DELETION	Returned when a task was successfully deleted
UV_SUCCESSFUL_SUSPENSION	Returned when a task is successfully suspended
UV_COULDNT_DELETE	Task was not successfully deleted
UV_COULDNT_SUSPEND	Task was not successfully suspended
UV_UNSAFE_STATE	Task has ended up in a fucked middle ground state

Definition at line 113 of file [uvfr_state_engine.h](#).

5.1.6.4.4 uv_task_cmd_e

```
enum uv_task_cmd_e
```

Special commands used to start and shutdown tasks.

Enumerator

UV_NO_CMD	The SCD has issued no command, and therefore no action is required
UV_KILL_CMD	The SCD has decreed that this task must be deleted
UV_SUSPEND_CMD	The SCD has decreed that this task must be suspended
UV_TASK_START_CMD	OK for task to begin execution

Definition at line 103 of file [uvfr_state_engine.h](#).

5.1.6.4.5 uv_task_state_t

enum [uv_task_state_t](#)

Enum representing the state of a managed task.

This is used as a flag to indicate whether or not the state_engine is aware of a task is running or not.

Enumerator

UV_TASK_NOT_STARTED	
UV_TASK_DELETED	
UV_TASK_RUNNING	
UV_TASK_SUSPENDED	

Definition at line 131 of file [uvfr_state_engine.h](#).

5.1.6.4.6 uv_vehicle_state_t

enum [uv_vehicle_state_t](#)

Type representing the overall state and operating mode of the vehicle.

Type made to represent the state of the vehicle, and the location in the state machine. The states are powers of two to make it easier to discern tasks that need to happen in multiple states

Enumerator

UV_INIT	Vehicle is in the process of initializing
UV_READY	Vehicle has initialized and is ready to drive
PROGRAMMING	The settings of the vehicle are being edited now
UV_DRIVING	The vehicle is actively driving
UV_SUSPENDED	The vehicle is not allowed to produce any torque, but not full shutdown
UV_LAUNCH_CONTROL	The vehicle is presently in launch control mode
UV_ERROR_STATE	Some error has occurred here
UV_BOOT	Pre-init, when the boot loader is going
UV_HALT	Stop literally everything, except for what is needed to reset vehicle

Definition at line 88 of file [uvfr_state_engine.h](#).

5.1.6.5 Function Documentation

5.1.6.5.1 changeVehicleState()

```
uv_status changeVehicleState (
    uint16_t state)
```

Function for changing the state of the vehicle, as well as the list of active + inactive tasks.

This function also changes out the tasks that are executing, by invoking the legendary _state_change_daemon

Parameters

<code>state</code>	is a member of <code>uv_status</code> , and therefore a power of two
--------------------	--

Return values

<code>returns</code>	a member of <code>uv_status</code> depending on whether execution is successful
----------------------	---

Example usage:

```
if ((brakepedal_pressed == true) && (start_button_pressed == true)) {
    changeVehicleState(UV_DRIVING);
}
```

As you can see, all you need to do is specify the new state. Naturally, the task should be ready to get deleted by the state_change_daemon, but that is neither here nor there. If the state we wish to change to is the same as the state we're in, then no need to be executing any of this fancy code

Transition from UV_INIT to UV_READY states

Transition from UV_INIT to UV_ERROR states

Definition at line 92 of file `uvfr_state_engine.c`.

References `_stateChangeDaemon()`, `isPowerOfTwo`, `state_change_daemon_args::meta_task_handle`, `previous_state`, `UV_ABORTED`, `UV_ERROR`, `UV_ERROR_STATE`, `UV_INIT`, `UV_OK`, `UV_READY`, and `vehicle_state`.

Referenced by `__uvPanic()`, `handleDiagnosticMsg()`, `handleIncomingLaptopMsg()`, `rtdTask()`, `testfunc()`, `uvInit()`, and `uvSettingsProgrammerTask()`.

Here is the call graph for this function:

5.1.6.5.2 initRTDtask()

```
uv_status initRTDtask (
    void * args)
```

Definition at line 12 of file `ready_to_drive.c`.

References `uv_task_info::active_states`, `uv_task_info::deletion_states`, `PROGRAMMING`, `rtdTask()`, `uv_task_info::stack_size`, `uv_task_info::suspension_states`, `uv_task_info::task_args`, `uv_task_info::task_function`, `uv_task_info::task_name`, `uv_task_info::task_period`, `uv_task_info::task_priority`, `UV_DRIVING`, `UV_ERROR`, `UV_ERROR_STATE`, `UV_OK`, `UV_READY`, and `uvCreateTask()`.

Referenced by `uvInitStateEngine()`.

Here is the call graph for this function:

5.1.6.5.3 uvCrashIntoWall()

```
void uvCrashIntoWall ()
```

Definition at line 204 of file [uvfr_state_engine.c](#).

5.1.6.5.4 uvCreateTask()

```
uv_task_info * uvCreateTask ()
```

This function gets called when you want to create a task, and register it with the task register. Theres some gnarlyness here, but not unacceptable levels. Pray this thing doesn't hang itself.

Do not exceed the number of tasks available

Acquire the pointer to the spot in the array, we are doing this since we need to return the pointer anyways, and it cleans up the syntax a little.

Definition at line 285 of file [uvfr_state_engine.c](#).

References [_UV_DEFAULT_TASK_STACK_SIZE](#), [uv_task_info::active_states](#), [uv_task_info::deletion_states](#), [MAX_NUM_MANAGED_TASKS](#), [uv_task_info::parent](#), [uv_task_info::stack_size](#), [uv_task_info::suspension_states](#), [uv_task_info::task_flags](#), [uv_task_info::task_function](#), [uv_task_info::task_handle](#), [uv_task_info::task_id](#), [uv_task_info::task_name](#), [uv_task_info::task_priority](#), [uv_task_info::task_state](#), [UV_TASK_NOT_STARTED](#), and [UV_TASK_VEHICLE_APPLICATION](#).

Referenced by [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), and [uvConfigSettingTask\(\)](#).

5.1.6.5.5 uvDeInitStateEngine()

```
uv_status uvDeInitStateEngine ()
```

Stops and frees all resources used by [uvfr_state_engine](#).

If we need to initialize the state engine, gotta de-initialize as well. This is the opposite of [uvInitStateEngine](#)

Definition at line 275 of file [uvfr_state_engine.c](#).

References [killEmAll\(\)](#).

Here is the call graph for this function:

5.1.6.5.6 uvInitStateEngine()

```
uv_status uvInitStateEngine ()
```

Function that prepares the state engine to do its thing.

This is called when the system is first starting up.

Definition at line 168 of file [uvfr_state_engine.c](#).

References [__uvInitPanic\(\)](#), [associateDaqParamWithVar\(\)](#), [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [MAX_NUM_MANAGED_TASKS](#), [OS_AVAILABLE_HEAP](#), [OS_LARGEST_FREE_BLOCK](#), [OS_MIN_EVER_FREE_BYTES](#), [OS_NUM_FREE_BLOCKS](#), [OS_NUM_SUCCESSFUL_ALLOCS](#), [OS_NUM_SUCCESSFUL_FREE_BLOCKS](#), [OS_SMALLEST_FREE_BLOCK](#), [svc_task_manager](#), [task_manager](#), [UV_OK](#), [uvConfigSettingTask\(\)](#), [uvCreateServiceTask\(\)](#), [VCU_VEHICLE_STATE](#), [vehicle_state](#), and [xHeapStats](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

5.1.6.5.7 uvStartStateMachine()

```
uv_status uvStartStateMachine ()
```

Actually starts up the state engine to do state engine things.

This function ensures that all of the managed tasks are setup in a legal way, and then it allocates resources for, and starts the state engine and the background tasks. This unlocks the ability for the vehicle to do basically anything.

Definition at line 213 of file [uvfr_state_engine.c](#).

References [current_vehicle_settings](#), [os_settings](#), [uv_vehicle_settings::os_settings](#), [previous_state](#), [uv_task_info::stack_size](#), [svc_task_manager](#), [uv_task_info::task_flags](#), [uv_task_info::task_function](#), [task_manager](#), [uv_task_info::task_name](#), [uv_task_info::task_period](#), [UV_ERROR](#), [UV_INIT](#), [UV_OK](#), [UV_TASK_MISSION_CRITICAL](#), [UV_TASK_SCD_IGNORE](#), [uvSVCTaskManager\(\)](#), [uvTaskManager\(\)](#), [uvValidateManagedTasks\(\)](#), and [vehicle_state](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

5.1.6.5.8 uvTaskPeriodEnd()

```
void uvTaskPeriodEnd (
    uv_task_info * t)
```

Function called at the end of the task period.

@

Definition at line 1469 of file [uvfr_state_engine.c](#).

References [uv_task_info::last_execution_time](#), [uv_task_info::task_id](#), [uv_task_info::task_period](#), [task_tardiness](#), [uvTaskResetDelayBit](#), and [uvTaskSetDelayBit](#).

Referenced by [uvTaskManager\(\)](#).

5.1.7 State Engine Internals

Collaboration diagram for State Engine Internals:

Functions

- `uv_status addTaskToTaskRegister (uv_task_id id, uint8_t assign_to_whom)`
 make sure the parameters of a task_info struct is valid
- `uv_status _uvValidateSpecificTask (uv_task_id id)`
- `uv_status uvValidateManagedTasks ()`
 ensure that all the tasks people have created actually make sense, and are valid
- `uv_status uvStartTask (uint32_t *tracker, uv_task_info *t)`
 : This is a function that starts tasks which are already registered in the system
- `uv_status uvDeleteTask (uint32_t *tracker, uv_task_info *t)`
 deletes a managed task via the system
- `uv_status uvAbortTaskDeletion (uv_task_info *t)`
 If a task is scheduled for deletion, we want to be able to resurrect it.
- `uv_status uvScheduleTaskDeletion (uint32_t *tracker, uv_task_info *t)`
 Schedule a task to be deleted in the future double plus ungood imho.
- `uv_status uvSuspendTask (uint32_t *tracker, uv_task_info *t)`
 function to suspend one of the managed tasks.
- `uv_status uvTaskCrashHandler (uv_task_info *t)`
 Called when a task has crashed and we need to figure out what to do with it.
- `void __uvPanic (char *msg, fault_event_type_e type, const char *file, const int line, const char *func)`
 Something bad has occurred here now we in trouble.
- `void killSelf (uv_task_info *t)`
 This function is called by a task to nuke itself. Is a wrapper function that is used to do all the different things.
- `void suspendSelf (uv_task_info *t)`
 Called by a task that needs to suspend itself, once the task has determined it is safe to do so.
- `void uvSendTaskStatusReport (uv_task_info *t)`
- `void _stateChangeDaemon (void *args) PRIVILEGED_FUNCTION`
 This collects all the data changing from different tasks, and makes sure that everything works properly.
- `uv_status uvThrottleNonCritTasks ()`
- `void uvLateTaskHandler (uv_task_info *t, TickType_t tdiff, uint8_t task_tardiness)`
- `uv_task_info * uvCreateServiceTask ()`
 Create a new service task, because fuck you, thats why.
- `uv_status uvStartSVCTask (uv_task_info *t)`
 Function to start a service task specifically.
- `uv_status uvSuspendSVCTask (uv_task_info *t)`
 Function that suspends a service task.
- `uv_status uvDeleteSVCTask (uv_task_info *t)`
 For when you need to delete a service task... for some reason...
- `uv_status uvRestartSVCTask (uv_task_info *t)`
 Function that takes a service part that may be messed up and tries to reboot it to recover.
- `uv_task_info * uvGetTaskFromName (char *tsk_name)`
- `uv_task_info * uvGetTaskFromRTOSHandle (TaskHandle_t t_handle)`
 Returns the pointer to the task info structure.

Variables

- `uint8_t is_can_ok`

5.1.7.1 Detailed Description

Attention

Do not edit these functions, or even contemplate calling one of them directly unless you 100% know what you are doing. These are DANGEROUS

This handles all the under the hood bullshit inherent to a system that dynamically starts and restarts RTOS tasks. Due to this being a safety critical system, great care must be taken to prevent the vehicle from entering an unsafe state as a result of anything happening in these functions.

5.1.7.2 Function Documentation

5.1.7.2.1 __uvPanic()

```
void __uvPanic (
    char * msg,
    fault_event_type_e type,
    const char * file,
    const int line,
    const char * func)
```

Something bad has occurred here now we in trouble.

General idea here: Something bad has happened that is severe enough that it requires the shutdown of the vehicle. This can mean several things, such as being on fire, etc... that need to be appropriately handled

This should also log whatever the fuck happened.

The following should happen, in order:

- Forcibly put vehicle into a safe state
- Change vehicle state to error, and invoke the SCD
- Log the error in our lil running journal

Should change vehicle state itself be the source of the error, we just need the software to completely fucking hang itself. If things are so fucked that we genuinely cannot even transition to the error state, then get that shit the fuck outta here, we shuttin down fr fr.

Definition at line [708](#) of file [uvfr_state_engine.c](#).

References [changeVehicleState\(\)](#), [is_can_ok](#), [logVehicleFault\(\)](#), and [UV_ERROR_STATE](#).

Here is the call graph for this function:

5.1.7.2.2 _stateChangeDaemon()

```
void _stateChangeDaemon (
    void * args)
```

This collects all the data changing from different tasks, and makes sure that everything works properly.

Attention

DO NOT EVER JUST CALL THIS FUNCTION. THIS SHOULD ONLY BE CALLED FROM `changeVehicleState`

Parameters

args	This accepts a <code>void*</code> pointer to avoid compile errors with freeRTOS, since freeRTOS expects a pointer to the function that accepts a void pointer
-------------	---

This is a one-shot RTOS task that spawns in when we want to change the state of the vehicle state. It performs this in the following way We get to iterate through all of the managed tasks. Goes via IDs as well. We load up the array entry as a temp pointer to a task info struct. As we go through it determines what to do by comparing the `uv_task_info.active_states` as well as `uv_task_info.deletion_states` and `uv_task_info.suspension_states` with `uv_vehicle_state`

This is done with the bitwise & operator, since the definition of the `uv_vehicle_state_t` enum facilitates this by only using factors of two.

Acquires pointer to task definition struct, then sets the queue in the struct to the SCD queue, so that the task actually does task things. Love when that happens. Next it sets the bit in the `task_tracker` corresponding to the task id, therefore marking that some action must be taken to either

- confirm that no action is necessary
- bring the task state into the correct state

Now we suspend the task because it has been misbehaving in school

Wait for all the tasks that had changes made to respond.

*/

```
uv_scd_response* response = NULL;

for(int i = 0; i < _LONGEST_SC_TIME/_SC_DAEMON_PERIOD; i++){ //This loop verifies to make sure things
    are actually chillin
    vTaskDelay(_SC_DAEMON_PERIOD);
    for(int j = 0;j<10;j++){ //What kinda magic number is this? Why 10?
        if(xQueueReceive(state_change_queue,&response,1) == pdPASS){

            if(response == NULL){//definately not supposed to happen
                uvPanic("null scd response",0);
            }

            if(processSCDMMsg(response)==UV_OK){
                task_tracker &= ~(0x01<<response->meta_id);
                if (_task_register[response->meta_id].task_state == UV_TASK_DELETED){
                    _task_register[response->meta_id].task_handle = NULL;
                }
            }

            }else{
                //Not ok, this means that process SCD has returned something weird. More detailed
                error_handling can be added later.
                uvPanic("Task giving Sass to SCD",0);
            }

            if(uvFree(response)!=UV_OK){
                uvPanic("failed to free memory", 0);
            }
            response = NULL;
        }
        else{
            break;
        }
    }

}

//You timed out didnt you... Naughty naughty...
if(task_tracker != 0){
    uvPanic("SCD Timeout",0);
}
```

```

//TODO: Forcibly reconcile vehicle state, and nuke whatever tasks require nuking, suspend whatever needs
suspended

//END_OF_STATE_CHANGE_DAEMON:

//}

TaskHandle_t scd_handle = ((state_change_daemon_args*)args)->meta_task_handle;
uvFree(args);

vQueueDelete(state_change_queue);
state_change_queue = NULL;
/***
The final act of the SCD, is to delete itself
*/
vTaskDelete(scd_handle);

```

Definition at line 881 of file [uvfr_state_engine.c](#).

References [_LONGEST_SC_TIME](#), [_SC_DAEMON_PERIOD](#), [uv_task_info::active_states](#), [uv_task_info::deletion_states](#), [uv_scd_response::meta_id](#), [uv_task_info::suspension_states](#), [uv_task_info::task_flags](#), [uv_task_info::task_handle](#), [uv_task_info::task_state](#), [UV_OK](#), [UV_TASK_AWAITING_DELETION](#), [UV_TASK_DEFER_DELETION](#), [UV_TASK_DELETED](#), [UV_TASK_NOT_STARTED](#), [UV_TASK_RUNNING](#), [UV_TASK_SUSPENDED](#), [uvDeleteTask\(\)](#), [uvScheduleTaskDeletion\(\)](#), [uvStartTask\(\)](#), [uvSuspendTask\(\)](#), and [vehicle_state](#).

Referenced by [changeVehicleState\(\)](#).

Here is the call graph for this function:

5.1.7.2.3

```

uv_status _uvValidateSpecificTask (
    uv_task_id id)

```

make sure the parameters of a task_info struct is valid

Definition at line 344 of file [uvfr_state_engine.c](#).

References [uv_task_info::active_states](#), [uv_task_info::deletion_states](#), [uv_task_info::suspension_states](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [UV_ERROR](#), and [UV_OK](#).

Referenced by [addTaskToTaskRegister\(\)](#), and [uvValidateManagedTasks\(\)](#).

5.1.7.2.4 addTaskToTaskRegister()

```

uv_status addTaskToTaskRegister (
    uv_task_id id,
    uint8_t assign_to_whom)

```

Definition at line 331 of file [uvfr_state_engine.c](#).

References [_uvValidateSpecificTask\(\)](#), and [UV_OK](#).

Here is the call graph for this function:

5.1.7.2.5 killSelf()

```
void killSelf (
    uv_task_info * t)
```

This function is called by a task to nuke itself. Is a wrapper function that is used to do all the different things.

First lets load up the queue and the values in it. These come from the task we are doing.

Definition at line [724](#) of file [uvfr_state_engine.c](#).

References `uv_task_info::cmd_data`, `uv_scd_response::meta_id`, `uv_scd_response::response_val`, `uv_task_info::task_handle`, `uv_task_info::task_id`, `uv_task_info::task_state`, `UV_NO_CMD`, `UV_SUCCESSFUL_DELETION`, and `UV_TASK_DELETED`.

Referenced by `CANbusRxSvcDaemon()`, `CANbusTxSvcDaemon()`, `daqMasterTask()`, `odometerTask()`, `rtdTask()`, `StartDrivingLoop()`, `tempMonitorTask()`, `uvSettingsProgrammerTask()`, and `uvTaskManager()`.

5.1.7.2.6 suspendSelf()

```
void suspendSelf (
    uv_task_info * t)
```

Called by a task that needs to suspend itself, once the task has determined it is safe to do so.

Definition at line [765](#) of file [uvfr_state_engine.c](#).

References `uv_task_info::cmd_data`, `uv_scd_response::meta_id`, `uv_scd_response::response_val`, `uv_task_info::task_handle`, `uv_task_info::task_id`, `uv_task_info::task_state`, `UV_NO_CMD`, `UV_SUCCESSFUL_SUSPENSION`, and `UV_TASK_SUSPENDED`.

Referenced by `CANbusRxSvcDaemon()`, `CANbusTxSvcDaemon()`, `odometerTask()`, `rtdTask()`, `StartDrivingLoop()`, `tempMonitorTask()`, and `uvTaskManager()`.

5.1.7.2.7 uvAbortTaskDeletion()

```
uv_status uvAbortTaskDeletion (
    uv_task_info * t)
```

If a task is scheduled for deletion, we want to be able to resurrect it.

Calling this will find the task deletion timer, and remove the task from the grave.

Definition at line [579](#) of file [uvfr_state_engine.c](#).

References `UV_ERROR`, and `UV_OK`.

5.1.7.2.8 uvCreateServiceTask()

```
uv_task_info * uvCreateServiceTask ()
```

Create a new service task, because fuck you, thats why.

Acquire the pointer to the spot in the array, we are doing this since we need to return the pointer anyways, and it cleans up the syntax a little.

Definition at line 1217 of file [uvfr_state_engine.c](#).

References [_UV_DEFAULT_TASK_STACK_SIZE](#), [uv_task_info::active_states](#), [uv_task_info::deletion_states](#), [MAX_NUM_MANAGED_TASKS](#), [uv_task_info::parent](#), [uv_task_info::stack_size](#), [uv_task_info::suspension_states](#), [uv_task_info::task_flags](#), [uv_task_info::task_function](#), [uv_task_info::task_handle](#), [uv_task_info::task_id](#), [uv_task_info::task_name](#), [uv_task_info::task_priority](#), [uv_task_info::task_state](#), [UV_TASK_GENERIC_SVC](#), [UV_TASK_NOT_STARTED](#), and [UV_TASK_SCD_IGNORE](#).

Referenced by [uvInit\(\)](#), and [uvInitStateEngine\(\)](#).

5.1.7.2.9 uvDeleteSVCTask()

```
uv_status uvDeleteSVCTask (
    uv_task_info * t)
```

For when you need to delete a service task... for some reason...

Definition at line 1313 of file [uvfr_state_engine.c](#).

References [uv_task_info::cmd_data](#), [uv_task_info::task_handle](#), [uv_task_info::task_state](#), [UV_ABORTED](#), [UV_ERROR](#), [UV_KILL_CMD](#), [UV_OK](#), [UV_TASK_DELETED](#), [UV_TASK_NOT_STARTED](#), [UV_TASK_RUNNING](#), and [UV_TASK_SUSPENDED](#).

Referenced by [uvRestartSVCTask\(\)](#).

5.1.7.2.10 uvDeleteTask()

```
uv_status uvDeleteTask (
    uint32_t * tracker,
    uv_task_info * t)
```

Deletes a managed task via the system

This function is the lowtier god of the program. It pulls up and is like "YOU SHOULD KILL YOURSELF, NOW!!" It sends a message to the task which tells it to kill itself.

The task complies. It does not have a choice. This checks with the RTOS kernel to see that the task as stated by the scheduler matches the state known by uvfr_utils

Definition at line 519 of file [uvfr_state_engine.c](#).

References [uv_task_info::cmd_data](#), [uv_task_info::task_handle](#), [uv_task_info::task_id](#), [uv_task_info::task_state](#), [UV_ABORTED](#), [UV_ERROR](#), [UV_KILL_CMD](#), [UV_OK](#), [UV_TASK_DELETED](#), [UV_TASK_NOT_STARTED](#), [UV_TASK_SUSPENDED](#), and [uvTaskIsDelaying](#).

Referenced by [_stateChangeDaemon\(\)](#), and [killEmAll\(\)](#).

5.1.7.2.11 uvGetTaskFromName()

```
uv_task_info * uvGetTaskFromName (
    char * tsk_name)
```

Sometimes you just gotta deal with it lol

Definition at line 1448 of file [uvfr_state_engine.c](#).

5.1.7.2.12 uvGetTaskFromRTOSHandle()

```
uv_task_info * uvGetTaskFromRTOSHandle (
    TaskHandle_t t_handle)
```

Returns the pointer to the task info structure.

Parameters

<i>t_handle</i>	A freeRTOS task handle.
-----------------	-------------------------

Return values

<i>A</i>	pointer to a uv_task_info data structure. This is mostly useful for cases where you know the RTOS handle, but not the task info struct
----------	--

Definition at line 1460 of file [uvfr_state_engine.c](#).

5.1.7.2.13 uvLateTaskHandler()

```
void uvLateTaskHandler (
    uv_task_info * t,
    TickType_t tdiff,
    uint8_t task_tardiness)
```

Definition at line 1068 of file [uvfr_state_engine.c](#).

References [uv_task_info::task_flags](#), [uv_task_info::task_period](#), [task_tardiness](#), and [UV_TASK_MISSION_CRITICAL](#).

Referenced by [uvTaskManager\(\)](#).

5.1.7.2.14 uvRestartSVCTask()

```
uv_status uvRestartSVCTask (
    uv_task_info * t)
```

Function that takes a service part that may be messed up and tries to reboot it to recover.

This may be necessary if a SVC task is not responding. Be careful though, since this has the potential to delay more important tasks :o Therefore, this technique should be used sparingly, and each task gets a limited number of attempts within a certain time period.

Definition at line 1341 of file [uvfr_state_engine.c](#).

References [UV_ERROR](#), [UV_OK](#), [uvDeleteSVCTask\(\)](#), and [uvStartSVCTask\(\)](#).

Here is the call graph for this function:

5.1.7.2.15 uvScheduleTaskDeletion()

```
uv_status uvScheduleTaskDeletion (
    uint32_t * tracker,
    uv_task_info * t)
```

Schedule a task to be deleted in the future double plus ungood imho.

Definition at line 591 of file [uvfr_state_engine.c](#).

References [uv_task_info::task_flags](#), [uv_task_info::task_id](#), [uv_task_info::task_state](#), [UV_ABORTED](#), [UV_ERROR](#), [UV_OK](#), [UV_TASK_AWAITING_DELETION](#), and [UV_TASK_DELETED](#).

Referenced by [_stateChangeDaemon\(\)](#).

5.1.7.2.16 uvSendTaskStatusReport()

```
void uvSendTaskStatusReport (
    uv_task_info * t)
```

Definition at line 865 of file [uvfr_state_engine.c](#).

5.1.7.2.17 uvStartSVCTask()

```
uv_status uvStartSVCTask (
    uv_task_info * t)
```

Function to start a service task specifically.

Definition at line 1257 of file [uvfr_state_engine.c](#).

References [uv_task_info::stack_size](#), [uv_task_info::task_args](#), [uv_task_info::task_flags](#), [uv_task_info::task_function](#), [uv_task_info::task_handle](#), [uv_task_info::task_name](#), [uv_task_info::task_priority](#), [uv_task_info::task_state](#), [UV_ABORTED](#), [UV_ERROR](#), [UV_OK](#), [UV_TASK_GENERIC_SVC](#), [UV_TASK_RUNNING](#), and [UV_TASK_SUSPENDED](#).

Referenced by [uvRestartSVCTask\(\)](#).

5.1.7.2.18 uvStartTask()

```
uv_status uvStartTask (
    uint32_t * tracker,
    uv_task_info * t)
```

: This is a function that starts tasks which are already registered in the system

This bad boi gets called from the stateChangeDaemon because it's a special little snowflake. The first thing we will do is check if the task is running, since this could theoretically get called from literally anywhere. If the task is running, then we check to see if `t->task_handle` is set to `NULL`. If it is null, that is a physically impossible state. Neither very mindful or very demure.

That being said, if the task appears legit, then just update the corresponding bits in the tracker, and return that the task has aborted.

If a task has been suspended, we do not want to create a new instance of the task, because then the task will go out of scope, and changing the task handle to a new instance will result in the task never being de-initialized, therefore causing a memory leak. We want to call `vTaskResume` instead, and just boot the task back into existence.

If none of the previous if statements caught the task handle, then that means that either this is our first time attempting to activate this task, or the task has been deleted at some point prior to this one

Definition at line 401 of file [uvfr_state_engine.c](#).

References [uv_task_info::last_execution_time](#), [uv_task_info::stack_size](#), [uv_task_info::task_function](#), [uv_task_info::task_handle](#), [uv_task_info::task_id](#), [uv_task_info::task_name](#), [uv_task_info::task_priority](#), [uv_task_info::task_state](#), [UV_ABORTED](#), [UV_ERROR](#), [UV_OK](#), [UV_TASK_RUNNING](#), and [UV_TASK_SUSPENDED](#).

Referenced by [_stateChangeDaemon\(\)](#), and [uvInit\(\)](#).

5.1.7.2.19 uvSuspendSVCTask()

```
uv_status uvSuspendSVCTask (
    uv_task_info * t)
```

Function that suspends a service task.

Definition at line 1298 of file [uvfr_state_engine.c](#).

References [uv_task_info::task_state](#), [UV_ABORTED](#), [UV_ERROR](#), [UV_OK](#), and [UV_TASK_SUSPENDED](#).

5.1.7.2.20 uvSuspendTask()

```
uv_status uvSuspendTask (
    uint32_t * tracker,
    uv_task_info * t)
```

function to suspend one of the managed tasks.

Parameters

<i>tracker</i>	is a pointer to an int. If the task actually suspends, we update the tracker, since no further action is needed.
<i>t</i>	is a pointer to a uv_task_info struct.

Definition at line 618 of file [uvfr_state_engine.c](#).

References [uv_task_info::cmd_data](#), [uv_task_info::task_handle](#), [uv_task_info::task_id](#), [uv_task_info::task_state](#), [UV_ERROR](#), [UV_OK](#), [UV_SUSPEND_CMD](#), [UV_TASK_DELETED](#), [UV_TASK_NOT_STARTED](#), [UV_TASK_SUSPENDED](#), and [uvTaskIsDelaying](#).

Referenced by [_stateChangeDaemon\(\)](#).

5.1.7.2.21 uvTaskCrashHandler()

```
uv_status uvTaskCrashHandler (
    uv_task_info * t)
```

Called when a task has crashed and we need to figure out what to do with it.

Effectively, there are a couple variables we care about here: 1) Can the vehicle continue operation without that task active? 2) Do we really care?

If the task is critical, then this needs to 100% result in a panic. If it isn't then we can try to restart the task, noting that this may result in strange undefined behavior down the line. Thankfully if a task is not safety critical, we don't really care whether it misbehaves. Appropriate countermeasures are in place to prevent one task from overflowing into another task, as well as to mitigate against possible memory leaks.

Definition at line 675 of file [uvfr_state_engine.c](#).

References [uv_task_info::task_flags](#), [UV_ERROR](#), [UV_OK](#), and [UV_TASK_MISSION_CRITICAL](#).

5.1.7.2.22 uvThrottleNonCritTasks()

```
uv_status uvThrottleNonCritTasks ()
```

Definition at line 1063 of file [uvfr_state_engine.c](#).

5.1.7.2.23 uvValidateManagedTasks()

```
uv_status uvValidateManagedTasks ()
```

ensure that all the tasks people have created actually make sense, and are valid

Definition at line 379 of file [uvfr_state_engine.c](#).

References [_uvValidateSpecificTask\(\)](#), and [UV_OK](#).

Referenced by [uvStartStateMachine\(\)](#).

Here is the call graph for this function:

5.1.7.3 Variable Documentation

5.1.7.3.1 is_can_ok

```
uint8_t is_can_ok [extern]
```

Definition at line 74 of file [can.c](#).

Referenced by [__uCANTxCritSection\(\)](#), [__uvPanic\(\)](#), [CANbusTxSvcDaemon\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [handleCANbusError\(\)](#), and [uvSendCanMSG\(\)](#).

5.2 UVFR Utilities

Module containing useful functions and abstractions that are used throughout the vehicle software system.

Collaboration diagram for UVFR Utilities:

Topics

- Utility Macros
handy macros that perform very common functionality

5.2.1 Detailed Description

Module containing useful functions and abstractions that are used throughout the vehicle software system.

This contains several abstractions such as useful macros, global typedefs, memory allocation, etc...

5.2.2 Utility Macros

handy macros that perform very common functionality

Collaboration diagram for Utility Macros:

Macros

- `#define _BV(x)`
- `#define _BV_8(x)`
- `#define _BV_16(x)`
- `#define _BV_32(x)`
- `#define endianSwap(x)`
- `#define endianSwap8(x)`
- `#define endianSwap16(x)`
- `#define endianSwap32(x)`
- `#define deserializeSmallE16(x, i)`
- `#define deserializeSmallE32(x, i)`
- `#define deserializeBigE16(x, i)`
- `#define deserializeBigE32(x, i)`
- `#define serializeSmallE16(x, d, i)`
- `#define serializeSmallE32(x, d, i)`
- `#define serializeBigE16(x, d, i)`
- `#define serializeBigE32(x, d, i)`
- `#define setBits(x, msk, data)`

macro to set bits of an int without touching the ones we dont want to edit
- `#define isPowerOfTwo(x)`

Returns a truthy value if "x" is a power of two.
- `#define safePtrRead(x)`

lil treat to help us avoid the dreaded null pointer dereference
- `#define safePtrWrite(p, x)`
- `#define false 0`
- `#define true !false`
- `#define TEXTIFY(A)`

5.2.2.1 Detailed Description

handy macros that perform very common functionality

5.2.2.2 Macro Definition Documentation

5.2.2.2.1 _BV

```
#define _BV(  
    x)
```

Value:

`_BV_16(x)`

Definition at line 71 of file [uvfr_utils.h](#).

5.2.2.2.2 _BV_16

```
#define _BV_16(  
    x)
```

Value:

```
((uint16_t) (0x01U >> x))
```

Definition at line 73 of file [uvfr_utils.h](#).

5.2.2.2.3 _BV_32

```
#define _BV_32(  
    x)
```

Value:

```
((uint32_t) (0x01U >> x))
```

Definition at line 74 of file [uvfr_utils.h](#).

Referenced by [killEmAll\(\)](#).

5.2.2.2.4 _BV_8

```
#define _BV_8(  
    x)
```

Value:

```
((uint8_t) (0x01U >> x))
```

Definition at line 72 of file [uvfr_utils.h](#).

5.2.2.2.5 deserializeBigE16

```
#define deserializeBigE16(  
    x,  
    i)
```

Value:

```
((x[i]<<8) | (x[i+1]))
```

Definition at line 83 of file [uvfr_utils.h](#).

5.2.2.2.6 deserializeBigE32

```
#define deserializeBigE32(  
    x,  
    i)
```

Value:

```
((x[i]<<24) | (x[i+1]<<16) | (x[i+2]<<8) | (x[i+3]))
```

Definition at line 84 of file [uvfr_utils.h](#).

5.2.2.2.7 deserializeSmallE16

```
#define deserializeSmallE16(
    x,
    i)
```

Value:

```
((x[i]) | (x[i+1] << 8))
```

Definition at line 81 of file [uvfr_utils.h](#).

5.2.2.2.8 deserializeSmallE32

```
#define deserializeSmallE32(
    x,
    i)
```

Value:

```
((x[i]) | (x[i+1] << 8) | (x[i+2] << 16) | (x[i+3] << 24))
```

Definition at line 82 of file [uvfr_utils.h](#).

5.2.2.2.9 endianSwap

```
#define endianSwap(
    x)
```

Value:

```
endianSwap16(x)
```

Definition at line 76 of file [uvfr_utils.h](#).

5.2.2.2.10 endianSwap16

```
#define endianSwap16(
    x)
```

Value:

```
((((x & 0x00FF) << 8) | (((x & 0xFF00) >> 8)))
```

Definition at line 78 of file [uvfr_utils.h](#).

5.2.2.2.11 endianSwap32

```
#define endianSwap32(
    x)
```

Value:

```
((((x & 0x000000FF) << 16) | (((x & 0x0000FF00) << 8) | (((x & 0x00FF0000) >> 8) | (((x & 0xFF000000) >> 16))))
```

Definition at line 79 of file [uvfr_utils.h](#).

5.2.2.2.12 endianSwap8

```
#define endianSwap8(  
    x)
```

Value:

x

Definition at line 77 of file [uvfr_utils.h](#).

5.2.2.2.13 false

```
#define false 0
```

Wish.com Boolean

Definition at line 129 of file [uvfr_utils.h](#).

5.2.2.2.14 isPowerOfTwo

```
#define isPowerOfTwo(  
    x)
```

Value:

(x&& (! (x& (x-1))))

Returns a truthy value if "x" is a power of two.

Definition at line 119 of file [uvfr_utils.h](#).

Referenced by [changeVehicleState\(\)](#).

5.2.2.2.15 safePtrRead

```
#define safePtrRead(  
    x)
```

Value:

(* ((x) ?x:uvPanic("nullptr_deref", 0)))

lil treat to help us avoid the dreaded null pointer dereference

Definition at line 124 of file [uvfr_utils.h](#).

5.2.2.2.16 safePtrWrite

```
#define safePtrWrite(  
    p,  
    x)
```

Value:

(* ((p) ?p:&x))

Definition at line 125 of file [uvfr_utils.h](#).

5.2.2.2.17 serializeBigE16

```
#define serializeBigE16(
    x,
    d,
    i)
```

Value:

```
x[i+1]=d&0x00FF; x[i]=(d&0xFF00)»8
```

Definition at line 88 of file [uvfr_utils.h](#).

5.2.2.2.18 serializeBigE32

```
#define serializeBigE32(
    x,
    d,
    i)
```

Value:

```
x[i+3]=d&0x000000FF; x[i+2]=(d&0x0000FF00)»8; x[i+1]=(d&0x00FF0000)»16; x[i]=(d&0xFF000000)»24
```

Definition at line 89 of file [uvfr_utils.h](#).

5.2.2.2.19 serializeSmallE16

```
#define serializeSmallE16(
    x,
    d,
    i)
```

Value:

```
x[i]=d&0x00FF; x[i+1]=(d&0xFF00)»8
```

Definition at line 86 of file [uvfr_utils.h](#).

5.2.2.2.20 serializeSmallE32

```
#define serializeSmallE32(
    x,
    d,
    i)
```

Value:

```
x[i]=d&0x000000FF; x[i+1]=(d&0x0000FF00)»8; x[i+2]=(d&0x00FF0000)»16; x[i+3]=(d&0xFF000000)»24
```

Definition at line 87 of file [uvfr_utils.h](#).

Referenced by [handleIncomingLaptopMsg\(\)](#).

5.2.2.2.21 setBits

```
#define setBits(
    x,
    msk,
    data)
```

Value:

```
x=(x&(~msk)) | data)
```

macro to set bits of an int without touching the ones we dont want to edit

Usage: Will set the values of certain bits of an int. This depends on the following however:

Parameters

<i>x</i>	represents the value you want to edit. Can be any signed or unsigned integer type.
<i>msk</i>	Bits of X will only be altered if the matching bit of msk is a 1
<i>data</i>	Bits of data will map to bits of x, provided that the corresponding bit of msk is a one

In practice this looks like the following:

```
uint8_t num = 0xF0; // int is 0b11110000
uint8_t mask = 0x22; // msk is 0b000100010
uint8_t data = 0x0F; // val is 0b00001111

//now we deploy the macro

setBits(num,mask,data);

//now, num = 0b11010010
```

Definition at line 114 of file [uvfr_utils.h](#).

5.2.2.22 TEXTIFY

```
#define TEXTIFY(
    A)
```

Value:

```
#A
```

Converts a macro argument to a string literal

Definition at line 133 of file [uvfr_utils.h](#).

5.2.2.23 true

```
#define true !false
```

Definition at line 130 of file [uvfr_utils.h](#).

5.3 UVFR Vehicle Commands

A fun lil API which is used to get the vehicle to do stuff.

A fun lil API which is used to get the vehicle to do stuff.

This is designed to be portable between different versions of the VCU and PMU

5.4 UVFR CANbus API

This is an api that simplifies usage of CANbus transmitting and receiving.

Functions

- void [insertCANMessageHandler](#) (uint32_t id, void *handlerfunc, int can_num)
Function to insert an id and function into the lookup table of callback functions.
- [uv_status uvSendCanMSG](#) (uv_CAN_msg *tx_msg)
Function to send CAN message.

5.4.1 Detailed Description

This is an api that simplifies usage of CANbus transmitting and receiving.

5.4.2 Function Documentation

5.4.2.1 [insertCANMessageHandler\(\)](#)

```
void insertCANMessageHandler (
    uint32_t id,
    void * handlerfunc,
    int can_num)
```

Function to insert an id and function into the lookup table of callback functions.

Checks if specific hash id already exists in the hash table If not, insert the message If it already exists, check to see if the actual CAN id matches. If yes, then previous entries are overwritten If it does not exist, then each node in the hash table functions as its own linked list

Definition at line [652](#) of file [can.c](#).

References [callback_table_1_mutex](#), [callback_table_2_mutex](#), [CAN_BUS_1](#), [CAN_callback_table_1](#), [CAN_callback_table_2](#), [CAN_Callback::CAN_id](#), [CAN_Callback::function](#), [generateHash\(\)](#), [CAN_Callback::next](#), and [UV_ERROR](#).

Referenced by [BMS_Init\(\)](#), [MC_Startup\(\)](#), [tempMonitorTask\(\)](#), and [uvSettingsInit\(\)](#).

Here is the call graph for this function:

5.4.2.2 [uvSendCanMSG\(\)](#)

```
uv_status uvSendCanMSG (
    uv_CAN_msg * tx_msg)
```

Function to send CAN message.

This function is the canonical team method of sending a CAN message. It invokes the canTxDaemon, to avoid any conflicts due to a context switch mid transmission Is it a little bit convoluted? Yes. Is that worth it? Still yes.

Definition at line [864](#) of file [can.c](#).

References [__uvCANtxCriticalSection\(\)](#), [is_can_ok](#), [UV_ERROR](#), and [UV_OK](#).

Referenced by [flushLogsToCAN\(\)](#), [handleIncomingLaptopMsg\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [sendTorqueToMotorController\(\)](#), [tempMonitorTask\(\)](#), [testfunc2\(\)](#), [testfunc3\(\)](#), [u19updatePduChannel\(\)](#), [uvSendSpecificParam\(\)](#), [uvSettingsInit\(\)](#), and [uvSettingsProgrammerTask\(\)](#).

Here is the call graph for this function:

5.5 CMSIS

Collaboration diagram for CMSIS:

Topics

- [Stm32f4xx_system](#)

5.5.1 Detailed Description

5.5.2 Stm32f4xx_system

Collaboration diagram for Stm32f4xx_system:

Topics

- [STM32F4xx_System_Private_Includes](#)
- [STM32F4xx_System_Private_TypesDefinitions](#)
- [STM32F4xx_System_Private_Defines](#)
- [STM32F4xx_System_Private_Macros](#)
- [STM32F4xx_System_Private_Variables](#)
- [STM32F4xx_System_Private_FunctionPrototypes](#)
- [STM32F4xx_System_Private_Functions](#)

5.5.2.1 Detailed Description

5.5.2.2 STM32F4xx_System_Private_Includes

Collaboration diagram for STM32F4xx_System_Private_Includes:

Macros

- `#define HSE_VALUE ((uint32_t)25000000)`
- `#define HSI_VALUE ((uint32_t)16000000)`

5.5.2.2.1 Detailed Description

5.5.2.2.2 Macro Definition Documentation

5.5.2.2.2.1 HSE_VALUE

```
#define HSE_VALUE ((uint32_t)25000000)
```

Default value of the External oscillator in Hz

Definition at line 51 of file [system_stm32f4xx.c](#).

Referenced by [SystemCoreClockUpdate\(\)](#).

5.5.2.2.2 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)16000000)
```

Value of the Internal oscillator in Hz

Definition at line 55 of file [system_stm32f4xx.c](#).

Referenced by [SystemCoreClockUpdate\(\)](#).

5.5.2.3 STM32F4xx_System_Private_TypesDefinitions

Collaboration diagram for STM32F4xx_System_Private_TypesDefinitions:

5.5.2.4 STM32F4xx_System_Private_Defines

Collaboration diagram for STM32F4xx_System_Private_Defines:

5.5.2.5 STM32F4xx_System_Private_Macros

Collaboration diagram for STM32F4xx_System_Private_Macros:

5.5.2.6 STM32F4xx_System_Private_Variables

Collaboration diagram for STM32F4xx_System_Private_Variables:

Variables

- uint32_t [SystemCoreClock](#) = 16000000
- const uint8_t [AHBPrescTable](#) [16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t [APBPrescTable](#) [8] = {0, 0, 0, 0, 1, 2, 3, 4}

5.5.2.6.1 Detailed Description

5.5.2.6.2 Variable Documentation

5.5.2.6.2.1 AHBPrescTable

```
const uint8_t AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
```

Definition at line 138 of file [system_stm32f4xx.c](#).

Referenced by [SystemCoreClockUpdate\(\)](#).

5.5.2.6.2.2 APBPrescTable

```
const uint8_t APBPrescTable[8] = {0, 0, 0, 0, 1, 2, 3, 4}
```

Definition at line 139 of file [system_stm32f4xx.c](#).

5.5.2.6.2.3 SystemCoreClock

```
uint32_t SystemCoreClock = 16000000
```

Definition at line 137 of file [system_stm32f4xx.c](#).

Referenced by [main\(\)](#), and [SystemCoreClockUpdate\(\)](#).

5.5.2.7 STM32F4xx_System_Private_FunctionPrototypes

Collaboration diagram for STM32F4xx_System_Private_FunctionPrototypes:

5.5.2.8 STM32F4xx_System_Private_Functions

Collaboration diagram for STM32F4xx_System_Private_Functions:

Functions

- void [SystemInit](#) (void)
Setup the microcontroller system Initialize the FPU setting, vector table location and External memory configuration.
- void [SystemCoreClockUpdate](#) (void)
Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

5.5.2.8.1 Detailed Description

5.5.2.8.2 Function Documentation

5.5.2.8.2.1 SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
    void )
```

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Note

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the [HSI_VALUE\(*\)](#)
- If SYSCLK source is HSE, SystemCoreClock will contain the [HSE_VALUE\(**\)](#)
- If SYSCLK source is PLL, SystemCoreClock will contain the [HSE_VALUE\(**\)](#) or [HSI_VALUE\(*\)](#) multiplied/divided by the PLL factors.

(*) HSI_VALUE is a constant defined in [stm32f4xx_hal_conf.h](#) file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.

(**) HSE_VALUE is a constant defined in [stm32f4xx_hal_conf.h](#) file (its value depends on the application requirements), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

- The result of this function could be not correct when using fractional value for HSE crystal.

Definition at line [216](#) of file [system_stm32f4xx.c](#).

References [AHBPrescTable](#), [HSE_VALUE](#), [HSI_VALUE](#), and [SystemCoreClock](#).

5.5.2.8.2.2 SystemInit()

```
void SystemInit (
    void )
```

Setup the microcontroller system Initialize the FPU setting, vector table location and External memory configuration.

Definition at line [165](#) of file [system_stm32f4xx.c](#).

Chapter 6

Data Structure Documentation

6.1 abstract_conifer_channel Struct Reference

```
#include <uvfr_conifer.h>
```

Data Fields

- uint16_t `status_control_reg`
- uint16_t `hardware_mapping`
- uint16_t `iilm`
- uint16_t `duty`

6.1.1 Detailed Description

Definition at line 12 of file [uvfr_conifer.h](#).

6.1.2 Field Documentation

6.1.2.1 `duty`

```
uint16_t abstract_conifer_channel::duty
```

Definition at line 16 of file [uvfr_conifer.h](#).

6.1.2.2 `hardware_mapping`

```
uint16_t abstract_conifer_channel::hardware_mapping
```

Definition at line 14 of file [uvfr_conifer.h](#).

Referenced by [__attribute__\(\)](#), [coniferGetChannelCurrent\(\)](#), [coniferGetChannelFbck\(\)](#), [coniferInit\(\)](#), and [u19updatePduChannel\(\)](#).

6.1.2.3 ilim

```
uint16_t abstract_conifer_channel::ilim
```

Definition at line 15 of file [uvfr_conifer.h](#).

6.1.2.4 status_control_reg

```
uint16_t abstract_conifer_channel::status_control_reg
```

Definition at line 13 of file [uvfr_conifer.h](#).

Referenced by [coniferDisChannel\(\)](#), [coniferDisLoadShedding\(\)](#), [coniferEnChannel\(\)](#), [coniferEnLoadShedding\(\)](#), [coniferInit\(\)](#), [coniferToggleChannel\(\)](#), and [u19updatePduChannel\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_conifer.h](#)

6.2 access_control_info Union Reference

```
#include <uvfr_utils.h>
```

Collaboration diagram for access_control_info:

Data Fields

- struct [uv_mutex_info mutex](#)
- struct [uv_binary_semaphore_info bin_semaphore](#)
- struct [uv_semaphore_info semaphore](#)

6.2.1 Detailed Description

Definition at line 261 of file [uvfr_utils.h](#).

6.2.2 Field Documentation

6.2.2.1 bin_semaphore

```
struct uv\_binary\_semaphore\_info access_control_info::bin_semaphore
```

Definition at line 263 of file [uvfr_utils.h](#).

6.2.2.2 mutex

```
struct uv_mutex_info access_control_info::mutex
```

Definition at line 262 of file [uvfr_utils.h](#).

6.2.2.3 semaphore

```
struct uv_semaphore_info access_control_info::semaphore
```

Definition at line 264 of file [uvfr_utils.h](#).

The documentation for this union was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.3 bms_settings_t Struct Reference

```
#include <bms.h>
```

Data Fields

- uint32_t **BMS_CAN_timeout**
- uint16_t **max_cell_temp**
- uint16_t **min_cell_temp**
- uint16_t **min_soc**
- uint16_t **min_cell_voltage**
- uint16_t **max_cell_voltage**
- uint16_t **max_pack_voltage**
- uint16_t **min_pack_voltage**
- uint16_t **max_variance_between_cells**

6.3.1 Detailed Description

Definition at line 13 of file [bms.h](#).

6.3.2 Field Documentation

6.3.2.1 BMS_CAN_timeout

```
uint32_t bms_settings_t::BMS_CAN_timeout
```

Definition at line 14 of file [bms.h](#).

6.3.2.2 **max_cell_temp**

```
uint16_t bms_settings_t::max_cell_temp
```

Definition at line 15 of file [bms.h](#).

6.3.2.3 **max_cell_voltage**

```
uint16_t bms_settings_t::max_cell_voltage
```

Definition at line 19 of file [bms.h](#).

6.3.2.4 **max_pack_voltage**

```
uint16_t bms_settings_t::max_pack_voltage
```

Definition at line 20 of file [bms.h](#).

6.3.2.5 **max_variance_between_cells**

```
uint16_t bms_settings_t::max_variance_between_cells
```

Definition at line 22 of file [bms.h](#).

6.3.2.6 **min_cell_temp**

```
uint16_t bms_settings_t::min_cell_temp
```

Definition at line 16 of file [bms.h](#).

6.3.2.7 **min_cell_voltage**

```
uint16_t bms_settings_t::min_cell_voltage
```

Definition at line 18 of file [bms.h](#).

6.3.2.8 **min_pack_voltage**

```
uint16_t bms_settings_t::min_pack_voltage
```

Definition at line 21 of file [bms.h](#).

6.3.2.9 min_soc

```
uint16_t bms_settings_t::min_soc
```

Definition at line 17 of file [bms.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[bms.h](#)

6.4 CAN_Callback Struct Reference

Collaboration diagram for CAN_Callback:

Data Fields

- uint32_t [CAN_id](#)
- void * [function](#)
- struct [CAN_Callback](#) * [next](#)

6.4.1 Detailed Description

Definition at line 55 of file [can.c](#).

6.4.2 Field Documentation

6.4.2.1 CAN_id

```
uint32_t CAN_Callback::CAN_id
```

Definition at line 56 of file [can.c](#).

Referenced by [insertCANMessageHandler\(\)](#).

6.4.2.2 function

```
void* CAN_Callback::function
```

Definition at line 57 of file [can.c](#).

Referenced by [insertCANMessageHandler\(\)](#).

6.4.2.3 next

```
struct CAN_Callback* CAN_Callback::next
```

Definition at line 58 of file [can.c](#).

Referenced by [insertCANMessageHandler\(\)](#), and [nuke_hash_table\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Src/[can.c](#)

6.5 conifer_ch_setting Struct Reference

```
#include <uvfr_conifer.h>
```

Collaboration diagram for conifer_ch_setting:

Data Fields

- struct [abstract_conifer_channel](#) ch_dat
- uint16_t ch
- uint16_t reserved

6.5.1 Detailed Description

Definition at line 38 of file [uvfr_conifer.h](#).

6.5.2 Field Documentation

6.5.2.1 ch

```
uint16_t conifer_ch_setting::ch
```

Definition at line 40 of file [uvfr_conifer.h](#).

6.5.2.2 ch_dat

```
struct abstract_conifer_channel conifer_ch_setting::ch_dat
```

Definition at line 39 of file [uvfr_conifer.h](#).

6.5.2.3 reserved

```
uint16_t conifer_ch_setting::reserved
```

Definition at line 41 of file [uvfr_conifer.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_conifer.h](#)

6.6 conifer_hw_channel Struct Reference

```
#include <uvfr_conifer.h>
```

Data Fields

- uint16_t [ch_info_reg](#)

6.6.1 Detailed Description

Definition at line 181 of file [uvfr_conifer.h](#).

6.6.2 Field Documentation

6.6.2.1 ch_info_reg

```
uint16_t conifer_hw_channel::ch_info_reg
```

Definition at line 182 of file [uvfr_conifer.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_conifer.h](#)

6.7 conifer_settings Struct Reference

```
#include <uvfr_conifer.h>
```

Collaboration diagram for conifer_settings:

Data Fields

- `uint32_t conif_flags`
- `uint16_t OCP_flt_time`
- `uint16_t OCP_load_shed_time`
- `uint16_t sys_fault_current`
- `uint16_t sys_cont_current`
- `uint16_t sys_excess_I2T`
- `uint16_t sys_ocp_timeout`
- `uint16_t sys_max_voltage`
- `uint16_t sys_lv_threshold_voltage`
- `uint16_t sys_cutoff_voltage`
- `uint16_t regbusA_target_voltage`
- `uint16_t regbusA_fault_current`
- `uint16_t regbusA_continuos_current`
- `uint16_t regbusB_target_voltage`
- `uint16_t regbusB_fault_current`
- `uint16_t regbusB_continuos_current`
- `uint8_t n_ch`
- `uint8_t pdu_bus`
- `uint8_t dpms_bus`
- `uint8_t ecumaster_bus`
- `struct conifer_ch_setting ch_list [32]`

6.7.1 Detailed Description

Definition at line 104 of file [uvfr_conifer.h](#).

6.7.2 Field Documentation

6.7.2.1 ch_list

```
struct conifer_ch_setting conifer_settings::ch_list[32]
```

Definition at line 144 of file [uvfr_conifer.h](#).

6.7.2.2 conif_flags

```
uint32_t conifer_settings::conif_flags
```

Definition at line 105 of file [uvfr_conifer.h](#).

6.7.2.3 dpms_bus

```
uint8_t conifer_settings::dpms_bus
```

Definition at line 141 of file [uvfr_conifer.h](#).

6.7.2.4 ecumaster_bus

```
uint8_t conifer_settings::ecumaster_bus
```

Definition at line 142 of file [uvfr_conifer.h](#).

6.7.2.5 n_ch

```
uint8_t conifer_settings::n_ch
```

Definition at line 137 of file [uvfr_conifer.h](#).

6.7.2.6 OCP_flt_time

```
uint16_t conifer_settings::OCP_flt_time
```

Definition at line 108 of file [uvfr_conifer.h](#).

6.7.2.7 OCP_load_shed_time

```
uint16_t conifer_settings::OCP_load_shed_time
```

Definition at line 109 of file [uvfr_conifer.h](#).

6.7.2.8 pdu_bus

```
uint8_t conifer_settings::pdu_bus
```

Definition at line 138 of file [uvfr_conifer.h](#).

6.7.2.9 regbusA_continuos_current

```
uint16_t conifer_settings::regbusA_continuos_current
```

Definition at line 129 of file [uvfr_conifer.h](#).

6.7.2.10 regbusA_fault_current

```
uint16_t conifer_settings::regbusA_fault_current
```

Definition at line 128 of file [uvfr_conifer.h](#).

6.7.2.11 regbusA_target_voltage

```
uint16_t conifer_settings::regbusA_target_voltage
```

Definition at line 125 of file [uvfr_conifer.h](#).

6.7.2.12 regbusB_continous_current

```
uint16_t conifer_settings::regbusB_continous_current
```

Definition at line 136 of file [uvfr_conifer.h](#).

6.7.2.13 regbusB_fault_current

```
uint16_t conifer_settings::regbusB_fault_current
```

Definition at line 133 of file [uvfr_conifer.h](#).

6.7.2.14 regbusB_target_voltage

```
uint16_t conifer_settings::regbusB_target_voltage
```

Definition at line 132 of file [uvfr_conifer.h](#).

6.7.2.15 sys_cont_current

```
uint16_t conifer_settings::sys_cont_current
```

Definition at line 113 of file [uvfr_conifer.h](#).

6.7.2.16 sys_cutoff_voltage

```
uint16_t conifer_settings::sys_cutoff_voltage
```

Definition at line 124 of file [uvfr_conifer.h](#).

6.7.2.17 sys_excess_I2T

```
uint16_t conifer_settings::sys_excess_I2T
```

Definition at line 116 of file [uvfr_conifer.h](#).

6.7.2.18 sys_fault_current

```
uint16_t conifer_settings::sys_fault_current
```

Definition at line 112 of file [uvfr_conifer.h](#).

6.7.2.19 sys_lv_threshold_voltage

```
uint16_t conifer_settings::sys_lv_threshold_voltage
```

Definition at line 121 of file [uvfr_conifer.h](#).

6.7.2.20 sys_max_voltage

```
uint16_t conifer_settings::sys_max_voltage
```

Definition at line 120 of file [uvfr_conifer.h](#).

6.7.2.21 sys_ocp_timeout

```
uint16_t conifer_settings::sys_ocp_timeout
```

Definition at line 117 of file [uvfr_conifer.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_conifer.h](#)

6.8 conifer_state Struct Reference

```
#include <uvfr_conifer.h>
```

Data Fields

- uint32_t [status_flags](#)
- uint8_t [load_shedding](#)

6.8.1 Detailed Description

Definition at line 164 of file [uvfr_conifer.h](#).

6.8.2 Field Documentation

6.8.2.1 load_shedding

```
uint8_t conifer_state::load_shedding
```

Definition at line 166 of file [uvfr_conifer.h](#).

Referenced by [coniferDisLoadShedding\(\)](#), and [coniferEnLoadShedding\(\)](#).

6.8.2.2 status_flags

```
uint32_t conifer_state::status_flags
```

Definition at line 165 of file [uvfr_conifer.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_conifer.h](#)

6.9 daq_child_task Struct Reference

Collaboration diagram for daq_child_task:

Data Fields

- struct [daq_child_task](#) * `next_task`
- TaskHandle_t `meta_task_handle`
- uint32_t `period`
- [daq_param_list_node](#) * `param_list`

6.9.1 Detailed Description

Definition at line [37](#) of file [daq.c](#).

6.9.2 Field Documentation

6.9.2.1 meta_task_handle

TaskHandle_t `daq_child_task::meta_task_handle`

Definition at line [39](#) of file [daq.c](#).

Referenced by [startDaqSubTasks\(\)](#), and [stopDaqSubTasks\(\)](#).

6.9.2.2 next_task

struct [daq_child_task](#)* `daq_child_task::next_task`

Definition at line [38](#) of file [daq.c](#).

Referenced by [insertParamToRegister\(\)](#), and [startDaqSubTasks\(\)](#).

6.9.2.3 param_list

[daq_param_list_node](#)* `daq_child_task::param_list`

Definition at line [41](#) of file [daq.c](#).

Referenced by [daqSubTask\(\)](#), and [insertParamToRegister\(\)](#).

6.9.2.4 period

```
uint32_t daq_child_task::period
```

Definition at line 40 of file [daq.c](#).

Referenced by [daqSubTask\(\)](#), and [insertParamToRegister\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Src/[daq.c](#)

6.10 daq_loop_args Struct Reference

This struct holds info of what needs to be logged.

```
#include <daq.h>
```

Data Fields

- uint16_t total_params_logged
- uint8_t throttle_daq_to_preserve_performance
- uint8_t minimum_daq_period
- uint8_t can_channel
- uint8_t daq_child_priority

6.10.1 Detailed Description

This struct holds info of what needs to be logged.

Definition at line 97 of file [daq.h](#).

6.10.2 Field Documentation

6.10.2.1 can_channel

```
uint8_t daq_loop_args::can_channel
```

Definition at line 101 of file [daq.h](#).

Referenced by [initDaqTask\(\)](#).

6.10.2.2 daq_child_priority

```
uint8_t daq_loop_args::daq_child_priority
```

Definition at line 102 of file [daq.h](#).

Referenced by [initDaqTask\(\)](#), and [startDaqSubTasks\(\)](#).

6.10.2.3 minimum_daq_period

```
uint8_t daq_loop_args::minimum_daq_period
```

Definition at line 100 of file [daq.h](#).

6.10.2.4 throttle_daq_to_preserve_performance

```
uint8_t daq_loop_args::throttle_daq_to_preserve_performance
```

Definition at line 99 of file [daq.h](#).

6.10.2.5 total_params_logged

```
uint16_t daq_loop_args::total_params_logged
```

Definition at line 98 of file [daq.h](#).

Referenced by [configureDaqSubTasks\(\)](#), and [uvResetFlashToDefault\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[daq.h](#)

6.11 daq_msg Struct Reference

```
#include <daq.h>
```

Data Fields

- uint32_t [can_id](#)
- uint16_t [param](#) [4]
- uint8_t [type](#) [4]
- uint8_t [period](#)

6.11.1 Detailed Description

Definition at line 83 of file [daq.h](#).

6.11.2 Field Documentation

6.11.2.1 can_id

```
uint32_t daq_msg::can_id
```

Definition at line 84 of file [daq.h](#).

Referenced by [insertParamToRegister\(\)](#).

6.11.2.2 param

`uint16_t daq_msg::param[4]`

Which loggable param are we logging?

Definition at line 85 of file [daq.h](#).

Referenced by [insertParamToRegister\(\)](#).

6.11.2.3 period

`uint8_t daq_msg::period`

Time between transmissions in ms

Definition at line 87 of file [daq.h](#).

Referenced by [insertParamToRegister\(\)](#).

6.11.2.4 type

`uint8_t daq_msg::type[4]`

Datatype of the data

Definition at line 86 of file [daq.h](#).

Referenced by [insertParamToRegister\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[daq.h](#)

6.12 daq_param_list_node Struct Reference

Collaboration diagram for daq_param_list_node:

Data Fields

- struct [daq_param_list_node](#) * `next`
- `uint32_t can_id`
- `uint16_t param [4]`
- `uint8_t size [4]`

6.12.1 Detailed Description

Definition at line 28 of file [daq.c](#).

6.12.2 Field Documentation

6.12.2.1 can_id

```
uint32_t daq_param_list_node::can_id
```

Definition at line 31 of file [daq.c](#).

Referenced by [insertParamToRegister\(\)](#).

6.12.2.2 next

```
struct daq_param_list_node* daq_param_list_node::next
```

Definition at line 29 of file [daq.c](#).

Referenced by [insertParamToRegister\(\)](#).

6.12.2.3 param

```
uint16_t daq_param_list_node::param[4]
```

Definition at line 32 of file [daq.c](#).

Referenced by [insertParamToRegister\(\)](#).

6.12.2.4 size

```
uint8_t daq_param_list_node::size[4]
```

Definition at line 34 of file [daq.c](#).

Referenced by [insertParamToRegister\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Src/[daq.c](#)

6.13 driving_loop_args Struct Reference

```
#include <driving_loop.h>
```

Collaboration diagram for driving_loop_args:

Data Fields

- `uint32_t absolute_max_acc_pwr`
- `uint32_t absolute_max_motor_torque`
- `uint32_t absolute_max_accum_current`
- `uint32_t max_accum_current_5s`
- `uint16_t absolute_max_motor_rpm`
- `uint16_t regen_rpm_cutoff`
- `uint16_t min_apps_offset`
- `uint16_t max_apps_offset`
- `uint16_t min_apps_value`
- `uint16_t apps1_abs_max_val`
- `uint16_t apps1_abs_min_val`
- `uint16_t apps2_abs_min_val`
- `uint16_t apps2_abs_max_val`
- `uint16_t min_BPS_value`
- `uint16_t max_BPS_value`
- `uint16_t apps1_top`
- `uint16_t apps1_bottom`
- `uint16_t apps2_top`
- `uint16_t apps2_bottom`
- `uint16_t apps_plausibility_check_threshold`
- `uint16_t bps_plausibility_check_threshold`
- `uint16_t bps_implausibility_recovery_threshold`
- `uint16_t apps_implausibility_recovery_threshold`
- `uint8_t num_driving_modes`
- `uint8_t period`
- `uint8_t accum_regen_soc_threshold`
- `drivingMode dmodes [8]`

6.13.1 Detailed Description

Definition at line 108 of file [driving_loop.h](#).

6.13.2 Field Documentation

6.13.2.1 absolute_max_acc_pwr

```
uint32_t driving_loop_args::absolute_max_acc_pwr
```

Maximum possible accum power

Definition at line 109 of file [driving_loop.h](#).

6.13.2.2 absolute_max_accum_current

```
uint32_t driving_loop_args::absolute_max_accum_current
```

Max current (ADC reading)

Definition at line 111 of file [driving_loop.h](#).

6.13.2.3 absolute_max_motor_rpm

```
uint16_t driving_loop_args::absolute_max_motor_rpm
```

Max limit of RPM

Definition at line 115 of file [driving_loop.h](#).

6.13.2.4 absolute_max_motor_torque

```
uint32_t driving_loop_args::absolute_max_motor_torque
```

Max power output

Definition at line 110 of file [driving_loop.h](#).

6.13.2.5 accum_regen_soc_threshold

```
uint8_t driving_loop_args::accum_regen_soc_threshold
```

Vehicle will not regen if above this SOC

Definition at line 144 of file [driving_loop.h](#).

6.13.2.6 apps1_abs_max_val

```
uint16_t driving_loop_args::apps1_abs_max_val
```

for detecting disconnects and short circuits

Definition at line 123 of file [driving_loop.h](#).

Referenced by [performSafetyChecks\(\)](#).

6.13.2.7 apps1_abs_min_val

```
uint16_t driving_loop_args::apps1_abs_min_val
```

for detecting disconnects and short circuits

Definition at line 124 of file [driving_loop.h](#).

6.13.2.8 apps1_bottom

```
uint16_t driving_loop_args::apps1_bottom
```

Min APPS input value, representing 0% throttle

Definition at line 131 of file [driving_loop.h](#).

Referenced by [calculateThrottlePercentage\(\)](#), and [performSafetyChecks\(\)](#).

6.13.2.9 apps1_top

```
uint16_t driving_loop_args::apps1_top
```

Max APPS input value, representing 100% throttle

Definition at line 130 of file [driving_loop.h](#).

Referenced by [calculateThrottlePercentage\(\)](#), and [performSafetyChecks\(\)](#).

6.13.2.10 apps2_abs_max_val

```
uint16_t driving_loop_args::apps2_abs_max_val
```

Definition at line 126 of file [driving_loop.h](#).

Referenced by [performSafetyChecks\(\)](#).

6.13.2.11 apps2_abs_min_val

```
uint16_t driving_loop_args::apps2_abs_min_val
```

for detecting disconnects and short circuits

Definition at line 125 of file [driving_loop.h](#).

6.13.2.12 apps2_bottom

```
uint16_t driving_loop_args::apps2_bottom
```

Definition at line 134 of file [driving_loop.h](#).

Referenced by [performSafetyChecks\(\)](#).

6.13.2.13 apps2_top

```
uint16_t driving_loop_args::apps2_top
```

Definition at line 133 of file [driving_loop.h](#).

Referenced by [performSafetyChecks\(\)](#).

6.13.2.14 apps_implausibility_recovery_threshold

```
uint16_t driving_loop_args::apps_implausibility_recovery_threshold
```

Threshold for brake position

Definition at line 140 of file [driving_loop.h](#).

6.13.2.15 apps_plausibility_check_threshold

```
uint16_t driving_loop_args::apps_plausibility_check_threshold
```

Threshold for accelerator position with

Definition at line 136 of file [driving_loop.h](#).

Referenced by [performSafetyChecks\(\)](#).

6.13.2.16 bps_implausibility_recovery_threshold

```
uint16_t driving_loop_args::bps_implausibility_recovery_threshold
```

Threshold for acclerator pedal position to recover from APPS check

Definition at line 139 of file [driving_loop.h](#).

6.13.2.17 bps_plausibility_check_threshold

```
uint16_t driving_loop_args::bps_plausibility_check_threshold
```

Brake pressure threshold for APPS

Definition at line 137 of file [driving_loop.h](#).

6.13.2.18 dmodes

```
drivingMode driving_loop_args::dmodes[8]
```

These are various driving modes

Definition at line 147 of file [driving_loop.h](#).

6.13.2.19 max_accum_current_5s

```
uint32_t driving_loop_args::max_accum_current_5s
```

Current maximum for 10s

Definition at line 112 of file [driving_loop.h](#).

6.13.2.20 max_apps_offset

```
uint16_t driving_loop_args::max_apps_offset
```

maximum APPS offset

Definition at line 121 of file [driving_loop.h](#).

6.13.2.21 max_BPS_value

`uint16_t driving_loop_args::max_BPS_value`

are the brakes valid?

Definition at line 128 of file [driving_loop.h](#).

Referenced by [calculateBrakePercentage\(\)](#), and [performSafetyChecks\(\)](#).

6.13.2.22 min_apps_offset

`uint16_t driving_loop_args::min_apps_offset`

minimum APPS offset

Definition at line 120 of file [driving_loop.h](#).

6.13.2.23 min_apps_value

`uint16_t driving_loop_args::min_apps_value`

for detecting disconnects and short circuits

Definition at line 122 of file [driving_loop.h](#).

6.13.2.24 min_BPS_value

`uint16_t driving_loop_args::min_BPS_value`

are the brakes valid?

Definition at line 127 of file [driving_loop.h](#).

Referenced by [calculateBrakePercentage\(\)](#).

6.13.2.25 num_driving_modes

`uint8_t driving_loop_args::num_driving_modes`

How many modes are actually populated

Definition at line 142 of file [driving_loop.h](#).

6.13.2.26 period

`uint8_t driving_loop_args::period`

how often does the driving loop execute

Definition at line 143 of file [driving_loop.h](#).

6.13.2.27 regen_rpm_cutoff

```
uint16_t driving_loop_args::regen_rpm_cutoff
```

No regen below this rpm

Definition at line 116 of file [driving_loop.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.14 drivingLoopArgs Struct Reference

Arguments for the driving loop. The reason this is a struct passed in as an argument, rather than a bunch of global variables or constants is to allow the code to take settings from flash memory, therefore allowing the team to meet it's goal of having an actual GUI to change vehicle settings.

```
#include <driving_loop.h>
```

6.14.1 Detailed Description

Arguments for the driving loop. The reason this is a struct passed in as an argument, rather than a bunch of global variables or constants is to allow the code to take settings from flash memory, therefore allowing the team to meet it's goal of having an actual GUI to change vehicle settings.

The documentation for this struct was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.15 drivingMode Struct Reference

This is where the driving mode and the [drivingModeParams](#) are at.

```
#include <driving_loop.h>
```

Collaboration diagram for drivingMode:

Data Fields

- char [dm_name](#) [16]
- uint32_t [max_acc_pwr](#)
- uint32_t [max_motor_torque](#)
- uint32_t [max_current](#)
- uint16_t [flags](#)
- [drivingModeParams](#) [map_fn_params](#)
- uint8_t [control_map_fn](#)

6.15.1 Detailed Description

This is where the driving mode and the [drivingModeParams](#) are at.

Definition at line [85](#) of file [driving_loop.h](#).

6.15.2 Field Documentation

6.15.2.1 control_map_fn

```
uint8_t drivingMode::control_map_fn
```

Definition at line [95](#) of file [driving_loop.h](#).

6.15.2.2 dm_name

```
char drivingMode::dm_name[16]
```

Name of mode, 15 chars + /0

Definition at line [86](#) of file [driving_loop.h](#).

6.15.2.3 flags

```
uint16_t drivingMode::flags
```

Definition at line [92](#) of file [driving_loop.h](#).

6.15.2.4 map_fn_params

```
drivingModeParams drivingMode::map_fn_params
```

Definition at line [94](#) of file [driving_loop.h](#).

6.15.2.5 max_acc_pwr

```
uint32_t drivingMode::max_acc_pwr
```

Definition at line [87](#) of file [driving_loop.h](#).

6.15.2.6 max_current

```
uint32_t drivingMode::max_current
```

Definition at line [89](#) of file [driving_loop.h](#).

6.15.2.7 max_motor_torque

```
uint32_t drivingMode::max_motor_torque
```

Definition at line 88 of file [driving_loop.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.16 drivingModeParams Union Reference

this struct is designed to hold information about each drivingmode's map params

```
#include <driving_loop.h>
```

6.16.1 Detailed Description

this struct is designed to hold information about each drivingmode's map params

Definition at line 75 of file [driving_loop.h](#).

The documentation for this union was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.17 exp_torque_map_args Struct Reference

struct to hold parameters used in an exponential torque map

```
#include <driving_loop.h>
```

Data Fields

- int32_t offset
- float gamma

6.17.1 Detailed Description

struct to hold parameters used in an exponential torque map

Definition at line 56 of file [driving_loop.h](#).

6.17.2 Field Documentation

6.17.2.1 gamma

```
float exp_torque_map_args::gamma
```

Definition at line 58 of file [driving_loop.h](#).

6.17.2.2 offset

```
int32_t exp_torque_map_args::offset
```

Definition at line 57 of file [driving_loop.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.18 helper_task_args Union Reference

Collaboration diagram for helper_task_args:

Data Fields

- [tx_all_settings_args setting_tx_args](#)
- [tx_journal_args journal_tx_args](#)

6.18.1 Detailed Description

Definition at line 57 of file [uvfr_settings.c](#).

6.18.2 Field Documentation

6.18.2.1 journal_tx_args

```
tx_journal_args helper_task_args::journal_tx_args
```

Definition at line 59 of file [uvfr_settings.c](#).

6.18.2.2 setting_tx_args

```
tx_all_settings_args helper_task_args::setting_tx_args
```

Definition at line 58 of file [uvfr_settings.c](#).

Referenced by [sendAllSettingsWorker\(\)](#).

The documentation for this union was generated from the following file:

- Core/Src/[uvfr_settings.c](#)

6.19 linear_torque_map_args Struct Reference

```
#include <driving_loop.h>
```

Data Fields

- int32_t `offset`
- float `slope`

6.19.1 Detailed Description

Definition at line 48 of file [driving_loop.h](#).

6.19.2 Field Documentation

6.19.2.1 offset

```
int32_t linear_torque_map_args::offset
```

Definition at line 49 of file [driving_loop.h](#).

6.19.2.2 slope

```
float linear_torque_map_args::slope
```

Definition at line 50 of file [driving_loop.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.20 motor_controller_settings Struct Reference

```
#include <uvfr_settings.h>
```

Data Fields

- uint32_t `can_id_tx`
- uint32_t `can_id_rx`
- uint32_t `mc_CAN_timeout`
- uint8_t `proportional_gain`
- uint32_t `integral_time_constant`
- uint8_t `integral_memory_max`
- uint16_t `max_speed`
- uint16_t `max_current`
- uint16_t `cont_current`
- uint16_t `max_torque`
- uint16_t `max_motor_temp`
- uint16_t `warning_motor_temp`
- uint8_t `mc_bus`

6.20.1 Detailed Description

Definition at line 178 of file [uvfr_settings.h](#).

6.20.2 Field Documentation

6.20.2.1 can_id_rx

```
uint32_t motor_controller_settings::can_id_rx
```

Definition at line 181 of file [uvfr_settings.h](#).

6.20.2.2 can_id_tx

```
uint32_t motor_controller_settings::can_id_tx
```

Definition at line 180 of file [uvfr_settings.h](#).

6.20.2.3 cont_current

```
uint16_t motor_controller_settings::cont_current
```

Definition at line 190 of file [uvfr_settings.h](#).

6.20.2.4 integral_memory_max

```
uint8_t motor_controller_settings::integral_memory_max
```

Definition at line 186 of file [uvfr_settings.h](#).

6.20.2.5 integral_time_constant

```
uint32_t motor_controller_settings::integral_time_constant
```

Definition at line 185 of file [uvfr_settings.h](#).

6.20.2.6 max_current

```
uint16_t motor_controller_settings::max_current
```

Definition at line 189 of file [uvfr_settings.h](#).

6.20.2.7 max_motor_temp

```
uint16_t motor_controller_settings::max_motor_temp
```

Definition at line 192 of file [uvfr_settings.h](#).

6.20.2.8 max_speed

```
uint16_t motor_controller_settings::max_speed
```

Definition at line 188 of file [uvfr_settings.h](#).

6.20.2.9 max_torque

```
uint16_t motor_controller_settings::max_torque
```

Definition at line 191 of file [uvfr_settings.h](#).

6.20.2.10 mc_bus

```
uint8_t motor_controller_settings::mc_bus
```

Definition at line 196 of file [uvfr_settings.h](#).

6.20.2.11 mc_CAN_timeout

```
uint32_t motor_controller_settings::mc_CAN_timeout
```

Definition at line 182 of file [uvfr_settings.h](#).

6.20.2.12 proportional_gain

```
uint8_t motor_controller_settings::proportional_gain
```

Definition at line 183 of file [uvfr_settings.h](#).

6.20.2.13 warning_motor_temp

```
uint16_t motor_controller_settings::warning_motor_temp
```

Definition at line 193 of file [uvfr_settings.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_settings.h](#)

6.21 output_channel Struct Reference

```
#include <uvfr_vehicle_commands.h>
```

Data Fields

- `uint16_t I_max`
- `uint8_t flags`
- `uint8_t chID`

6.21.1 Detailed Description

Definition at line 29 of file [uvfr_vehicle_commands.h](#).

6.21.2 Field Documentation

6.21.2.1 chID

```
uint8_t output_channel::chID
```

Definition at line 32 of file [uvfr_vehicle_commands.h](#).

6.21.2.2 flags

```
uint8_t output_channel::flags
```

Definition at line 31 of file [uvfr_vehicle_commands.h](#).

6.21.2.3 I_max

```
uint16_t output_channel::I_max
```

Definition at line 30 of file [uvfr_vehicle_commands.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_vehicle_commands.h](#)

6.22 output_channel_settings Struct Reference

```
#include <uvfr_vehicle_commands.h>
```

Data Fields

- `uint16_t var`

6.22.1 Detailed Description

Definition at line 38 of file [uvfr_vehicle_commands.h](#).

6.22.2 Field Documentation

6.22.2.1 var

```
uint16_t output_channel_settings::var
```

Definition at line 39 of file [uvfr_vehicle_commands.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_vehicle_commands.h](#)

6.23 p_status Struct Reference

```
#include <uvfr_utils.h>
```

Data Fields

- `uv_status peripheral_status`
- `TickType_t activation_time`

6.23.1 Detailed Description

Definition at line 329 of file [uvfr_utils.h](#).

6.23.2 Field Documentation

6.23.2.1 activation_time

```
TickType_t p_status::activation_time
```

Definition at line 331 of file [uvfr_utils.h](#).

6.23.2.2 peripheral_status

```
uv_status p_status::peripheral_status
```

Definition at line 330 of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.24 rbnod Struct Reference

Node of a Red-Black binary search tree.

```
#include <rb_tree.h>
```

Collaboration diagram for rbnod:

Data Fields

- struct [rbnode](#) * left
- struct [rbnode](#) * right
- struct [rbnode](#) * parent
- void * [data](#)
- char [color](#)

6.24.1 Detailed Description

Node of a Red-Black binary search tree.

Definition at line 27 of file [rb_tree.h](#).

6.24.2 Field Documentation

6.24.2.1 color

```
char rbnod::color
```

The color of the node (internal use only)

Definition at line 32 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbDelete\(\)](#), and [rblInsert\(\)](#).

6.24.2.2 data

```
void* rbnodet::data
```

Pointer to some data contained by the tree

Definition at line 31 of file [rb_tree.h](#).

Referenced by [rbApplyNode\(\)](#), [rbCreate\(\)](#), [rbDelete\(\)](#), [rbFind\(\)](#), and [rbInsert\(\)](#).

6.24.2.3 left

```
struct rbnodet* rbnodet::left
```

Left sub-tree

Definition at line 28 of file [rb_tree.h](#).

Referenced by [rbApplyNode\(\)](#), [rbCreate\(\)](#), [rbDelete\(\)](#), [rbFind\(\)](#), [rbInsert\(\)](#), and [rbSuccessor\(\)](#).

6.24.2.4 parent

```
struct rbnodet* rbnodet::parent
```

Parent of node

Definition at line 30 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbDelete\(\)](#), [rbInsert\(\)](#), and [rbSuccessor\(\)](#).

6.24.2.5 right

```
struct rbnodet* rbnodet::right
```

Right sub-tree

Definition at line 29 of file [rb_tree.h](#).

Referenced by [rbApplyNode\(\)](#), [rbCreate\(\)](#), [rbDelete\(\)](#), [rbFind\(\)](#), [rbInsert\(\)](#), and [rbSuccessor\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[rb_tree.h](#)

6.25 rbtree Struct Reference

struct representing a binary search tree

```
#include <rb_tree.h>
```

Collaboration diagram for rbtree:

Data Fields

- int(* `compare`)(const void *, const void *)
- void(* `print`)(void *)
- void(* `destroy`)(void *)
- `rbnode` `root`
- `rbnode` `nil`
- `rbnode` * `min`
- int `count`

6.25.1 Detailed Description

struct representing a binary search tree

Definition at line 39 of file [rb_tree.h](#).

6.25.2 Field Documentation

6.25.2.1 compare

```
int(* rbtree::compare) (const void *, const void *)
```

Function to compare between two different nodes

Definition at line 40 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbFind\(\)](#), and [rblInsert\(\)](#).

6.25.2.2 count

```
int rbtree::count
```

number of items stored in the tree

Definition at line 53 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbDelete\(\)](#), and [rblInsert\(\)](#).

6.25.2.3 destroy

```
void(* rbtree::destroy) (void *)
```

Destructor function for whatever data is stored in the tree

Definition at line 42 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbDelete\(\)](#), and [rblInsert\(\)](#).

6.25.2.4 min

`rbnode* rbtree::min`

Pointer to minimum element

Definition at line 50 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbDelete\(\)](#), and [rblInsert\(\)](#).

6.25.2.5 nil

`rbnode rbtree::nil`

The "NIL" node of the tree, used to avoid fucked null errors

Definition at line 45 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#).

6.25.2.6 print

`void(* rbtree::print) (void *)`

For printing purposes. NOT YET IMPLEMENTED ON ANY SYSTEMS IN THE CAR

Definition at line 41 of file [rb_tree.h](#).

6.25.2.7 root

`rbnode rbtree::root`

Root of actual tree

Definition at line 44 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[rb_tree.h](#)

6.26 s_curve_torque_map_args Struct Reference

struct for s-curve parameters for torque

```
#include <driving_loop.h>
```

Data Fields

- int32_t a
- int32_t b
- int32_t c [16]

6.26.1 Detailed Description

struct for s-curve parameters for torque

Definition at line 66 of file [driving_loop.h](#).

6.26.2 Field Documentation

6.26.2.1 a

int32_t s_curve_torque_map_args::a

Definition at line 67 of file [driving_loop.h](#).

6.26.2.2 b

int32_t s_curve_torque_map_args::b

Definition at line 68 of file [driving_loop.h](#).

6.26.2.3 c

int32_t s_curve_torque_map_args::c[16]

Definition at line 69 of file [driving_loop.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[driving_loop.h](#)

6.27 setting_helper_task Struct Reference

Collaboration diagram for setting_helper_task:

Data Fields

- TaskHandle_t [meta_task_handle](#)
- uint32_t [status](#)
- [helper_task_args args](#)

6.27.1 Detailed Description

Definition at line 65 of file [uvfr_settings.c](#).

6.27.2 Field Documentation

6.27.2.1 args

```
helper_task_args setting_helper_task::args
```

Definition at line 71 of file [uvfr_settings.c](#).

Referenced by [sendAllSettingsWorker\(\)](#).

6.27.2.2 meta_task_handle

```
TaskHandle_t setting_helper_task::meta_task_handle
```

Definition at line 66 of file [uvfr_settings.c](#).

Referenced by [sendAllSettingsWorker\(\)](#).

6.27.2.3 status

```
uint32_t setting_helper_task::status
```

Definition at line 68 of file [uvfr_settings.c](#).

Referenced by [sendAllSettingsWorker\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Src/[uvfr_settings.c](#)

6.28 state_change_daemon_args Struct Reference

Data Fields

- TaskHandle_t [meta_task_handle](#)

6.28.1 Detailed Description

Definition at line 64 of file [uvfr_state_engine.c](#).

6.28.2 Field Documentation

6.28.2.1 meta_task_handle

TaskHandle_t state_change_daemon_args::meta_task_handle

Definition at line 65 of file [uvfr_state_engine.c](#).

Referenced by [changeVehicleState\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Src/[uvfr_state_engine.c](#)

6.29 task_management_info Struct Reference

Struct to contain data about a parent task.

```
#include <uvfr_state_engine.h>
```

Data Fields

- TaskHandle_t [task_handle](#)
- QueueHandle_t [parent_msg_queue](#)

6.29.1 Detailed Description

Struct to contain data about a parent task.

This contains the information required for the child task to communicate with it's parent.

This will be a queue, since one parent task can in theory have several child tasks

Definition at line 161 of file [uvfr_state_engine.h](#).

6.29.2 Field Documentation

6.29.2.1 parent_msg_queue

QueueHandle_t task_management_info::parent_msg_queue

Definition at line 163 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskManager\(\)](#).

6.29.2.2 task_handle

TaskHandle_t task_management_info::task_handle

Actual handle of parent

Definition at line 162 of file [uvfr_state_engine.h](#).

Referenced by [uvSVCTaskManager\(\)](#), and [uvTaskManager\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_state_engine.h](#)

6.30 task_status_block Struct Reference

Information about the task.

```
#include <uvfr_state_engine.h>
```

Data Fields

- uint32_t [task_high_water_mark](#)
- TickType_t [task_report_time](#)

6.30.1 Detailed Description

Information about the task.

Definition at line 169 of file [uvfr_state_engine.h](#).

6.30.2 Field Documentation

6.30.2.1 task_high_water_mark

uint32_t task_status_block::task_high_water_mark

Definition at line 170 of file [uvfr_state_engine.h](#).

6.30.2.2 task_report_time

TickType_t task_status_block::task_report_time

Definition at line 171 of file [uvfr_state_engine.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_state_engine.h](#)

6.31 tx_all_settings_args Struct Reference

Data Fields

- `uint8_t * sblock_origin`

6.31.1 Detailed Description

These are arguments passed to the "Transmit All Settings Over CANbus" Subroutine.

Definition at line 41 of file [uvfr_settings.c](#).

6.31.2 Field Documentation

6.31.2.1 sblock_origin

`uint8_t* tx_all_settings_args::sblock_origin`

Definition at line 42 of file [uvfr_settings.c](#).

Referenced by [sendAllSettingsWorker\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Src/[uvfr_settings.c](#)

6.32 tx_journal_args Struct Reference

Data Fields

- `uint32_t start_time`
- `uint32_t end_time`

6.32.1 Detailed Description

Definition at line 49 of file [uvfr_settings.c](#).

6.32.2 Field Documentation

6.32.2.1 end_time

`uint32_t tx_journal_args::end_time`

Definition at line 51 of file [uvfr_settings.c](#).

6.32.2.2 start_time

```
uint32_t tx_journal_args::start_time
```

Definition at line 50 of file [uvfr_settings.c](#).

The documentation for this struct was generated from the following file:

- Core/Src/[uvfr_settings.c](#)

6.33 uv19_pdu_settings Struct Reference

```
#include <pdu.h>
```

Data Fields

- uint32_t [PDU_rx_addr](#)
- uint32_t [PDU_tx_addr](#)
- uint32_t [expected_period](#)

6.33.1 Detailed Description

Definition at line 16 of file [pdu.h](#).

6.33.2 Field Documentation

6.33.2.1 expected_period

```
uint32_t uv19_pdu_settings::expected_period
```

Definition at line 19 of file [pdu.h](#).

6.33.2.2 PDU_rx_addr

```
uint32_t uv19_pdu_settings::PDU_rx_addr
```

Definition at line 17 of file [pdu.h](#).

6.33.2.3 PDU_tx_addr

```
uint32_t uv19_pdu_settings::PDU_tx_addr
```

Definition at line 18 of file [pdu.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[pdu.h](#)

6.34 uv_binary_semaphore_info Struct Reference

```
#include <uvfr_utils.h>
```

Data Fields

- SemaphoreHandle_t [handle](#)

6.34.1 Detailed Description

Definition at line [251](#) of file [uvfr_utils.h](#).

6.34.2 Field Documentation

6.34.2.1 handle

```
SemaphoreHandle_t uv_binary_semaphore_info::handle
```

Definition at line [252](#) of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.35 uv_CAN_msg Struct Reference

Representative of a CAN message.

```
#include <uvfr_utils.h>
```

Data Fields

- uint8_t [flags](#)
- uint8_t [dlc](#)
- uint32_t [msg_id](#)
- uint8_t [data](#) [8]

6.35.1 Detailed Description

Representative of a CAN message.

Definition at line [282](#) of file [uvfr_utils.h](#).

6.35.2 Field Documentation

6.35.2.1 data

```
uint8_t uv_CAN_msg::data[8]
```

The actual data packet contained within the CAN message

Definition at line 289 of file [uvfr_utils.h](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [BMS_msg1\(\)](#), [BMS_msg2\(\)](#), [CANbusTxSvcDaemon\(\)](#), [flushLogsToCAN\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [handleDiagnosticMsg\(\)](#), [handleIncomingLaptopMsg\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [MC_SetAndVerify_Param\(\)](#), [Parse_Bamocar_Response\(\)](#), [ProcessMotorControllerResponse\(\)](#), [sendTorqueToMotorController\(\)](#), [tempMonitorTask\(\)](#), [testfunc2\(\)](#), [testfunc3\(\)](#), [u19updatePduChannel\(\)](#), [uvSendSpecificParam\(\)](#), and [uvSettingsProgrammerTask\(\)](#).

6.35.2.2 dlc

```
uint8_t uv_CAN_msg::dlc
```

Data Length Code, representing how many bytes of data are present

Definition at line 287 of file [uvfr_utils.h](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [flushLogsToCAN\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [handleIncomingLaptopMsg\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [Parse_Bamocar_Response\(\)](#), [ProcessMotorControllerResponse\(\)](#), [sendTorqueToMotorController\(\)](#), [tempMonitorTask\(\)](#), and [uvSendSpecificParam\(\)](#).

6.35.2.3 flags

```
uint8_t uv_CAN_msg::flags
```

Bitfield that contains some basic information about the message: -Bit 0: Is the message an extended ID message, or a standard ID message? 1 For extended. -Bits 1:2 Which CANbus is being used to send the message? 00-> whatever the default is 01 -> CAN1 10 -> CAN2 11-> CAN3 (doesnt exist yet). Will default to CAN1 if all zeros

Definition at line 283 of file [uvfr_utils.h](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [flushLogsToCAN\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [initDaqTask\(\)](#), [initPDU\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [sendTorqueToMotorController\(\)](#), [tempMonitorTask\(\)](#), [testfunc2\(\)](#), [testfunc3\(\)](#), and [uvSendSpecificParam\(\)](#).

6.35.2.4 msg_id

```
uint32_t uv_CAN_msg::msg_id
```

The ID of a message

Definition at line 288 of file [uvfr_utils.h](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [flushLogsToCAN\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [sendTorqueToMotorController\(\)](#), [tempMonitorTask\(\)](#), and [uvSendSpecificParam\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.36 uv_imd_settings Struct Reference

```
#include <imd.h>
```

Data Fields

- `uint16_t min_isolation_resistances`
- `uint16_t expected_isolation_capacitances`
- `uint16_t max_imd_temperature`

6.36.1 Detailed Description

Definition at line 29 of file [imd.h](#).

6.36.2 Field Documentation

6.36.2.1 `expected_isolation_capacitances`

```
uint16_t uv_imd_settings::expected_isolation_capacitances
```

Definition at line 31 of file [imd.h](#).

6.36.2.2 `max_imd_temperature`

```
uint16_t uv_imd_settings::max_imd_temperature
```

Definition at line 32 of file [imd.h](#).

6.36.2.3 `min_isolation_resistances`

```
uint16_t uv_imd_settings::min_isolation_resistances
```

Definition at line 30 of file [imd.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[imd.h](#)

6.37 uv_init_struct Struct Reference

```
#include <uvfr_utils.h>
```

Data Fields

- `bool use_default_settings`

6.37.1 Detailed Description

contains info relevant to initializing the vehicle

Definition at line 296 of file [uvfr_utils.h](#).

6.37.2 Field Documentation

6.37.2.1 `use_default_settings`

`bool uv_init_struct::use_default_settings`

Definition at line 297 of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.38 `uv_init_task_args` Struct Reference

Struct designed to act like the `uv_task_info` struct, but for the initialisation tasks. As a result it takes fewer arguments.

```
#include <uvfr_utils.h>
```

Data Fields

- `void * specific_args`
- `QueueHandle_t init_info_queue`
- `TaskHandle_t meta_task_handle`

6.38.1 Detailed Description

Struct designed to act like the `uv_task_info` struct, but for the initialisation tasks. As a result it takes fewer arguments.

Definition at line 341 of file [uvfr_utils.h](#).

6.38.2 Field Documentation

6.38.2.1 init_info_queue

QueueHandle_t uv_init_task_args::init_info_queue

Definition at line 343 of file [uvfr_utils.h](#).

Referenced by [BMS_Init\(\)](#), [initIMD\(\)](#), [MC_Startup\(\)](#), and [uvInit\(\)](#).

6.38.2.2 meta_task_handle

TaskHandle_t uv_init_task_args::meta_task_handle

Definition at line 344 of file [uvfr_utils.h](#).

Referenced by [BMS_Init\(\)](#), [initIMD\(\)](#), and [uvInit\(\)](#).

6.38.2.3 specific_args

void* uv_init_task_args::specific_args

Definition at line 342 of file [uvfr_utils.h](#).

Referenced by [uvInit\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.39 uv_init_task_response Struct Reference

Struct representing the response of one of the initialization tasks.

```
#include <uvfr_utils.h>
```

Data Fields

- [uv_status status](#)
- [uv_ext_device_id device](#)
- [uint8_t nchar](#)
- [char * errmsg](#)

6.39.1 Detailed Description

Struct representing the response of one of the initialization tasks.

Is returned in the initialization queue, and is read by [uvInit\(\)](#) to determine whether the initialization of the internal device has failed or succeeded.

Definition at line 367 of file [uvfr_utils.h](#).

6.39.2 Field Documentation

6.39.2.1 device

```
uv_ext_device_id uv_init_task_response::device
```

Definition at line 369 of file [uvfr_utils.h](#).

Referenced by [MC_Startup\(\)](#), and [uvInit\(\)](#).

6.39.2.2 errmsg

```
char* uv_init_task_response::errmsg
```

Definition at line 371 of file [uvfr_utils.h](#).

Referenced by [uvInit\(\)](#).

6.39.2.3 nchar

```
uint8_t uv_init_task_response::nchar
```

Definition at line 370 of file [uvfr_utils.h](#).

Referenced by [uvInit\(\)](#).

6.39.2.4 status

```
uv_status uv_init_task_response::status
```

Definition at line 368 of file [uvfr_utils.h](#).

Referenced by [MC_Startup\(\)](#), and [uvInit\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.40 uv_internal_params Struct Reference

Data used by the uvfr_utils library to do what it needs to do :)

```
#include <uvfr_utils.h>
```

Collaboration diagram for uv_internal_params:

Data Fields

- `uv_init_struct * init_params`
- `uv_vehicle_settings * vehicle_settings`
- `p_status peripheral_status [8]`
- `uint16_t e_code [8]`

6.40.1 Detailed Description

Data used by the uvfr_utils library to do what it needs to do :)

This is a global variable that is initialized at some point at launch

Definition at line 353 of file [uvfr_utils.h](#).

6.40.2 Field Documentation

6.40.2.1 e_code

```
uint16_t uv_internal_params::e_code[8]
```

Definition at line 357 of file [uvfr_utils.h](#).

6.40.2.2 init_params

```
uv_init_struct* uv_internal_params::init_params
```

Definition at line 354 of file [uvfr_utils.h](#).

6.40.2.3 peripheral_status

```
p_status uv_internal_params::peripheral_status[8]
```

Definition at line 356 of file [uvfr_utils.h](#).

6.40.2.4 vehicle_settings

```
uv_vehicle_settings* uv_internal_params::vehicle_settings
```

Definition at line 355 of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.41 uv_mutex_info Struct Reference

```
#include <uvfr_utils.h>
```

Data Fields

- SemaphoreHandle_t [handle](#)

6.41.1 Detailed Description

Definition at line [246](#) of file [uvfr_utils.h](#).

6.41.2 Field Documentation

6.41.2.1 handle

```
SemaphoreHandle_t uv_mutex_info::handle
```

Definition at line [247](#) of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.42 uv_os_settings Struct Reference

Settings that dictate state engine behavior.

```
#include <uvfr_state_engine.h>
```

Data Fields

- TickType_t [svc_task_manager_period](#)
- TickType_t [task_manager_period](#)
- TickType_t [max_svc_task_period](#)
- TickType_t [max_task_period](#)
- TickType_t [min_task_period](#)
- float [task_overshoot_margin_noncrit](#)
- float [task_overshoot_margin_crit](#)
- float [task_throttle_increment](#)
- uint16_t [os_flags](#)

6.42.1 Detailed Description

Settings that dictate state engine behavior.

Definition at line [184](#) of file [uvfr_state_engine.h](#).

6.42.2 Field Documentation

6.42.2.1 max_svc_task_period

```
TickType_t uv_os_settings::max_svc_task_period
```

Definition at line 187 of file [uvfr_state_engine.h](#).

6.42.2.2 max_task_period

```
TickType_t uv_os_settings::max_task_period
```

Definition at line 188 of file [uvfr_state_engine.h](#).

6.42.2.3 min_task_period

```
TickType_t uv_os_settings::min_task_period
```

Definition at line 189 of file [uvfr_state_engine.h](#).

6.42.2.4 os_flags

```
uint16_t uv_os_settings::os_flags
```

Definition at line 194 of file [uvfr_state_engine.h](#).

6.42.2.5 svc_task_manager_period

```
TickType_t uv_os_settings::svc_task_manager_period
```

Definition at line 185 of file [uvfr_state_engine.h](#).

6.42.2.6 task_manager_period

```
TickType_t uv_os_settings::task_manager_period
```

Definition at line 186 of file [uvfr_state_engine.h](#).

6.42.2.7 task_overshoot_margin_crit

```
float uv_os_settings::task_overshoot_margin_crit
```

Definition at line 191 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskManager\(\)](#).

6.42.2.8 task_overshoot_margin_noncrit

```
float uv_os_settings::task_overshoot_margin_noncrit
```

Definition at line 190 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskManager\(\)](#).

6.42.2.9 task_throttle_increment

```
float uv_os_settings::task_throttle_increment
```

Definition at line 192 of file [uvfr_state_engine.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_state_engine.h](#)

6.43 uv_persistent_data_frame Struct Reference

```
#include <odometer.h>
```

Data Fields

- `uint64_t total_vehicle_uptime`
- `uint64_t total_time_driving`
- `uint64_t total_distance_cm`

6.43.1 Detailed Description

Definition at line 13 of file [odometer.h](#).

6.43.2 Field Documentation

6.43.2.1 total_distance_cm

```
uint64_t uv_persistent_data_frame::total_distance_cm
```

Definition at line 16 of file [odometer.h](#).

6.43.2.2 total_time_driving

```
uint64_t uv_persistent_data_frame::total_time_driving
```

Definition at line 15 of file [odometer.h](#).

6.43.2.3 total_vehicle_uptime

```
uint64_t uv_persistent_data_frame::total_vehicle_uptime
```

Definition at line 14 of file [odometer.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[odometer.h](#)

6.44 uv_scd_response Struct Reference

```
#include <uvfr_state_engine.h>
```

Data Fields

- enum [uv_scd_response_e response_val](#)
- [uv_task_id meta_id](#)

6.44.1 Detailed Description

Definition at line 121 of file [uvfr_state_engine.h](#).

6.44.2 Field Documentation

6.44.2.1 meta_id

```
uv_task_id uv_scd_response::meta_id
```

Definition at line 123 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#), [killSelf\(\)](#), and [suspendSelf\(\)](#).

6.44.2.2 response_val

```
enum uv_scd_response_e uv_scd_response::response_val
```

Definition at line 122 of file [uvfr_state_engine.h](#).

Referenced by [killSelf\(\)](#), and [suspendSelf\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_state_engine.h](#)

6.45 uv_semaphore_info Struct Reference

```
#include <uvfr_utils.h>
```

Data Fields

- SemaphoreHandle_t [handle](#)

6.45.1 Detailed Description

Definition at line [256](#) of file [uvfr_utils.h](#).

6.45.2 Field Documentation

6.45.2.1 handle

```
SemaphoreHandle_t uv_semaphore_info::handle
```

Definition at line [257](#) of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.46 uv_task_info Struct Reference

This struct is designed to hold necessary information about an RTOS task that will be managed by [uvfr_state_engine](#).

```
#include <uvfr_state_engine.h>
```

Collaboration diagram for [uv_task_info](#):

Data Fields

- [uv_task_id](#) [task_id](#)
- char * [task_name](#)
- [uv_timespan_ms](#) [task_period](#)
- [uv_timespan_ms](#) [deletion_delay](#)
- TaskFunction_t [task_function](#)
- osPriority [task_priority](#)
- uint32_t [stack_size](#)
- [uv_task_status](#) [task_state](#)
- TaskHandle_t [task_handle](#)
- [uv_task_cmd](#) [cmd_data](#)
- void * [task_args](#)
- struct [uv_task_info_t](#) * [parent](#)
- [task_management_info](#) * [tmi](#)
- MessageBufferHandle_t [task_rx_mailbox](#)
- TickType_t [last_execution_time](#)
- uint16_t [active_states](#)
- uint16_t [deletion_states](#)
- uint16_t [suspension_states](#)
- uint16_t [task_flags](#)
- uint8_t [throttle_factor](#)

6.46.1 Detailed Description

This struct is designed to hold necessary information about an RTOS task that will be managed by `uvfr_state_engine`.

Pay close attention, because this is one of the most cursed structs in the project, as well as one of the most important

Definition at line 227 of file `uvfr_state_engine.h`.

6.46.2 Field Documentation

6.46.2.1 active_states

```
uint16_t uv_task_info::active_states
```

Definition at line 258 of file `uvfr_state_engine.h`.

Referenced by `_stateChangeDaemon()`, `_uvValidateSpecificTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initRTDtask()`, `initTempMonitor()`, `uvConfigSettingTask()`, `uvCreateServiceTask()`, `uvCreateTask()`, and `uvInit()`.

6.46.2.2 cmd_data

```
uv_task_cmd uv_task_info::cmd_data
```

how we communicate with the task rn - THIS SUCKS SO BAD

Definition at line 248 of file `uvfr_state_engine.h`.

Referenced by `CANbusRxSvcDaemon()`, `CANbusTxSvcDaemon()`, `daqMasterTask()`, `killSelf()`, `odometerTask()`, `rtdTask()`, `StartDrivingLoop()`, `suspendSelf()`, `tempMonitorTask()`, `uvDeleteSVCTask()`, `uvDeleteTask()`, `uvSettingsProgrammerTask()`, `uvSuspendTask()`, and `uvTaskManager()`.

6.46.2.3 deletion_delay

```
uv_timespan_ms uv_task_info::deletion_delay
```

If deferred deletion is enabled, how long to wait before we delete task?

Definition at line 232 of file `uvfr_state_engine.h`.

6.46.2.4 deletion_states

```
uint16_t uv_task_info::deletion_states
```

Definition at line 259 of file `uvfr_state_engine.h`.

Referenced by `_stateChangeDaemon()`, `_uvValidateSpecificTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initRTDtask()`, `initTempMonitor()`, `uvConfigSettingTask()`, `uvCreateServiceTask()`, and `uvCreateTask()`.

6.46.2.5 last_execution_time

```
TickType_t uv_task_info::last_execution_time
```

Definition at line 256 of file [uvfr_state_engine.h](#).

Referenced by [uvStartTask\(\)](#), [uvTaskManager\(\)](#), and [uvTaskPeriodEnd\(\)](#).

6.46.2.6 parent

```
struct uv_task_info_t* uv_task_info::parent
```

info about the parent of the task

Definition at line 252 of file [uvfr_state_engine.h](#).

Referenced by [uvCreateServiceTask\(\)](#), and [uvCreateTask\(\)](#).

6.46.2.7 stack_size

```
uint32_t uv_task_info::stack_size
```

Number of words allocated to the stack of the task

Definition at line 238 of file [uvfr_state_engine.h](#).

Referenced by [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [uvConfigSettingTask\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvStartStateMachine\(\)](#), [uvStartSVCTask\(\)](#), and [uvStartTask\(\)](#).

6.46.2.8 suspension_states

```
uint16_t uv_task_info::suspension_states
```

Definition at line 260 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#), [_uvValidateSpecificTask\(\)](#), [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [uvConfigSettingTask\(\)](#), [uvCreateServiceTask\(\)](#), and [uvCreateTask\(\)](#).

6.46.2.9 task_args

```
void* uv_task_info::task_args
```

arguments for the specific task, this is where we will likely pass in task settings

Definition at line 250 of file [uvfr_state_engine.h](#).

Referenced by [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [uvConfigSettingTask\(\)](#), and [uvStartSVCTask\(\)](#).

6.46.2.10 task_flags

`uint16_t uv_task_info::task_flags`

- Bits 0:1 - | Task MGMT | Vehicle Application task - 01 | Periodic SVC Task - 10 | Dormant SVC Task - 11
- Bit 2 - Log task start + stop time
- Bit 3 - Log mem usage
- Bit 4 - SCD ignore flag (only use if task is application layer)
- Bit 5 - is parent
- Bit 6 - is child
- Bit 7 - is orphaned
- Bit 8 - error in child task
- Bit 9 - awaiting deferred deletion
- Bit 10 - deferred deletion enabled
- Bits 11:12 - Deadline firmness | No enforcement - 00 | Gradual Priority Incrementation - 01 | Firm deadline 10 | Critical Deadline - 11
- Bit 13 - mission critical, if this specific task crashes, the car will not continue to run
- Bit 14 - Task currently delaying, either by vTaskDelay or vTaskDelayUntil

Definition at line 262 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvLateTaskHandler\(\)](#), [uvScheduleTaskDeletion\(\)](#), [uvStartStateMachine\(\)](#), [uvStartSVCTask\(\)](#), [uvTaskCrashHandler\(\)](#), and [uvTaskManager\(\)](#).

6.46.2.11 task_function

`TaskFunction_t uv_task_info::task_function`

Pointer to function that implements the task

Definition at line 234 of file [uvfr_state_engine.h](#).

Referenced by [_uvValidateSpecificTask\(\)](#), [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [uvConfigSettingTask\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvInit\(\)](#), [uvStartStateMachine\(\)](#), [uvStartSVCTask\(\)](#), and [uvStartTask\(\)](#).

6.46.2.12 task_handle

`TaskHandle_t uv_task_info::task_handle`

Handle of freeRTOS task control block

Definition at line 246 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#), [CANbusRxSvcDaemon\(\)](#), [killSelf\(\)](#), [suspendSelf\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvDeleteSVCTask\(\)](#), [uvDeleteTask\(\)](#), [uvSettingsProgrammerTask\(\)](#), [uvStartSVCTask\(\)](#), [uvStartTask\(\)](#), [uvSuspendTask\(\)](#), and [uvTaskManager\(\)](#).

6.46.2.13 task_id

```
uv_task_id uv_task_info::task_id
```

Detailed description after the member

Definition at line 228 of file [uvfr_state_engine.h](#).

Referenced by [killSelf\(\)](#), [logVehicleFault\(\)](#), [suspendSelf\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvDeleteTask\(\)](#), [uvScheduleTaskDeletion\(\)](#), [uvStartTask\(\)](#), [uvSuspendTask\(\)](#), and [uvTaskPeriodEnd\(\)](#).

6.46.2.14 task_name

```
char* uv_task_info::task_name
```

Detailed description after the member

Definition at line 229 of file [uvfr_state_engine.h](#).

Referenced by [_uvValidateSpecificTask\(\)](#), [compareTaskByName\(\)](#), [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [logVehicleFault\(\)](#), [uvConfigSettingTask\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvInit\(\)](#), [uvStartStateMachine\(\)](#), [uvStartSVCTask\(\)](#), and [uvStartTask\(\)](#).

6.46.2.15 task_period

```
uv_timespan_ms uv_task_info::task_period
```

Maximum period between task execution

Definition at line 231 of file [uvfr_state_engine.h](#).

Referenced by [daqMasterTask\(\)](#), [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [odometerTask\(\)](#), [StartDrivingLoop\(\)](#), [tempMonitorTask\(\)](#), [uvConfigSettingTask\(\)](#), [uvLateTaskHandler\(\)](#), [uvStartStateMachine\(\)](#), [uvTaskManager\(\)](#), and [uvTaskPeriodEnd\(\)](#).

6.46.2.16 task_priority

```
osPriority uv_task_info::task_priority
```

Priority of the task. Int between 0 and 7

Definition at line 235 of file [uvfr_state_engine.h](#).

Referenced by [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [initOdometer\(\)](#), [initRTDtask\(\)](#), [initTempMonitor\(\)](#), [uvConfigSettingTask\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvStartSVCTask\(\)](#), and [uvStartTask\(\)](#).

6.46.2.17 task_rx_mailbox

```
MessageBufferHandle_t uv_task_info::task_rx_mailbox
```

Incoming messages for this task

Definition at line 255 of file [uvfr_state_engine.h](#).

6.46.2.18 task_state

```
uv_task_status uv_task_info::task_state
```

Definition at line 243 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#), [killSelf\(\)](#), [logVehicleFault\(\)](#), [suspendSelf\(\)](#), [uvCreateServiceTask\(\)](#), [uvCreateTask\(\)](#), [uvDeleteSVCTask\(\)](#), [uvDeleteTask\(\)](#), [uvScheduleTaskDeletion\(\)](#), [uvStartSVCTask\(\)](#), [uvStartTask\(\)](#), [uvSuspendSVCTask\(\)](#), [uvSuspendTask\(\)](#), and [uvTaskManager\(\)](#).

6.46.2.19 throttle_factor

```
uint8_t uv_task_info::throttle_factor
```

How much to throttle the task

Definition at line 278 of file [uvfr_state_engine.h](#).

6.46.2.20 tmi

```
task_management_info* uv_task_info::tmi
```

how we will be communicating in the future

Definition at line 254 of file [uvfr_state_engine.h](#).

Referenced by [uvTaskManager\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_state_engine.h](#)

6.47 uv_task_msg_t Struct Reference

Struct containing a message between two tasks.

```
#include <uvfr_utils.h>
```

Collaboration diagram for uv_task_msg_t:

Data Fields

- uint32_t [message_type](#)
- [uv_task_info](#) * [sender](#)
- [uv_task_info](#) * [intended_recipient](#)
- TickType_t [time_sent](#)
- size_t [message_size](#)
- void * [msg_contents](#)

6.47.1 Detailed Description

Struct containing a message between two tasks.

This is a generic type that is best used in situations where the message could mean a variety of different things. For niche applications or where efficiency is paramount, we recommend creating a bespoke protocol.

Definition at line 313 of file [uvfr_utils.h](#).

6.47.2 Field Documentation

6.47.2.1 intended_recipient

```
uv_task_info* uv_task_msg_t::intended_recipient
```

Definition at line 316 of file [uvfr_utils.h](#).

6.47.2.2 message_size

```
size_t uv_task_msg_t::message_size
```

Definition at line 318 of file [uvfr_utils.h](#).

6.47.2.3 message_type

```
uint32_t uv_task_msg_t::message_type
```

Definition at line 314 of file [uvfr_utils.h](#).

6.47.2.4 msg_contents

```
void* uv_task_msg_t::msg_contents
```

Definition at line 319 of file [uvfr_utils.h](#).

6.47.2.5 sender

```
uv_task_info* uv_task_msg_t::sender
```

Definition at line 315 of file [uvfr_utils.h](#).

6.47.2.6 time_sent

```
TickType_t uv_task_msg_t::time_sent
```

Definition at line 317 of file [uvfr_utils.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_utils.h](#)

6.48 uv_vehicle_settings Struct Reference

```
#include <uvfr_settings.h>
```

Collaboration diagram for uv_vehicle_settings:

Data Fields

- struct [veh_gen_info](#) * [veh_info](#)
- struct [uv_os_settings](#) * [os_settings](#)
- struct [motor_controller_settings](#) * [mc_settings](#)
- [driving_loop_args](#) * [driving_loop_settings](#)
- struct [uv_imd_settings](#) * [imd_settings](#)
- [bms_settings_t](#) * [bms_settings](#)
- [daq_loop_args](#) * [daq_settings](#)
- [daq_msg](#) * [daq_param_list](#)
- struct [conifer_settings](#) * [conifer_settings](#)
- uint16_t [flags](#)

6.48.1 Detailed Description

Definition at line 152 of file [uvfr_settings.h](#).

6.48.2 Field Documentation

6.48.2.1 bms_settings

```
bms_settings_t* uv_vehicle_settings::bms_settings
```

Definition at line 161 of file [uvfr_settings.h](#).

Referenced by [setupDefaultSettings\(\)](#), [uvInit\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.2 conifer_settings

```
struct conifer_settings* uv_vehicle_settings::conifer_settings
```

Definition at line 168 of file [uvfr_settings.h](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.3 daq_param_list

```
daq_msg* uv_vehicle_settings::daq_param_list
```

Definition at line 165 of file [uvfr_settings.h](#).

Referenced by [configureDaqSubTasks\(\)](#), [initDaqTask\(\)](#), [setupDefaultSettings\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.4 daq_settings

```
daq_loop_args* uv_vehicle_settings::daq_settings
```

Definition at line 163 of file [uvfr_settings.h](#).

Referenced by [initDaqTask\(\)](#), [setupDefaultSettings\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.5 driving_loop_settings

```
driving_loop_args* uv_vehicle_settings::driving_loop_settings
```

Definition at line 158 of file [uvfr_settings.h](#).

Referenced by [initDrivingLoop\(\)](#), [setupDefaultSettings\(\)](#), [StartDrivingLoop\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.6 flags

```
uint16_t uv_vehicle_settings::flags
```

Bitfield containing info on whether each settings instance is factory default. 0 default, 1 altered

Definition at line 171 of file [uvfr_settings.h](#).

Referenced by [setupDefaultSettings\(\)](#).

6.48.2.7 imd_settings

```
struct uv_imd_settings* uv_vehicle_settings::imd_settings
```

Definition at line 160 of file [uvfr_settings.h](#).

Referenced by [setupDefaultSettings\(\)](#), [uvInit\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.8 mc_settings

```
struct motor_controller_settings* uv_vehicle_settings::mc_settings
```

Definition at line 156 of file [uvfr_settings.h](#).

Referenced by [setupDefaultSettings\(\)](#), [uvInit\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

6.48.2.9 os_settings

```
struct uv_os_settings* uv_vehicle_settings::os_settings
```

Definition at line 155 of file [uvfr_settings.h](#).

Referenced by [setupDefaultSettings\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvStartStateMachine\(\)](#).

6.48.2.10 veh_info

```
struct veh_gen_info* uv_vehicle_settings::veh_info
```

Definition at line 153 of file [uvfr_settings.h](#).

Referenced by [uvLoadSettingsFromFlash\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_settings.h](#)

6.49 veh_gen_info Struct Reference

```
#include <uvfr_settings.h>
```

Data Fields

- uint32_t [wheel_size](#)
- uint32_t [drive_ratio](#)
- uint16_t [test1](#)
- uint16_t [test2](#)
- uint16_t [test3](#)
- uint8_t [test4](#)
- uint8_t [test5](#)
- uint32_t [test6](#)

6.49.1 Detailed Description

Definition at line 111 of file [uvfr_settings.h](#).

6.49.2 Field Documentation

6.49.2.1 drive_ratio

```
uint32_t veh_gen_info::drive_ratio
```

Definition at line 113 of file [uvfr_settings.h](#).

6.49.2.2 test1

```
uint16_t veh_gen_info::test1
```

Definition at line 114 of file [uvfr_settings.h](#).

6.49.2.3 test2

```
uint16_t veh_gen_info::test2
```

Definition at line 115 of file [uvfr_settings.h](#).

6.49.2.4 test3

```
uint16_t veh_gen_info::test3
```

Definition at line 116 of file [uvfr_settings.h](#).

6.49.2.5 test4

```
uint8_t veh_gen_info::test4
```

Definition at line 117 of file [uvfr_settings.h](#).

6.49.2.6 test5

```
uint8_t veh_gen_info::test5
```

Definition at line 118 of file [uvfr_settings.h](#).

6.49.2.7 test6

```
uint32_t veh_gen_info::test6
```

Definition at line 120 of file [uvfr_settings.h](#).

6.49.2.8 wheel_size

```
uint32_t veh_gen_info::wheel_size
```

Definition at line 112 of file [uvfr_settings.h](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_settings.h](#)

6.50 vehicle_log_entry Struct Reference

```
#include <uvfr_vehicle_logger.h>
```

Data Fields

- `fault_event_type_e type`
- `const char * task_name`
- `const char * msg`
- `uint32_t timestamp`
- `const char * file`
- `int line`
- `const char * func`
- `uint8_t is_panic`
- `uv_task_id task_id`
- `uv_vehicle_state vehicle_state`
- `uv_task_status task_state`

6.50.1 Detailed Description

Definition at line 42 of file [uvfr_vehicle_logger.h](#).

6.50.2 Field Documentation

6.50.2.1 file

```
const char* vehicle_log_entry::file
```

Definition at line 47 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.2 func

```
const char* vehicle_log_entry::func
```

Definition at line 49 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.3 is_panic

```
uint8_t vehicle_log_entry::is_panic
```

Definition at line 50 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), [flushLogsToCAN\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.4 line

```
int vehicle_log_entry::line
```

Definition at line 48 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.5 msg

```
const char* vehicle_log_entry::msg
```

Definition at line 45 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.6 task_id

```
uv_task_id vehicle_log_entry::task_id
```

Definition at line 51 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), [flushLogsToCAN\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.7 task_name

```
const char* vehicle_log_entry::task_name
```

Definition at line 44 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.8 task_state

```
uv_task_status vehicle_log_entry::task_state
```

Definition at line 53 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.9 timestamp

```
uint32_t vehicle_log_entry::timestamp
```

Definition at line 46 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), [flushLogsToCAN\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.10 type

```
fault_event_type_e vehicle_log_entry::type
```

Definition at line 43 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), [flushLogsToCAN\(\)](#), and [logVehicleFault\(\)](#).

6.50.2.11 vehicle_state

```
uv_vehicle_state vehicle_log_entry::vehicle_state
```

Definition at line 52 of file [uvfr_vehicle_logger.h](#).

Referenced by [dumpLogsToUART\(\)](#), [flushLogsToCAN\(\)](#), and [logVehicleFault\(\)](#).

The documentation for this struct was generated from the following file:

- Core/Inc/[uvfr_vehicle_logger.h](#)

Chapter 7

File Documentation

7.1 Core/Inc/adc.h File Reference

This file contains all the function prototypes for the [adc.c](#) file.

```
#include "main.h"
```

Include dependency graph for adc.h: This graph shows which files directly or indirectly include this file:

Macros

- `#define ADC1_BUF_LEN 40`
- `#define ADC1_CHNL_CNT 4`
- `#define ADC1_SAMPLES 10`
- `#define ADC2_BUF_LEN 2`
- `#define ADC2_CHNL_CNT 2`
- `#define ADC2_SAMPLES 1`
- `#define ADC1_MIN_VOLT 500`
- `#define ADC1_MAX_VOLT 2850`
- `#define ADC2_MIN_VOLT 69`
- `#define ADC2_MAX_VOLT 69`

Functions

- `void MX_ADC1_Init (void)`
- `void MX_ADC2_Init (void)`
- `void MX_ADC3_Init (void)`
- `void initADCTask (void)`
- `void processADCBuffer (uint8_t adc)`

Variables

- `ADC_HandleTypeDef hadc1`
- `ADC_HandleTypeDef hadc2`
- `ADC_HandleTypeDef hadc3`

7.1.1 Detailed Description

This file contains all the function prototypes for the [adc.c](#) file.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [adc.h](#).

7.1.2 Macro Definition Documentation

7.1.2.1 ADC1_BUF_LEN

```
#define ADC1_BUF_LEN 40
```

Definition at line [45](#) of file [adc.h](#).

7.1.2.2 ADC1_CHNL_CNT

```
#define ADC1_CHNL_CNT 4
```

Definition at line [46](#) of file [adc.h](#).

7.1.2.3 ADC1_MAX_VOLT

```
#define ADC1_MAX_VOLT 2850
```

Definition at line [57](#) of file [adc.h](#).

7.1.2.4 ADC1_MIN_VOLT

```
#define ADC1_MIN_VOLT 500
```

Definition at line [56](#) of file [adc.h](#).

7.1.2.5 ADC1_SAMPLES

```
#define ADC1_SAMPLES 10
```

Definition at line [47](#) of file [adc.h](#).

7.1.2.6 ADC2_BUFLLEN

```
#define ADC2_BUFLLEN 2
```

Definition at line 50 of file [adc.h](#).

7.1.2.7 ADC2_CHNL_CNT

```
#define ADC2_CHNL_CNT 2
```

Definition at line 51 of file [adc.h](#).

7.1.2.8 ADC2_MAX_VOLT

```
#define ADC2_MAX_VOLT 69
```

Definition at line 60 of file [adc.h](#).

7.1.2.9 ADC2_MIN_VOLT

```
#define ADC2_MIN_VOLT 69
```

Definition at line 59 of file [adc.h](#).

7.1.2.10 ADC2_SAMPLES

```
#define ADC2_SAMPLES 1
```

Definition at line 52 of file [adc.h](#).

7.1.3 Function Documentation

7.1.3.1 initADCTask()

```
void initADCTask (
    void )
```

Definition at line 687 of file [adc.c](#).

References [Error_Handler\(\)](#), and [StartADCTask\(\)](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.1.3.2 MX_ADC1_Init()

```
void MX_ADC1_Init (
    void )
```

Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time

Configures for the selected ADC injected channel its corresponding rank in the sequencer and its sample time

Definition at line [57](#) of file [adc.c](#).

References [Error_Handler\(\)](#), and [hadc1](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.1.3.3 MX_ADC2_Init()

```
void MX_ADC2_Init (
    void )
```

Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Definition at line [181](#) of file [adc.c](#).

References [Error_Handler\(\)](#), and [hadc2](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.1.3.4 MX_ADC3_Init()

```
void MX_ADC3_Init (
    void )
```

Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Definition at line [282](#) of file [adc.c](#).

References [Error_Handler\(\)](#), and [hadc3](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.1.3.5 processADCBuffer()

```
void processADCBuffer (
    uint8_t adc)
```

Definition at line [654](#) of file [adc.c](#).

References [adc1_APPS1](#), [adc1_APPS2](#), [adc1_BPS1](#), [adc1_BPS2](#), [adc1_pack_curr](#), [adc2_CoolantFlow](#), [adc2_CoolantTemp1](#), [adc2_CoolantTemp2](#), [adc2_steering_pos](#), [adc3_damper_FL](#), [adc3_damper_FR](#), [adc3_damper_RL](#), [adc3_damper_RR](#), [adc_buf1](#), [adc_buf2](#), and [adc_buf3](#).

Referenced by [HAL_ADC_ConvCpltCallback\(\)](#).

7.1.4 Variable Documentation

7.1.4.1 hadc1

```
ADC_HandleTypeDef hadc1 [extern]
```

Definition at line [49](#) of file [adc.c](#).

Referenced by [HAL_ADC_ConvCpltCallback\(\)](#), and [HAL_ADC_LevelOutOfWindowCallback\(\)](#).

7.1.4.2 hadc2

```
ADC_HandleTypeDef hadc2 [extern]
```

Definition at line [50](#) of file [adc.c](#).

Referenced by [HAL_ADC_ConvCpltCallback\(\)](#).

7.1.4.3 hadc3

```
ADC_HandleTypeDef hadc3 [extern]
```

Definition at line 51 of file [adc.c](#).

Referenced by [HAL_ADC_ConvCpltCallback\(\)](#).

7.2 adc.h

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -----*/
00021 #ifndef __ADC_H__
00022 #define __ADC_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes -----*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 extern ADC_HandleTypeDef hadc1;
00036
00037 extern ADC_HandleTypeDef hadc2;
00038
00039 extern ADC_HandleTypeDef hadc3;
00040
00041 /* USER CODE BEGIN Private defines */
00042
00043 // 12 bit ADC resolution
00044 // Buffer 1 has 4 channels, each is sampled 10 times per update
00045 #define ADC1_BUF_LEN 40
00046 #define ADC1_CHNL_CNT 4
00047 #define ADC1_SAMPLES 10
00048
00049 // Buffer 2 has 2 channels, each is sampled 1 time per update
00050 #define ADC2_BUF_LEN 2
00051 #define ADC2_CHNL_CNT 2
00052 #define ADC2_SAMPLES 1
00053
00054 // Calculated voltage ranges for ADCs (may remove since specified in the ioc file
00055 // ADC1 is for APPS and BPS sensors
00056 #define ADC1_MIN_VOLT 500
00057 #define ADC1_MAX_VOLT 2850
00058 // ADC2 is for coolant temp and flow sensors
00059 #define ADC2_MIN_VOLT 69
00060 #define ADC2_MAX_VOLT 69
00061
00062 /* USER CODE END Private defines */
00063
00064 void MX_ADC1_Init(void);
00065 void MX_ADC2_Init(void);
00066 void MX_ADC3_Init(void);
00067
00068 /* USER CODE BEGIN Prototypes */
00069
00070 void initADCTask(void);
00071 void processADCBuffer(uint8_t adc);
00072
00073
00074 /* USER CODE END Prototypes */
00075
00076 #ifdef __cplusplus
00077 }
00078 #endif
00079
00080 #endif /* __ADC_H__ */
00081
```

7.3 Core/Inc/bms.h File Reference

```
#include "main.h"
#include "uvfr_utils.h"
```

Include dependency graph for bms.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [bms_settings_t](#)

Macros

- #define [DEFAULT_BMS_CAN_TIMEOUT](#) (([uv_timespan_ms](#))200)

Typedefs

- typedef struct bms_settings_t [bms_settings_t](#)

Functions

- void [BMS_Init](#) (void *args)

7.3.1 Macro Definition Documentation

7.3.1.1 DEFAULT_BMS_CAN_TIMEOUT

```
#define DEFAULT_BMS_CAN_TIMEOUT ((uv\_timespan\_ms)200)
```

Definition at line 11 of file [bms.h](#).

7.3.2 Typedef Documentation

7.3.2.1 bms_settings_t

```
typedef struct bms_settings_t bms_settings_t
```

7.3.3 Function Documentation

7.3.3.1 BMS_Init()

```
void BMS_Init (
    void * args)
```

Definition at line 100 of file [bms.c](#).

References [BMS](#), [BMS_msg1\(\)](#), [BMS_msg2\(\)](#), [bms_settings](#), [CAN_BUS_1](#), [uv_init_task_args::init_info_queue](#), [insertCANMessageHandler\(\)](#), [is_bms_connected](#), [uv_init_task_args::meta_task_handle](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.4 bms.h

[Go to the documentation of this file.](#)

```
00001 // Code to make human readable CAN messages for the device
00002
00003
00004
00005 #ifndef _BMS_H_
00006 #define _BMS_H_
00007
00008 #include "main.h"
00009 #include "uvfr_utils.h"
00010
00011 #define DEFAULT_BMS_CAN_TIMEOUT ((uv_timespan_ms)200)
00012
00013 typedef struct bms_settings_t{ //TODO Needs populating
00014     uint32_t BMS_CAN_timeout;
00015     uint16_t max_cell_temp;
00016     uint16_t min_cell_temp;
00017     uint16_t min_soc;
00018     uint16_t min_cell_voltage;
00019     uint16_t max_cell_voltage;
00020     uint16_t max_pack_voltage;
00021     uint16_t min_pack_voltage;
00022     uint16_t max_variance_between_cells;
00023
00024 }bms_settings_t;
00025
00026 void BMS_Init(void* args);
00027
00028 #endif
00029
00030
00031
```

7.5 Core/Inc/can.h File Reference

This file contains all the function prototypes for the [can.c](#) file.

```
#include "main.h"
#include "constants.h"
#include "uvfr_utils.h"
```

Include dependency graph for can.h: This graph shows which files directly or indirectly include this file:

Macros

- `#define CAN_TX_DAEMON_NAME "CanTxDaemon"`
- `#define CAN_RX_DAEMON_NAME "CanRxDaemon"`

Typedefs

- `typedef struct uv_CAN_msg uv_CAN_msg`
Representative of a CAN message.
- `typedef enum uv_status_t uv_status`
This is meant to be a return type from functions that indicates what is actually going on.

Functions

- void [MX_CAN1_Init](#) (void)
- void [MX_CAN2_Init](#) (void)
- void [HAL_CAN_RxFifo0MsgPendingCallback](#) (CAN_HandleTypeDef *[hcan2](#))
- void [HAL_CAN_RxFifo1MsgPendingCallback](#) (CAN_HandleTypeDef *[hcan2](#))
- [uv_status uvSendCanMSG](#) ([uv_CAN_msg](#) *[msg](#))
Function to send CAN message.
- void [CANbusTxSvcDaemon](#) (void *[args](#))
Background task that handles any CAN messages that are being sent.
- void [CANbusRxSvcDaemon](#) (void *[args](#))
Background task that executes the CAN message callback functions.
- void [insertCANMessageHandler](#) (uint32_t [id](#), void *[handlerfunc](#), int [can_num](#))
Function to insert an id and function into the lookup table of callback functions.
- void [nuke_hash_table](#) (int [can_num](#))

Variables

- CAN_HandleTypeDef [hcan1](#)
- CAN_HandleTypeDef [hcan2](#)

7.5.1 Detailed Description

This file contains all the function prototypes for the [can.c](#) file.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [can.h](#).

7.5.2 Macro Definition Documentation

7.5.2.1 CAN_RX_DAEMON_NAME

```
#define CAN_RX_DAEMON_NAME "CanRxDaemon"
```

Definition at line 43 of file [can.h](#).

Referenced by [uvInit\(\)](#).

7.5.2.2 CAN_TX_DAEMON_NAME

```
#define CAN_TX_DAEMON_NAME "CanTxDaemon"
```

Definition at line 42 of file [can.h](#).

Referenced by [uvInit\(\)](#).

7.5.3 Typedef Documentation

7.5.3.1 uv_CAN_msg

```
typedef struct uv_CAN_msg uv_CAN_msg
```

Representative of a CAN message.

Definition at line 45 of file [can.h](#).

7.5.3.2 uv_status

```
typedef enum uv_status_t uv_status
```

This is meant to be a return type from functions that indicates what is actually going on.

Use this as a return value for functions you want to know the success of. In general, any function you write must return something, as well as account for any possible errors that may have occurred.

Definition at line 46 of file [can.h](#).

7.5.4 Function Documentation

7.5.4.1 CANbusRxSvcDaemon()

```
void CANbusRxSvcDaemon (
    void * args)
```

Background task that executes the CAN message callback functions.

dequeue can message to call function

Definition at line 1045 of file [can.c](#).

References [callback_table_1_mutex](#), [callback_table_2_mutex](#), [uv_task_info::cmd_data](#), [killSelf\(\)](#), [suspendSelf\(\)](#), [uv_task_info::task_handle](#), [UV_KILL_CMD](#), [UV_OK](#), and [UV_SUSPEND_CMD](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.5.4.2 CANbusTxSvcDaemon()

```
void CANbusTxSvcDaemon (
    void * args)
```

Background task that handles any CAN messages that are being sent.

This task sits idle, until the time is right (it receives a notification from the uvSendCanMSG function) Once this condition has been met, it will actually call the HAL_CAN_AddTxMessage function. This is a very high priority task, meaning that it will pause whatever other code is going in order to run

dequeue can message to put into can bus

Definition at line 920 of file [can.c](#).

References [CAN_BUS_1](#), [uv_task_info::cmd_data](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [hcan1](#), [hcan2](#), [is_can_ok](#), [killSelf\(\)](#), [uv_CAN_msg::msg_id](#), [suspendSelf\(\)](#), [TxHeader](#), [TxHeader2](#), [TxMailbox](#), [UV_CAN_EXTENDED_ID](#), [UV_KILL_CMD](#), and [UV_SUSPEND_CMD](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.5.4.3 HAL_CAN_RxFifo0MsgPendingCallback()

```
void HAL_CAN_RxFifo0MsgPendingCallback (
    CAN_HandleTypeDef * hcan2)
```

Definition at line 456 of file [can.c](#).

References [CAN_BUS_1](#), [CAN_BUS_2](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [Error_Handler\(\)](#), [uv_CAN_msg::flags](#), [hcan1](#), [is_can_ok](#), [uv_CAN_msg::msg_id](#), [RxHeader](#), [RxHeader2](#), and [UV_CAN_EXTENDED_ID](#).

Here is the call graph for this function:

7.5.4.4 HAL_CAN_RxFifo1MsgPendingCallback()

```
void HAL_CAN_RxFifo1MsgPendingCallback (
    CAN_HandleTypeDef * hcan2)
```

Definition at line 512 of file [can.c](#).

References [CAN_BUS_1](#), [CAN_BUS_2](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [Error_Handler\(\)](#), [uv_CAN_msg::flags](#), [hcan1](#), [is_can_ok](#), [uv_CAN_msg::msg_id](#), [RxHeader](#), [RxHeader2](#), and [UV_CAN_EXTENDED_ID](#).

Here is the call graph for this function:

7.5.4.5 MX_CAN1_Init()

```
void MX_CAN1_Init (
    void )
```

Definition at line 156 of file [can.c](#).

References [Error_Handler\(\)](#), [hcan1](#), and [TxHeader](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.5.4.6 MX_CAN2_Init()

```
void MX_CAN2_Init (
    void )
```

Definition at line 232 of file [can.c](#).

References [Error_Handler\(\)](#), [hcan2](#), and [TxHeader](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.5.4.7 nuke_hash_table()

```
void nuke_hash_table (
    int can_num)
```

Function to free all mallocoed memory Index through the hash table and free all the mallocoed memory at each index

Definition at line 773 of file [can.c](#).

References [CAN_BUS_1](#), [CAN_BUS_2](#), [CAN_callback_table_1](#), [CAN_callback_table_2](#), [CAN_Callback::next](#), and [table_size](#).

7.5.5 Variable Documentation

7.5.5.1 hcan1

```
CAN_HandleTypeDef hcan1 [extern]
```

Definition at line 152 of file [can.c](#).

7.5.5.2 hcan2

```
CAN_HandleTypeDef hcan2 [extern]
```

Definition at line 153 of file [can.c](#).

Referenced by [IMD_Request_Status\(\)](#), [main\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.6 can.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00003 // *****
00004 // * @file      can.h
00005 // * @brief     This file contains all the function prototypes for
00006 // *             the can.c file
00007 // *****
00008 // * @attention
00009 // *
00010 // * Copyright (c) 2023 STMicroelectronics.
00011 // * All rights reserved.
00012 // *
00013 // * This software is licensed under terms that can be found in the LICENSE file
00014 // * in the root directory of this software component.
00015 // * If no LICENSE file comes with this software, it is provided AS-IS.
00016 // *
00017 // *****
00018 // */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -----*/
00021 #ifndef __CAN_H__
00022 #define __CAN_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes -----*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032 #include "constants.h"
00033 #include "uvfr_utils.h"
00034 /* USER CODE END Includes */
00035
00036 extern CAN_HandleTypeDef hcan1;
00037
00038 extern CAN_HandleTypeDef hcan2;
00039
00040 /* USER CODE BEGIN Private defines */
00041
00042 #define CAN_TX_DAEMON_NAME "CanTxDaemon"
00043 #define CAN_RX_DAEMON_NAME "CanRxDaemon"
00044
00045 typedef struct uv_CAN_msg uv_CAN_msg;
00046 typedef enum uv_status_t uv_status;
00047
00048 /* USER CODE END Private defines */
00049
00050 void MX_CAN1_Init(void);
00051 void MX_CAN2_Init(void);
00052
00053 /* USER CODE BEGIN Prototypes */
00054
00055 void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan2);
00056 void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan2);
00057
00058 uv_status uvSendCanMSG(uv_CAN_msg * msg);
00059
00060 void CANbusTxSvcDaemon(void* args);
00061 void CANbusRxSvcDaemon(void* args);
00062
00063
00064 //int callFunctionFromCANid(uint32_t CAN_id, uint8_t* data, uint8_t length);
00065 void insertCANMessageHandler(uint32_t id, void* handlerfunc, int can_num);
00066 void nuke_hash_table(int can_num);
00067 /* USER CODE END Prototypes */
00068
00069 #ifdef __cplusplus
00070 }
00071 #endif
00072
00073 #endif /* __CAN_H__ */
00074

```

7.7 Core/Inc/constants.h File Reference

This graph shows which files directly or indirectly include this file:

Enumerations

- enum CAN_IDs {
 IMD_CAN_ID_Tx = 0xA100101 , IMD_CAN_ID_Rx = 0xA100100 , PDU_CAN_ID_Tx = 0x710 ,
 MC_CAN_ID_Tx = 0x201 ,
 MC_CAN_ID_Rx = 0x181 }

Variables

- CAN_TxHeaderTypeDef TxHeader
- CAN_RxHeaderTypeDef RxHeader
- CAN_TxHeaderTypeDef TxHeader2
- CAN_RxHeaderTypeDef RxHeader2
- uint8_t TxData [8]
- uint32_t TxMailbox
- uint8_t RxData [8]

7.7.1 Enumeration Type Documentation

7.7.1.1 CAN_IDs

enum CAN_IDs

Enumerator

IMD_CAN_ID_Tx	
IMD_CAN_ID_Rx	
PDU_CAN_ID_Tx	
MC_CAN_ID_Tx	
MC_CAN_ID_Rx	

Definition at line 18 of file [constants.h](#).

7.7.2 Variable Documentation

7.7.2.1 RxData

uint8_t RxData[8] [extern]

Definition at line 12 of file [constants.c](#).

7.7.2.2 RxHeader

CAN_RxHeaderTypeDef RxHeader [extern]

Definition at line 6 of file [constants.c](#).

Referenced by [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), and [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#).

7.7.2.3 RxHeader2

```
CAN_RxHeaderTypeDef RxHeader2 [extern]
```

Definition at line 7 of file [constants.c](#).

Referenced by [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), and [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#).

7.7.2.4 TxData

```
uint8_t TxData[8] [extern]
```

Definition at line 10 of file [constants.c](#).

Referenced by [IMD_Request_Status\(\)](#), [main\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.7.2.5 TxHeader

```
CAN_TxHeaderTypeDef TxHeader [extern]
```

Definition at line 4 of file [constants.c](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [IMD_Request_Status\(\)](#), [main\(\)](#), [MX_CAN1_Init\(\)](#), [MX_CAN2_Init\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.7.2.6 TxHeader2

```
CAN_TxHeaderTypeDef TxHeader2 [extern]
```

Definition at line 5 of file [constants.c](#).

7.7.2.7 TxMailbox

```
uint32_t TxMailbox [extern]
```

Definition at line 11 of file [constants.c](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [IMD_Request_Status\(\)](#), [main\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.8 constants.h

[Go to the documentation of this file.](#)

```

00001 #ifndef SOME_UNIQUE_NAME_HERE
00002 #define SOME_UNIQUE_NAME_HERE
00003
00004 // your declarations (and certain types of definitions) here
00005
00006 extern CAN_TxHeaderTypeDef TxHeader;
00007 extern CAN_RxHeaderTypeDef RxHeader;
00008
00009 extern CAN_TxHeaderTypeDef TxHeader2;
00010 extern CAN_RxHeaderTypeDef RxHeader2;
00011
00012 extern uint8_t TxData[8];
00013 extern uint32_t TxMailbox;
00014 extern uint8_t RxData[8];
00015
00016
00017 // For constant CAN IDs, might be easier to put them in here
00018 enum CAN_IDS{
00019     IMD_CAN_ID_Tx = 0xA100101, // This is the ID for MCU sending msg to IMD
00020     IMD_CAN_ID_Rx = 0xA100100, // MCU will receive messages with this id from IMD
00021     PDU_CAN_ID_Tx = 0x710, // ID to send messages to the PDU
00022     MC_CAN_ID_Tx = 0x201, // We send messages to the motor controller with this ID
00023     MC_CAN_ID_Rx = 0x181,
00024
00025     // TODO figure out BMS IDs
00026     //BMS_COBIE_ID = ((uint32_t)0x20U),
00027     //BMS_HEARTBEAT_ID = ((uint32_t)0x80U),
00028     //BMS_MSG_ID_1 = ((uint32_t)(0x180U + BMS_COBIE_ID)),
00029     //BMS_MSG_ID_2 = ((uint32_t)(0x280U + BMS_COBIE_ID)),
00030     //BMS_MSG_ID_3 = ((uint32_t)(0x380U + BMS_COBIE_ID)),
00031 };
00032
00033
00034
00035 #endif
00036
00037
00038

```

7.9 Core/Inc/daq.h File Reference

```
#include "uvfr_utils.h"
#include "rb_tree.h"
```

Include dependency graph for daq.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [daq_msg](#)
- struct [daq_loop_args](#)

This struct holds info of what needs to be logged.

Macros

- [#define _NUM_LOGGABLE_PARAMS](#)

Typedefs

- [typedef struct daq_msg daq_msg](#)
- [typedef struct daq_loop_args daq_loop_args](#)

This struct holds info of what needs to be logged.

Enumerations

- enum `loggable_params` {
 `VCU_VEHICLE_STATE`, `VCU_ERROR_BITFIELD1`, `VCU_ERROR_BITFIELD2`, `VCU_ERROR_BITFIELD3`,
`VCU_ERROR_BITFIELD4`, `VCU_CURRENT_UPTIME`, `VCU_TOTAL_UPTIME`, `OS_AVAILABLE_HEAP`,
`OS_LARGEST_FREE_BLOCK`, `OS_SMALLEST_FREE_BLOCK`, `OS_NUM_FREE_BLOCKS`, `OS_MIN_EVER_FREE_BYT`,
`OS_NUM_SUCCESSFUL_ALLOCS`, `OS_NUM_SUCCESSFUL_FREES`, `VEH_DISTANCE_RUN`,
`VEH_DISTANCE_TOTAL`,
`VEH_LAPNUM`, `VEH_SPEED`, `MOTOR_RPM`, `MOTOR_TEMP`,
`MOTOR_CURRENT`, `MC_VOLTAGE`, `MC_CURRENT`, `MC_TEMP`,
`MC_ERRORS`, `BMS_CURRENT`, `BMS_VOLTAGE`, `BMS_ERRORS`,
`MAX_CELL_TEMP`, `MIN_CELL_TEMP`, `AVG_CELL_TEMP`, `ACCUM_SOC`,
`ACCUM_SOH`, `ACCUM_POWER`, `ACCUM_POWER_LIMIT`, `APPS1_ADC_VAL`,
`APPS2_ADC_VAL`, `BPS1_ADC_VAL`, `BPS2_ADC_VAL`, `COOLANT_TEMP_ADC`,
`MOTOR_TEMP_ADC`, `ACCELERATOR_PEDAL_RATIO`, `BRAKE_PRESSURE_PA`, `POWER_DERATE_FACTOR`,
`CURRENT_DRIVING_MODE`, `IMD_VOLTAGE`, `IMD_STATUS`, `IMD_ERRORS`,
`SUS_DAMPER_FL`, `SUS_DUMPER_FR`, `SUS_DAMPER_RL`, `SUS_DAMPER_RR`,
`WSS_FR`, `WSS_FL`, `WSS_RL`, `WSS_RR`,
`WSS_F_AVG`, `WSS_R_AVG`, `WSS_SLIP`, `MAX_LOGGABLE_PARAMS` }

Functions

- `uv_status associateDaqParamWithVar (uint16_t paramID, void *var)`
- `uv_status initDaqTask (void *args)`
initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks
- `void daqMasterTask (void *args)`
Controls the Daq.

7.9.1 Macro Definition Documentation

7.9.1.1 `_NUM_LOGGABLE_PARAMS`

```
#define _NUM_LOGGABLE_PARAMS
```

Definition at line 14 of file `daq.h`.

7.9.2 Typedef Documentation

7.9.2.1 `daq_loop_args`

```
typedef struct daq_loop_args daq_loop_args
```

This struct holds info of what needs to be logged.

7.9.2.2 `daq_msg`

```
typedef struct daq_msg daq_msg
```

7.9.3 Enumeration Type Documentation

7.9.3.1 `loggable_params`

```
enum loggable_params
```

Enumerator

VCU_VEHICLE_STATE	VCU Current Vehicle State
VCU_ERROR_BITFIELD1	
VCU_ERROR_BITFIELD2	
VCU_ERROR_BITFIELD3	
VCU_ERROR_BITFIELD4	
VCU_CURRENT_UPTIME	
VCU_TOTAL_UPTIME	
OS_AVAILABLE_HEAP	
OS_LARGEST_FREE_BLOCK	
OS_SMALLEST_FREE_BLOCK	
OS_NUM_FREE_BLOCKS	
OS_MIN_EVER_FREE_BYTES	
OS_NUM_SUCCESSFUL_ALLOCS	
OS_NUM_SUCCESSFUL_FREES	
VEH_DISTANCE_RUN	
VEH_DISTANCE_TOTAL	
VEH_LAPNUM	
VEH_SPEED	
MOTOR_RPM	RPM as reported by motor controller
MOTOR_TEMP	Motor Temp as reported by motor controller
MOTOR_CURRENT	Motor Phase currents as reported by motor controller
MC_VOLTAGE	Pack voltage as measured by motor_controller
MC_CURRENT	Pack current as measured by motor_controller
MC_TEMP	Motor controller temperature
MC_ERRORS	Motor controller errors bitfield
BMS_CURRENT	Pack current measured by BMS
BMS_VOLTAGE	Pack voltage as measured by BMS
BMS_ERRORS	Error codes in BMS
MAX_CELL_TEMP	Max Temperature of a cell from BMS
MIN_CELL_TEMP	Min Temperature of a cell
AVG_CELL_TEMP	Average Cell Temp
ACCUM_SOC	
ACCUM_SOH	
ACCUM_POWER	
ACCUM_POWER_LIMIT	
APPSC1_ADC_VAL	
APPSC2_ADC_VAL	
BPS1_ADC_VAL	
BPS2_ADC_VAL	
COOLANT_TEMP_ADC	
MOTOR_TEMP_ADC	
ACCELERATOR_PEDAL_RATIO	
BRAKE_PRESSURE_PA	
POWER_DERATE_FACTOR	
CURRENT_DRIVING_MODE	
IMD_VOLTAGE	Accumulator voltage as measured by IMD
IMD_STATUS	
IMD_ERRORS	

Enumerator

SUS_DAMPER_FL	
SUS_DUMPER_FR	
SUS_DAMPER_RL	
SUS_DAMPER_RR	
WSS_FR	
WSS_FL	
WSS_RL	
WSS_RR	
WSS_F_AVG	
WSS_R_AVG	
WSS_SLIP	
MAX_LOGGABLE_PARAMS	THIS MUST BE THE FINAL PARAM

Definition at line 18 of file [daq.h](#).

7.9.4 Function Documentation

7.9.4.1 associateDaqParamWithVar()

```
uv_status associateDaqParamWithVar (
    uint16_t paramID,
    void * var)
```

Definition at line 87 of file [daq.c](#).

References [MAX_LOGGABLE_PARAMS](#), [UV_ERROR](#), [UV_OK](#), and [uvIsPTRValid\(\)](#).

Referenced by [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), and [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.9.4.2 daqMasterTask()

```
void daqMasterTask (
    void * args)
```

Controls the Daq.

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
/*
//TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
//TickType_t last_time = xTaskGetTickCount(); */
```

Definition at line 305 of file [daq.c](#).

References [uv_task_info::cmd_data](#), [killSelf\(\)](#), [startDaqSubTasks\(\)](#), [uv_task_info::task_period](#), [UV_KILL_CMD](#), [UV_OK](#), [UV_SUSPEND_CMD](#), and [uvTaskDelay\(\)](#).

Referenced by [initDaqTask\(\)](#).

Here is the call graph for this function:

7.9.4.3 initDaqTask()

```
uv_status initDaqTask (
    void * args)
```

initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks

This is a fairly standard function. Here are the things that it does in order:

Step 1: Get Daq settings.

Step 2: Create and configure DAQ task.

Step 3: Read which parameters we want to read.

Step 4: Generate Subtask Metadata

Step 5: Assign params to subtasks

Definition at line 257 of file [daq.c](#).

References [uv_task_info::active_states](#), [associateDaqParamWithVar\(\)](#), [daq_loop_args::can_channel](#), [configureDaqSubTasks\(\)](#), [COOLANT_TEMP_ADC](#), [coolant_temp_adc](#), [curr_daq_settings](#), [current_vehicle_settings](#), [daq_loop_args::daq_child_priority](#), [uv_vehicle_settings::daq_param_list](#), [uv_vehicle_settings::daq_settings](#), [daqMasterTask\(\)](#), [uv_task_info::deletion_states](#), [uv_CAN_msg::flags](#), [MOTOR_TEMP_ADC](#), [motor_temp_adc](#), [PROGRAMMING](#), [uv_task_info::stack_size](#), [uv_task_info::suspension_states](#), [uv_task_info::task_args](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [uv_task_info::task_period](#), [uv_task_info::task_priority](#), [tmp_daq_msg](#), [UV_DRIVING](#), [UV_ERROR](#), [UV_ERROR_STATE](#), [UV_LAUNCH_CONTROL](#), [UV_OK](#), [UV_READY](#), and [uvCreateTask\(\)](#).

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.10 daq.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * daq.h
00003  *
00004  * Created on: Oct 15, 2024
00005  *      Author: byo10
00006 */
00007
00008 #ifndef INC_DAQ_H_
00009 #define INC_DAQ_H_
00010
00011 #include "uvfr_utils.h"
00012 #include "rb_tree.h"
00013
00014 #define _NUM_LOGGABLE_PARAMS
00015
00016
00017
00018 typedef enum{
00019
00020     VCU_VEHICLE_STATE,
00021     VCU_ERROR_BITFIELD1,
00022     VCU_ERROR_BITFIELD2,
00023     VCU_ERROR_BITFIELD3,
00024     VCU_ERROR_BITFIELD4,
00025     VCU_CURRENT_UPTIME,
00026     VCU_TOTAL_UPTIME,
00027     OS_AVAILABLE_HEAP,
00028     OS_LARGEST_FREE_BLOCK,
00029     OS_SMALLEST_FREE_BLOCK,
00030     OS_NUM_FREE_BLOCKS,
```

```
00031     OS_MIN_EVER_FREE_BYTES,
00032     OS_NUM_SUCCESSFUL_ALLOCS,
00033     OS_NUM_SUCCESSFUL_FREES,
00034     VEH_DISTANCE_RUN,
00035     VEH_DISTANCE_TOTAL,
00036     VEH_LAPNUM,
00037     VEH_SPEED,
00038     MOTOR_RPM,
00039     MOTOR_TEMP,
00040     MOTOR_CURRENT,
00041     MC_VOLTAGE,
00042     MC_CURRENT,
00043     MC_TEMP,
00044     MC_ERRORS,
00045     BMS_CURRENT,
00046     BMS_VOLTAGE,
00047     BMS_ERRORS,
00048     MAX_CELL_TEMP,
00049     MIN_CELL_TEMP,
00050     AVG_CELL_TEMP,
00051     ACCUM_SOC,
00052     ACCUM_SOH,
00053     ACCUM_POWER,
00054     ACCUM_POWER_LIMIT,
00055     APPS1_ADC_VAL,
00056     APPS2_ADC_VAL,
00057     BPS1_ADC_VAL,
00058     BPS2_ADC_VAL,
00059     COOLANT_TEMP_ADC,
00060     MOTOR_TEMP_ADC,
00061     ACCELERATOR_PEDAL_RATIO,
00062     BRAKE_PRESSURE_PA,
00063     POWER_DERATE_FACTOR,
00064     CURRENT_DRIVING_MODE,
00065     IMD_VOLTAGE,
00066     IMD_STATUS,
00067     IMD_ERRORS,
00068     SUS_DAMPER_FL,
00069     SUS_DUMPER_FR,
00070     SUS_DAMPER_RL,
00071     SUS_DAMPER_RR,
00072     WSS_FR,
00073     WSS_FL,
00074     WSS_RL,
00075     WSS_RR,
00076     WSS_F_AVG,
00077     WSS_R_AVG,
00078     WSS_SLIP,
00079     MAX_LOGGABLE_PARAMS
00080 }loggable_params;
00081
00082
00083 typedef struct daq_msg{ //8 bytes, convenient, no?
00084     uint32_t can_id;
00085     uint16_t param[4];
00086     uint8_t type[4];
00087     uint8_t period;
00088 }daq_msg;
00089
00090
00091
00092
00093 typedef struct daq_loop_args{
00094     uint16_t total_params_logged;
00095     uint8_t throttle_daq_to_preserve_performance;
00096     uint8_t minimum_daq_period;
00097     uint8_t can_channel;
00098     uint8_t daq_child_priority;
00099 }daq_loop_args;
00100
00101
00102
00103
00104
00105 typedef enum uv_status_t uv_status;
00106
00107
00108 uv_status associateDaqParamWithVar(uint16_t paramID, void* var);
00109 uv_status initDaqTask(void * args);
00110 void daqMasterTask(void* args);
00111
00112
00113
00114
00115 #endif /* INC_DAQ_H_ */
```

7.11 Core/Inc/dash.h File Reference

```
#include "main.h"
```

Include dependency graph for dash.h: This graph shows which files directly or indirectly include this file:

Enumerations

- enum [dash_can_ids](#) { [Dash_RPM](#) = 0x80 , [Dash_Battery_Temperature](#) = 0x82 , [Dash_Motor_Temperature](#) = 0x88 , [Dash_State_of_Charge](#) = 0x87 }

Functions

- void [Update_RPM](#) (int16_t value)
- void [Update_Batt_Temp](#) (uint8_t value)
- void [Update_State_Of_Charge](#) (uint8_t value)

7.11.1 Enumeration Type Documentation

7.11.1.1 dash_can_ids

```
enum dash\_can\_ids
```

Enumerator

Dash_RPM	
Dash_Battery_Temperature	
Dash_Motor_Temperature	
Dash_State_of_Charge	

Definition at line 14 of file [dash.h](#).

7.11.2 Function Documentation

7.11.2.1 Update_Batt_Temp()

```
void Update\_Batt\_Temp (
    uint8_t value)
```

Definition at line 29 of file [dash.c](#).

References [Dash_Battery_Temperature](#), [Error_Handler\(\)](#), [hcan2](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Here is the call graph for this function:

7.11.2.2 Update_RPM()

```
void Update_RPM (
    int16_t value)
```

Definition at line 9 of file [dash.c](#).

References [Dash_RPM](#), [Error_Handler\(\)](#), [hcan2](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.11.2.3 Update_State_Of_Charge()

```
void Update_State_Of_Charge (
    uint8_t value)
```

Definition at line 48 of file [dash.c](#).

References [Dash_State_of_Charge](#), [Error_Handler\(\)](#), [hcan2](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Here is the call graph for this function:

7.12 dash.h

[Go to the documentation of this file.](#)

```
00001 // Code to make human readable CAN messages for the device
00002
00003 // This file will define message data as human readable stuff
00004
00005 // The dash does not have a unique CAN ID
00006 // Each function on the dash has a unique CAN ID
00007
00008 #include "main.h"
00009
00010 #ifndef __DASH_H__
00011 #define __DASH_H__
00012
00013 // CAN IDs correspond to specific values on the dash
00014 enum dash_can_ids{
00015     Dash_RPM = 0x80,
00016     Dash_Battery_Temperature = 0x82,
00017     Dash_Motor_Temperature = 0x88,
00018     Dash_State_of_Charge = 0x87,
00019 };
00020
00021 // Battery Temperature 0x82
00022
00023 // Motor Temperature 0x87
00024
00025 // Inverter Temperature TODO
00026
00027 // RPM 0x80
00028
00029 // State of Charge 0x87
00030
00031 // Need to define these
00032
00033 void Update_RPM(int16_t value);
00034 void Update_Batt_Temp(uint8_t value);
00035 void Update_State_Of_Charge(uint8_t value);
00036
00037 #endif
00038
```

7.13 Core/Inc/dma.h File Reference

This file contains all the function prototypes for the [dma.c](#) file.

```
#include "main.h"
```

Include dependency graph for dma.h: This graph shows which files directly or indirectly include this file:

Functions

- void [MX_DMA_Init](#) (void)

7.13.1 Detailed Description

This file contains all the function prototypes for the [dma.c](#) file.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [dma.h](#).

7.13.2 Function Documentation

7.13.2.1 MX_DMA_Init()

```
void MX_DMA_Init (
    void )
```

Enable DMA controller clock

Definition at line 39 of file [dma.c](#).

Referenced by [main\(\)](#).

7.14 dma.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -----*/
00021 #ifndef __DMA_H__
00022 #define __DMA_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes -----*/
00029 #include "main.h"
00030
00031 /* DMA memory to memory transfer handles -----*/
00032
00033 /* USER CODE BEGIN Includes */
00034
00035 /* USER CODE END Includes */
00036
00037 /* USER CODE BEGIN Private defines */
00038
00039 /* USER CODE END Private defines */
00040
00041 void MX_DMA_Init(void);
00042
00043 /* USER CODE BEGIN Prototypes */
00044
00045 /* USER CODE END Prototypes */
00046
00047 #ifdef __cplusplus
00048 }
00049 #endif
00050
00051 #endif /* __DMA_H__ */
00052

```

7.15 Core/Inc/driving_loop.h File Reference

```
#include "motor_controller.h"
#include "uvfr_utils.h"
```

Include dependency graph for driving_loop.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [linear_torque_map_args](#)
 struct to hold parameters used in an exponential torque map
- struct [exp_torque_map_args](#)
 struct for s-curve parameters for torque
- union [drivingModeParams](#)
 this struct is designed to hold information about each drivingmode's map params
- struct [drivingMode](#)
 This is where the driving mode and the [drivingModeParams](#) are at.
- struct [driving_loop_args](#)

Typedefs

- `typedef uint16_t MC_Torque`
- `typedef uint16_t MC_RPM`
- `typedef uint16_t MC_POWER`
- `typedef struct linear_torque_map_args linear_torque_map_args`
- `typedef struct exp_torque_map_args exp_torque_map_args`
`struct to hold parameters used in an exponential torque map`
- `typedef struct s_curve_torque_map_args s_curve_torque_map_args`
`struct for s-curve parameters for torque`
- `typedef union drivingModeParams drivingModeParams`
`this struct is designed to hold information about each drivingmode's map params`
- `typedef struct drivingMode drivingMode`
`This is where the driving mode and the drivingModeParams are at.`
- `typedef struct driving_loop_args driving_loop_args`

Enumerations

- `enum map_mode {`
`linear_speed_map , s_curve_speed_map , exp_speed_map , linear_torque_map ,`
`s_curve_torque_map , exp_torque_map }`
 - `enum DL_internal_state { Plausible = 0x01 , Implausible = 0x02 , Erroneous = 0x04 }`
- DL_PERIOD is meant to represent how often the driving loop executes, in ms.*

Functions

- `enum uv_status_t initDrivingLoop (void *argument)`
- `void StartDrivingLoop (void *argument)`
`Sends the filtered torque value to the motor controller.`

7.15.1 Typedef Documentation

7.15.1.1 driving_loop_args

```
typedef struct driving_loop_args driving_loop_args
```

7.15.1.2 drivingMode

```
typedef struct drivingMode drivingMode
```

This is where the driving mode and the `drivingModeParams` are at.

7.15.1.3 drivingModeParams

```
typedef union drivingModeParams drivingModeParams
```

this struct is designed to hold information about each drivingmode's map params

7.15.1.4 exp_torque_map_args

```
typedef struct exp_torque_map_args exp_torque_map_args
```

struct to hold parameters used in an exponential torque map

7.15.1.5 linear_torque_map_args

```
typedef struct linear_torque_map_args linear_torque_map_args
```

7.15.1.6 MC_POWER

```
typedef uint16_t MC_POWER
```

Definition at line 16 of file [driving_loop.h](#).

7.15.1.7 MC_RPM

```
typedef uint16_t MC_RPM
```

Definition at line 15 of file [driving_loop.h](#).

7.15.1.8 MC_Torque

```
typedef uint16_t MC_Torque
```

Definition at line 14 of file [driving_loop.h](#).

7.15.1.9 s_curve_torque_map_args

```
typedef struct s_curve_torque_map_args s_curve_torque_map_args
```

struct for s-curve parameters for torque

7.15.2 Enumeration Type Documentation

7.15.2.1 DL_internal_state

```
enum DL_internal_state
```

Enumerator

Plausible	
Implausible	
Erroneous	

Definition at line 42 of file [driving_loop.h](#).

7.15.2.2 map_mode

```
enum map_mode
```

DL_PERIOD is meant to represent how often the driving loop executes, in ms.

This is a define since I would eventually like this to be configurable via a global variable, or possibly be dynamic in the future.

Just replace the number with the name of the variable, and you're all set.

enum meant to represent the different types of pedal map

This enum is meant to represent different functions that map the torque to speed.

Enumerator

linear_speed_map	
s_curve_speed_map	
exp_speed_map	
linear_torque_map	
s_curve_torque_map	
exp_torque_map	

Definition at line 33 of file [driving_loop.h](#).

7.15.3 Function Documentation

7.15.3.1 initDrivingLoop()

```
enum uv_status_t initDrivingLoop (
    void * argument)
```

Definition at line 108 of file [driving_loop.c](#).

References [uv_task_info::active_states](#), [adc1_APPS1](#), [adc1_APPS2](#), [adc1_BPS1](#), [adc1_BPS2](#), [APPSS1_ADC_VAL](#), [APPSS2_ADC_VAL](#), [associateDaqParamWithVar\(\)](#), [BPS1_ADC_VAL](#), [BPS2_ADC_VAL](#), [current_vehicle_settings](#), [uv_task_info::deletion_states](#), [driving_args](#), [uv_vehicle_settings::driving_loop_settings](#), [PROGRAMMING](#), [uv_task_info::stack_size](#), [StartDrivingLoop\(\)](#), [uv_task_info::suspension_states](#), [uv_task_info::task_args](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [uv_task_info::task_period](#), [uv_task_info::task_priority](#), [UV_DRIVING](#), [UV_ERROR](#), [UV_ERROR_STATE](#), [UV_INIT](#), [UV_LAUNCH_CONTROL](#), [UV_OK](#), [UV_READY](#), [UV_SUSPENDED](#), and [uvCreateTask\(\)](#).

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.15.3.2 StartDrivingLoop()

```
void StartDrivingLoop (
    void * argument)
```

Sends the filtered torque value to the motor controller.

Rachan

Parameters

<i>T_filtered</i>	Final torque value after filtering.
-------------------	-------------------------------------

This line extracts the specific driving loop parameters as specified in the vehicle settings

```
/*
driving_loop_args* dl_params = current_vehicle_settings->driving_loop_settings;

//Timeout values
//static TickType_t last_input_change_time = 0;
//static float last_throttle_percent = 0.0f;
//static float last_brake_percent = 0.0f;

TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = xTaskGetTickCount();
last_input_change_time = last_time;

/**
```

Definition at line 254 of file [driving_loop.c](#).

References `adc1_APPS1`, `adc1_APPS2`, `adc1_BPS1`, `adc1_BPS2`, `BRAKE_CHANGE_THRESHOLD`, `calculateBrakePercentage()`, `calculateThrottlePercentage()`, `uv_task_info::cmd_data`, `current_vehicle_settings`, `uv_vehicle_settings::driving_loop_settings`, `INPUT_TIMEOUT_MS`, `is_accelerating`, `killSelf()`, `last_driver_input_time`, `performSafetyChecks()`, `Plausible`, `sendTorqueToMotorController()`, `suspendSelf()`, `T_PREV`, `T_REQ`, `uv_task_info::task_period`, `THROTTLE_CHANGE_THRESHOLD`, `UV_DRIVING`, `UV_KILL_CMD`, `UV_SUSPEND_CMD`, and `vehicle_state`.

Referenced by `initDrivingLoop()`.

Here is the call graph for this function:

7.16 driving_loop.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 * driving_loop.h
00003 *
00004 * Created on: Oct 14, 2024
00005 * Author: byo10
00006 */
00007
00008 #ifndef INC_DRIVING_LOOP_H_
00009 #define INC_DRIVING_LOOP_H_
00010
00011 #include "motor_controller.h"
00012 #include "uvfr_utils.h"
00013
00014 typedef uint16_t MC_Torque;
00015 typedef uint16_t MC_RPM;
00016 typedef uint16_t MC_POWER;
00017
00026 // #define DEFAULT_PERIOD 50
00027
00033 enum map_mode{
00034     linear_speed_map,
00035     s_curve_speed_map,
00036     exp_speed_map,
00037     linear_torque_map,
00038     s_curve_torque_map,
00039     exp_torque_map
00040 };
00041
00042 enum DL_internal_state{
00043     Plausible = 0x01,
00044     Implausible = 0x02,
00045     Erroneous = 0x04
00046 };
00047
00048 typedef struct linear_torque_map_args{ //y = mx + b bitch
00049     int32_t offset;
```

```

00050     float slope;
00051 }linear_torque_map_args;
00052
00056 typedef struct exp_torque_map_args{
00057     int32_t offset;
00058     float gamma;
00059
00060
00061 }exp_torque_map_args;
00062
00066 typedef struct s_curve_torque_map_args{
00067     int32_t a;
00068     int32_t b;
00069     int32_t c[16];
00070 }s_curve_torque_map_args;
00071
00075 typedef union drivingModeParams{
00076     struct linear_torque_map_args;
00077     struct exp_torque_map_args;
00078     struct s_curve_torque_map_args;
00079 }drivingModeParams;
00080
00081
00085 typedef struct drivingMode{
00086     char dm_name[16];
00087     uint32_t max_acc_pwr;
00088     uint32_t max_motor_torque;
00089     uint32_t max_current;
00090
00091
00092     uint16_t flags;
00093
00094     drivingModeParams map_fn_params;
00095     uint8_t control_map_fn;
00096 }drivingMode;
00097
00098
00099
00100
00108 typedef struct driving_loop_args{
00109     uint32_t absolute_max_acc_pwr;
00110     uint32_t absolute_max_motor_torque;
00111     uint32_t absolute_max_accum_current;
00112     uint32_t max_accum_current_5s;
00115     uint16_t absolute_max_motor_rpm;
00116     uint16_t regen_rpm_cutoff;
00119 //uint8_t current_driving_mode; //what is the driving mode?
00120     uint16_t min_apps_offset;
00121     uint16_t max_apps_offset;
00122     uint16_t min_apps_value;
00123     uint16_t apps1_abs_max_val;
00124     uint16_t apps1_abs_min_val;
00125     uint16_t apps2_abs_min_val;
00126     uint16_t apps2_abs_max_val;
00127     uint16_t min_BPS_value;
00128     uint16_t max_BPS_value;
00130     uint16_t apps1_top;
00131     uint16_t apps1_bottom;
00133     uint16_t apps2_top;
00134     uint16_t apps2_bottom;
00135
00136     uint16_t apps_plausibility_check_threshold;
00137     uint16_t bps_plausibility_check_threshold;
00139     uint16_t bps_implausibility_recovery_threshold;
00140     uint16_t apps_implausibility_recovery_threshold;
00142     uint8_t num_driving_modes;
00143     uint8_t period;
00144     uint8_t accum_regen_soc_threshold;
00147     drivingMode dmodes[8];
00149 }driving_loop_args;
00150
00151 enum uv_status_t initDrivingLoop(void *argument);
00152
00153 void StartDrivingLoop(void *argument);
00154
00155 #endif /* INC_DRIVING_LOOP_H_ */

```

7.17 Core/Inc/errorLUT.h File Reference

Macros

- `#define _NUM_ERRORS_ 256`

7.17.1 Macro Definition Documentation

7.17.1.1 `_NUM_ERRORS_`

```
#define _NUM_ERRORS_ 256
```

Definition at line 11 of file [errorLUT.h](#).

7.18 errorLUT.h

[Go to the documentation of this file.](#)

```
00001 /*  
00002 * errorLUT.h  
00003 *  
00004 * Created on: Mar 7, 2024  
00005 * Author: byo10  
00006 */  
00007  
00008 #ifndef INC_ERRORLUT_H_  
00009 #define INC_ERRORLUT_H_  
00010  
00011 #define _NUM_ERRORS_ 256  
00012  
00013 #endif /* INC_ERRORLUT_H_ */
```

7.19 Core/Inc/gpio.h File Reference

This file contains all the function prototypes for the [gpio.c](#) file.

```
#include "main.h"
```

Include dependency graph for gpio.h: This graph shows which files directly or indirectly include this file:

Functions

- void [MX_GPIO_Init](#) (void)

7.19.1 Detailed Description

This file contains all the function prototypes for the [gpio.c](#) file.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [gpio.h](#).

7.19.2 Function Documentation

7.19.2.1 MX_GPIO_Init()

```
void MX_GPIO_Init (
    void )
```

Configure pins as Analog Input Output EVENT_OUT EXTI

Definition at line 42 of file [gpio.c](#).

References [DEBUG_LED_Pin](#).

Referenced by [main\(\)](#).

7.20 gpio.h

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -----*/
00021 #ifndef __GPIO_H__
00022 #define __GPIO_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes -----*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 /* USER CODE BEGIN Private defines */
00036
00037 /* USER CODE END Private defines */
00038
00039 void MX_GPIO_Init(void);
00040
00041 /* USER CODE BEGIN Prototypes */
00042
00043 /* USER CODE END Prototypes */
00044
00045 #ifdef __cplusplus
00046 }
00047 #endif
00048 #endif /*__ GPIO_H__ */
00049
```

7.21 Core/Inc/imd.h File Reference

```
#include "main.h"
```

Include dependency graph for imd.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [uv_imd_settings](#)

Typedefs

- `typedef struct uv_imd_settings uv_imd_settings`

Enumerations

- `enum imd_status_bits {
 Isolation_Status_bit0 = 0b00000001 , Isolation_Status_bit1 = 0b00000010 , Low_Battery_Voltage =
 0b00000100 , High_Battery_Voltage = 0b00001000 ,
 Exc_off = 0b00010000 , High_Uncertainty = 0b00100000 , Touch_energy_fault = 0b01000000 ,
 Hardware_Error = 0b10000000 }`
- `enum imd_status_requests {
 isolation_state = 0xE0 , isolation_resistances = 0xE1 , isolation_capacitances = 0xE2 , voltages_Vp_and_Vn
 = 0xE3 ,
 battery_voltage = 0xE4 , Error_flags = 0xE5 , safety_touch_energy = 0xE6 , safety_touch_current = 0xE7 ,
 Max_battery_working_voltage = 0xF0 , Temperature = 0x80 }`
- `enum imd_error_flags {
 Err_temp = 0x0080 , Err_clock = 0x0100 , Err_Watchdog = 0x0200 , Err_Vpwr = 0x0400 ,
 Err_Vxi = 0x0800 , Err_VxR = 0x1000 , Err_CH = 0x2000 , Err_Vx1 = 0x4000 ,
 Err_Vx2 = 0x8000 }`
- `enum imd_manufacturer_requests {
 Part_name_0 = 0x01 , Part_name_1 = 0x02 , Part_name_2 = 0x03 , Part_name_3 = 0x04 ,
 Version_0 = 0x05 , Version_1 = 0x06 , Version_2 = 0x07 , Serial_number_0 = 0x08 ,
 Serial_number_1 = 0x09 , Serial_number_2 = 0x0A , Serial_number_3 = 0x0B , Uptime_counter = 0x0C }`
- `enum imd_high_resolution_measurements {
 Vn_hi_res = 0x60 , Vp_hi_res = 0x61 , Vexc_hi_res = 0x62 , Vb_hi_res = 0x63 ,
 Vpwr_hi_res = 0x65 }`

Functions

- `void IMD_Parse_Message (int DLC, uint8_t Data[])`
- `void IMD_Check_Status_Bits (uint8_t Data)`
- `void IMD_Check_Error_Flags (uint8_t Data[])`
- `void IMD_Check_Isolation_State (uint8_t Data[])`
- `void IMD_Check_Isolation_Resistances (uint8_t Data[])`
- `void IMD_Check_Isolation_Capacitances (uint8_t Data[])`
- `void IMD_Check_Voltages_Vp_and_Vn (uint8_t Data[])`
- `void IMD_Check_Battery_Voltage (uint8_t Data[])`
- `void IMD_Check_Safety_Touch_Energy (uint8_t Data[])`
- `void IMD_Check_Safety_Touch_Current (uint8_t Data[])`
- `void IMD_Check_Temperature (uint8_t Data[])`
- `void IMD_Check_Max_Battery_Working_Voltage (uint8_t Data[])`
- `void IMD_Check_Part_Name (uint8_t Data[])`
- `void IMD_Check_Version (uint8_t Data[])`
- `void IMD_Check_Serial_Number (uint8_t Data[])`
- `void IMD_Check_Uptime (uint8_t Data[])`
- `void IMD_Request_Status (uint8_t Status)`
- `void IMD_Startup ()`
- `void initIMD (void *args)`

7.21.1 Typedef Documentation

7.21.1.1 uv_imd_settings

```
typedef struct uv_imd_settings uv_imd_settings
```

7.21.2 Enumeration Type Documentation

7.21.2.1 `imd_error_flags`

```
enum imd_error_flags
```

Enumerator

Err_temp	
Err_clock	
Err_Watchdog	
Err_Vpwr	
Err_Vexi	
Err_VxR	
Err_CH	
Err_Vx1	
Err_Vx2	

Definition at line [75](#) of file `imd.h`.

7.21.2.2 `imd_high_resolution_measurements`

```
enum imd_high_resolution_measurements
```

Enumerator

Vn_hi_res	
Vp_hi_res	
Vexc_hi_res	
Vb_hi_res	
Vpwr_hi_res	

Definition at line [105](#) of file `imd.h`.

7.21.2.3 `imd_manufacturer_requests`

```
enum imd_manufacturer_requests
```

Enumerator

Part_name_0	
Part_name_1	
Part_name_2	
Part_name_3	
Version_0	
Version_1	
Version_2	
Serial_number↔_0	

Enumerator

Serial_number_1	
Serial_number_2	
Serial_number_3	
Uptime_counter	

Definition at line 89 of file [imd.h](#).

7.21.2.4 imd_status_bits

```
enum imd_status_bits
```

Enumerator

Isolation_status_bit0	
Isolation_status_bit1	
Low_Battery_Voltage	
High_Battery_Voltage	
Exc_off	
High_Uncertainty	
Touch_energy_fault	
Hardware_Error	

Definition at line 16 of file [imd.h](#).

7.21.2.5 imd_status_requests

```
enum imd_status_requests
```

Enumerator

isolation_state	
isolation_resistances	
isolation_capacitances	
voltages_Vp_and_Vn	
battery_voltage	
Error_flags	
safety_touch_energy	
safety_touch_current	
Max_battery_working_voltage	
Temperature	

Definition at line 39 of file [imd.h](#).

7.21.3 Function Documentation

7.21.3.1 IMD_Check_Battery_Voltage()

```
void IMD_Check_Battery_Voltage (
    uint8_t Data[])
```

Definition at line 356 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.2 IMD_Check_Error_Flags()

```
void IMD_Check_Error_Flags (
    uint8_t Data[])
```

Definition at line 262 of file [imd.c](#).

References [Err_CH](#), [Err_clock](#), [Err_temp](#), [Err_Vexi](#), [Err_Vpwr](#), [Err_Vx1](#), [Err_Vx2](#), [Err_VxR](#), and [Err_Watchdog](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.3 IMD_Check_Isolation_Capacitances()

```
void IMD_Check_Isolation_Capacitances (
    uint8_t Data[])
```

Definition at line 342 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.4 IMD_Check_Isolation_Resistances()

```
void IMD_Check_Isolation_Resistances (
    uint8_t Data[])
```

Definition at line 317 of file [imd.c](#).

References [IMD_High_Uncertainty](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.5 IMD_Check_Isolation_State()

```
void IMD_Check_Isolation_State (
    uint8_t Data[])
```

Definition at line 301 of file [imd.c](#).

References [IMD_High_Uncertainty](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.6 IMD_Check_Max_Battery_Working_Voltage()

```
void IMD_Check_Max_Battery_Working_Voltage (
    uint8_t Data[ ])
```

Definition at line 393 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.7 IMD_Check_Part_Name()

```
void IMD_Check_Part_Name (
    uint8_t Data[ ])
```

Definition at line 406 of file [imd.c](#).

References [IMD_Expected_Part_Name](#), [IMD_Part_Name_0_Set](#), [IMD_Part_Name_1_Set](#), [IMD_Part_Name_2_Set](#), [IMD_Part_Name_3_Set](#), [IMD_Part_Name_Set](#), [IMD_Read_Part_Name](#), [Part_name_0](#), [Part_name_1](#), [Part_name_2](#), and [Part_name_3](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.8 IMD_Check_Safety_Touch_Current()

```
void IMD_Check_Safety_Touch_Current (
    uint8_t Data[ ])
```

Definition at line 381 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.9 IMD_Check_Safety_Touch_Energy()

```
void IMD_Check_Safety_Touch_Energy (
    uint8_t Data[ ])
```

Definition at line 374 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.10 IMD_Check_Serial_Number()

```
void IMD_Check_Serial_Number (
    uint8_t Data[ ])
```

Definition at line 488 of file [imd.c](#).

References [IMD_Expected_Serial_Number](#), [IMD_Read_Serial_Number](#), [IMD_Serial_Number_0_Set](#), [IMD_Serial_Number_1_Set](#), [IMD_Serial_Number_2_Set](#), [IMD_Serial_Number_3_Set](#), [IMD_Serial_Number_Set](#), [Serial_number_0](#), [Serial_number_1](#), [Serial_number_2](#), and [Serial_number_3](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.11 IMD_Check_Status_Bits()

```
void IMD_Check_Status_Bits (
    uint8_t Data)
```

Definition at line 218 of file [imd.c](#).

References [Error_flags](#), [Hardware_Error](#), [High_Battery_Voltage](#), [High_Uncertainty](#), [IMD_error_flags_requested](#), [IMD_High_Uncertainty](#), [IMD_Request_Status\(\)](#), [Isolation_status_bit0](#), [Isolation_status_bit1](#), and [Low_Battery_Voltage](#).

Referenced by [IMD_Parse_Message\(\)](#).

Here is the call graph for this function:

7.21.3.12 IMD_Check_Temperature()

```
void IMD_Check_Temperature (
    uint8_t Data[])
```

Definition at line 363 of file [imd.c](#).

References [IMD_Temperature](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.13 IMD_Check_Uptime()

```
void IMD_Check_Uptime (
    uint8_t Data[])
```

Definition at line 529 of file [imd.c](#).

7.21.3.14 IMD_Check_Version()

```
void IMD_Check_Version (
    uint8_t Data[])
```

Definition at line 448 of file [imd.c](#).

References [IMD_Expected_Version](#), [IMD_Read_Version](#), [IMD_Version_0_Set](#), [IMD_Version_1_Set](#), [IMD_Version_2_Set](#), [IMD_Version_Set](#), [Version_0](#), [Version_1](#), and [Version_2](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.15 IMD_Check_Voltages_Vp_and_Vn()

```
void IMD_Check_Voltages_Vp_and_Vn (
    uint8_t Data[])
```

Definition at line 349 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.21.3.16 IMD_Parse_Message()

```
void IMD_Parse_Message (
    int DLC,
    uint8_t Data[ ])
```

Definition at line 73 of file [imd.c](#).

References [battery_voltage](#), [Error_flags](#), [Error_Handler\(\)](#), [IMD_Check_Battery_Voltage\(\)](#), [IMD_Check_Error_Flags\(\)](#), [IMD_Check_Isolation_Capacitances\(\)](#), [IMD_Check_Isolation_Resistances\(\)](#), [IMD_Check_Isolation_State\(\)](#), [IMD_Check_Max_Battery_Working_Voltage\(\)](#), [IMD_Check_Part_Name\(\)](#), [IMD_Check_Safety_Touch_Current\(\)](#), [IMD_Check_Safety_Touch_Energy\(\)](#), [IMD_Check_Serial_Number\(\)](#), [IMD_Check_Status_Bits\(\)](#), [IMD_Check_Temperature\(\)](#), [IMD_Check_Version\(\)](#), [IMD_Check_Voltages_Vp_and_Vn\(\)](#), [isolation_capacitances](#), [isolation_resistances](#), [isolation_state](#), [Max_battery_working_voltage](#), [Part_name_0](#), [Part_name_1](#), [Part_name_2](#), [Part_name_3](#), [safety_touch_current](#), [safety_touch_energy](#), [Serial_number_0](#), [Serial_number_1](#), [Serial_number_2](#), [Serial_number_3](#), [Temperature](#), [Uptime_counter](#), [Vb_hi_res](#), [Version_0](#), [Version_1](#), [Version_2](#), [Vexc_hi_res](#), [Vn_hi_res](#), [voltages_Vp_and_Vn](#), [Vp_hi_res](#), and [Vpwr_hi_res](#).

Here is the call graph for this function:

7.21.3.17 IMD_Request_Status()

```
void IMD_Request_Status (
    uint8_t Status)
```

Definition at line 185 of file [imd.c](#).

References [Error_Handler\(\)](#), [hcan2](#), [IMD_CAN_ID_Tx](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Referenced by [IMD_Check_Status_Bits\(\)](#), and [IMD_Startup\(\)](#).

Here is the call graph for this function:

7.21.3.18 IMD_Startup()

```
void IMD_Startup ()
```

Definition at line 533 of file [imd.c](#).

References [IMD_Request_Status\(\)](#), [isolation_state](#), [Max_battery_working_voltage](#), [Part_name_0](#), [Part_name_1](#), [Part_name_2](#), [Part_name_3](#), [Serial_number_0](#), [Serial_number_1](#), [Serial_number_2](#), [Serial_number_3](#), [Version_0](#), [Version_1](#), and [Version_2](#).

Here is the call graph for this function:

7.21.3.19 initIMD()

```
void initIMD (
    void * args)
```

Definition at line 559 of file [imd.c](#).

References [IMD](#), [uv_init_task_args::init_info_queue](#), [uv_init_task_args::meta_task_handle](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

7.22 imd.h

[Go to the documentation of this file.](#)

```

00001 // Code to make human readable CAN messages for the device
00002
00003 // This file will define message data as human readable stuff
00004
00005 #include "main.h"
00006
00007 #ifndef __IMD_H__
00008 #define __IMD_H__
00009
00010 // CAN ID is currently extended
00011
00012 // Needs to be changed to standard ID
00013
00014 // Faults
00015 // The first byte returned from a status request message will have 8 status bits and the data as the
     rest
00016 enum imd_status_bits{
00017     // I have just left this as plain binary. Could also be declared by bitshifting but I think this
     is easier
00018     // 12345678
00019     Isolation_status_bit0 = 0b00000001, // ISO
00020     Isolation_status_bit1 = 0b00000010, // IS1
00021     Low_Battery_Voltage = 0b00000100, // LV 15V threshold (1 if below 15V)
00022     High_Battery_Voltage = 0b00001000, // HV (1 if higher than Max_battery_working_voltage)
00023     Exc_off = 0b00010000, // EO (Excitation pulse 0 for operating)
00024     High_Uncertainty = 0b00100000, // HU (1 for greater than 5%)
00025     Touch_energy_fault = 0b01000000, // EF (1 for energy exceeds 0.2J)
00026     Hardware_Error = 0b10000000, // HE (1 for error)
00027 };
00028
00029 typedef struct uv_imd_settings{
00030     uint16_t min_isolation_resistances;
00031     uint16_t expected_isolation_capacitances;
00032     uint16_t max_imd_temperature;
00033
00034 }uv_imd_settings;
00035
00036
00037 // The MCU needs to send a message to the IMD requesting info
00038 // These are requests that will return status bits (defined above) & the value requested
00039 enum imd_status_requests{
00040     // The electrical isolation is in bytes 2 & 3
00041     isolation_state = 0xE0,
00042
00043     // Rp is the resistance from the positive of the battery to chassis
00044     // Rn is the resistance from the negative of the battery to chassis
00045     isolation_resistances = 0xE1,
00046
00047     // Cp is the capacitance from the positive of the battery to chassis
00048     // Cn is the capacitance from the negative of the battery to chassis
00049     // Bytes 2 & 3 are Cp, bytes 5 & 6 are Cn, in nF
00050     isolation_capacitances = 0xE2,
00051
00052     // High voltage battery voltages to chassis
00053     voltages_Vp_and_Vn = 0xE3,
00054
00055     // GLV battery voltage
00056     battery_voltage = 0xE4,
00057
00058     // Error flags are a series of bits described below
00059     Error_flags = 0xE5,
00060
00061     // The IMD monitors charge stored in the system
00062     safety_touch_energy = 0xE6,
00063
00064     // Also monitors if it is safe to touch
00065     safety_touch_current = 0xE7,
00066
00067     // This is just a parameter we can set
00068     Max_battery_working_voltage = 0xF0,
00069
00070     // We can read the temperature of the board
00071     Temperature = 0x80,
00072 };
00073
00074 // If one of the error flags is set, then the hardware error bit will go to 1
00075 enum imd_error_flags{
00076     // Bits 0-6 are reserved
00077     Err_temp = 0x0080, // HT high temperature (0 if temperature is below 105C)
00078     Err_clock = 0x0100, // CE (0 if clock is good)
00079     Err_Watchdog = 0x0200, // WD Watchdog (0 if watchdog is good)
00080     Err_Vpwr = 0x0400, // V_PWR power supply (0 if power supply voltage is good)

```

```

00081     Err_Vxi = 0x0800, // V_EXI Excitation voltage (0 if excitation voltage is good)
00082     Err_VxR = 0x1000, // VxR (0 if both Vx1 and Vx2 are good)
00083     Err_CH = 0x2000, // CH chassis connections (0 if both chassis connections are good)
00084     Err_Vx1 = 0x4000, // Vx1 connection (0 if SIM101 connection to Hv+ is good)
00085     Err_Vx2 = 0x8000, // Vx2 connection (0 if SIM101 connection to Hv- is good)
00086 };
00087
00088 // Probably not useful and won't be used for a bit but may be useful in the future
00089 enum imd_manufacturer_requests{
00090     // The numbers get broken up over multiple bytes
00091     Part_name_0 = 0x01,
00092     Part_name_1 = 0x02,
00093     Part_name_2 = 0x03,
00094     Part_name_3 = 0x04,
00095     Version_0 = 0x05,
00096     Version_1 = 0x06,
00097     Version_2 = 0x07,
00098     Serial_number_0 = 0x08,
00099     Serial_number_1 = 0x09,
00100    Serial_number_2 = 0x0A,
00101    Serial_number_3 = 0x0B,
00102    Uptime_counter = 0x0C,
00103 };
00104
00105 enum imd_high_resolution_measurements{
00106     Vn_hi_res = 0x60,
00107     Vp_hi_res = 0x61,
00108     Vexc_hi_res = 0x62,
00109     Vb_hi_res = 0x63,
00110     Vpwr_hi_res = 0x65,
00111 };
00112
00113 // -----
00114 // Function declarations
00115
00116 // This will parse the data received from CAN message
00117 void IMD_Parse_Message(int DLC, uint8_t Data[]);
00118
00119
00120 // Functions to check states are okay
00121 void IMD_Check_Status_Bits(uint8_t Data);
00122 void IMD_Check_Error_Flags(uint8_t Data[]);
00123
00124 // Functions to check values are okay
00125 void IMD_Check_Isolation_State(uint8_t Data[]);
00126 void IMD_Check_Isolation_Resistances(uint8_t Data[]);
00127 void IMD_Check_Isolation_Capacitances(uint8_t Data[]);
00128 void IMD_Check_Voltages_Vp_and_Vn(uint8_t Data[]);
00129 void IMD_Check_Battery_Voltage(uint8_t Data[]);
00130 void IMD_Check_Safety_Touch_Energy(uint8_t Data[]);
00131 void IMD_Check_Safety_Touch_Current(uint8_t Data[]);
00132 void IMD_Check_Temperature(uint8_t Data[]);
00133
00134 // Functions to check on startup
00135 void IMD_Check_Max_Battery_Working_Voltage(uint8_t Data[]);
00136 void IMD_Check_Part_Name(uint8_t Data[]);
00137 void IMD_Check_Version(uint8_t Data[]);
00138 void IMD_Check_Serial_Number(uint8_t Data[]);
00139 void IMD_Check_Uptime(uint8_t Data[]);
00140
00141 // High resolution measurements
00142
00143
00144 // Function to request data from the IMD
00145 void IMD_Request_Status(uint8_t Status);
00146
00147 // called on startup @deprecated
00148 void IMD_Startup();
00149
00150
00151 void initIMD(void* args);
00152
00153
00154
00155 #endif

```

7.23 Core/Inc/main.h File Reference

: Header for [main.c](#) file. This file contains the common defines of the application.

```
#include "stm32f4xx_hal.h"
#include <stdarg.h>
```

```
#include "uvfr_utils.h"
```

Include dependency graph for main.h: This graph shows which files directly or indirectly include this file:

Macros

- #define DEBUG_LED_Pin GPIO_PIN_12
- #define DEBUG_LED_GPIO_Port GPIOC

Functions

- void [Error_Handler](#) (void)
This function is executed in case of error occurrence.

7.23.1 Detailed Description

: Header for [main.c](#) file. This file contains the common defines of the application.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [main.h](#).

7.23.2 Macro Definition Documentation

7.23.2.1 DEBUG_LED_GPIO_Port

```
#define DEBUG_LED_GPIO_Port GPIOC
```

Definition at line [65](#) of file [main.h](#).

7.23.2.2 DEBUG_LED_Pin

```
#define DEBUG_LED_Pin GPIO_PIN_12
```

Definition at line [64](#) of file [main.h](#).

Referenced by [MX_GPIO_Init\(\)](#).

7.23.3 Function Documentation

7.23.3.1 Error_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

None	
------	--

Definition at line 365 of file [main.c](#).

Referenced by [HAL_ADC_MspInit\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [IMD_Parse_Message\(\)](#), [IMD_Request_Status\(\)](#), [initADCTask\(\)](#), [MX_ADC1_Init\(\)](#), [MX_ADC2_Init\(\)](#), [MX_ADC3_Init\(\)](#), [MX_CAN1_Init\(\)](#), [MX_CAN2_Init\(\)](#), [MX_TIM11_Init\(\)](#), [MX_TIM3_Init\(\)](#), [SystemClock_Config\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.24 main.h

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -----*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes -----*/
00030 #include "stm32f4xx_hal.h"
00031
00032 /* Private includes -----*/
00033 /* USER CODE BEGIN Includes */
00034 #include<stdarg.h>
00035 #include"uvfr_utils.h"
00036
00037 /* USER CODE END Includes */
00038
00039 /* Exported types -----*/
00040 /* USER CODE BEGIN ET */
00041
00042
00043
00044 /* USER CODE END ET */
00045
00046 /* Exported constants -----*/
00047 /* USER CODE BEGIN EC */
00048
00049 /* USER CODE END EC */
00050
00051 /* Exported macro -----*/
00052 /* USER CODE BEGIN EM */
00053
00054 /* USER CODE END EM */
00055
00056 /* Exported functions prototypes -----*/
00057 void Error_Handler(void);
00058
00059 /* USER CODE BEGIN EFP */
00060 //void Set_Vehicle_State(enum vehicle_state_t new_state);
00061 /* USER CODE END EFP */
00062
00063 /* Private defines -----*/
00064 #define DEBUG_LED_Pin GPIO_PIN_12
00065 #define DEBUG_LED_GPIO_Port GPIOC
00066
00067 /* USER CODE BEGIN Private defines */
00068
00069
00070
00071
00072 /* USER CODE END Private defines */
00073
00074 #ifdef __cplusplus
00075 }
00076 #endif
00077
00078 #endif /* __MAIN_H */
```

7.25 Core/Inc/motor_controller.h File Reference

```
#include "main.h"
#include "freeRTOS.h"
#include "uvfr_utils.h"
#include "uvfr_settings.h"
#include "can.h"
```

Include dependency graph for motor_controller.h: This graph shows which files directly or indirectly include this file:

Macros

- #define SERIAL_NUMBER_REGISTER 0x62
- #define FIRMWARE_VERSION_REGISTER 0x1B
- #define N_set_cmd N_set

Typedefs

- typedef struct motor_controller_settings motor_controller_settings

Enumerations

- enum motor_controller_speed_parameters {
 N_actual = 0x30 , N_set = 0x31 , M_set = 0x90 , N_cmd = 0x32 ,
 N_error = 0x33 , M_out = 0xA0
 }
- enum motor_controller_status { LOGIMAP_ERRORS = 0x82 , LOGIMAP_IO = 0x83 , POS_ACTUAL = 0x86
 }
- enum motor_controller_current_parameters { CURRENT_ACTUAL = 0x69 }
- enum motor_controller_motor_constants {
 nominal_motor_frequency = 0x05 , nominal_motor_voltage = 0x06 , power_factor = 0x0E , motor_max_current
 = 0x4D ,
 motor_continuous_current = 0x4E , motor_pole_number = 0x4F , motor_kt_constant = 0x87 ,
 motor_ke_constant = 0x87 ,
 rated_motor_speed = 0x59 , motor_temperature_switch_off_point = 0xA3 , stator_leakage_inductance = 0xB1 ,
 nominal_magnitizing_current = 0xB2 ,
 motor_magnetising_inductance = 0xB3 , rotor_resistance = 0xB4 , minimum_magnetising_current = 0xB5 ,
 time_constant_rotor = 0xB6 ,
 leakage_inductance_ph_ph = 0xBB , stator_resistance_ph_ph = 0xBC , time_constant_stator = 0xBD }
- enum motor_controller_temperatures {
 igt_temperature = 0x4A , max_motor_temp = 0xA3 , warning_motor_temp = 0xA2 , motor_temperature = 0x49 ,
 air_temperature = 0x4B , current_derate_temperature = 0x4C , temp_sensor_pt1 = 0x9C , temp_sensor_pt2
 = 0x9D ,
 temp_sensor_pt3 = 0x9E , temp_sensor_pt4 = 0x9F }
- enum motor_controller_measurements { DC_bus_voltage = 0xEB }
- enum motor_controller_status_information_errors_warnings {
 motor_controller_errors_warnings = 0x8F , eeprom_read_error = 1 << 8 , hardware_fault = 1 << 9 ,
 rotate_field_enable_not_present_run = 1 << 10 ,
 CAN_timeout_error = 1 << 11 , feedback_signal_error = 1 << 12 , mains_voltage_min_limit = 1 << 13 ,
 motor_temp_max_limit = 1 << 14 ,
 IGBT_temp_max_limit = 1 << 15 , mains_voltage_max_limit = 1 , critical_AC_current = 1 << 1 ,
 race_away_detected = 1 << 2 ,
 ecode_timeout_error = 1 << 3 , watchdog_reset = 1 << 4 , AC_current_offset_fault = 1 << 5 ,
 internal_hardware_voltage_problem = 1 << 6 ,

```

bleed_resistor_overload = 1 << 7 , parameter_conflict_detected = 1 << 8 , special_CPU_fault = 1 << 9 ,
rotate_field_enable_not_present_norun = 1 << 10 ,
auxiliary_voltage_min_limit = 1 << 11 , feedback_signal_problem = 1 << 12 , warning_5 = 1 << 13 ,
motor_temperature_warning = 1 << 14 ,
IGBT_temperature_warning = 1 << 15 , Vout_saturation_max_limit = 1 , warning_9 = 1 << 1 ,
speed_actual_resolution_limit = 1 << 2 ,
check_ecode_ID = 1 << 3 , tripzone_glitch_detected = 1 << 4 , ADC_sequencer_problem = 1 << 5 ,
ADC_measurement_problem = 1 << 6 ,
bleeder_resistor_warning = 1 << 7 }
• enum motor_controller_io { todo6969 = 6969 }
• enum motor_controller_PI_values {
    accelerate_ramp = 0x35 , dismantling_ramp = 0xED , recuperation_ramp = 0xC7 , proportional_gain = 0x1C
    ,
    integral_time_constant = 0x1D , integral_memory_max = 0x2B , proportional_gain_2 = 0xC9 ,
    current_feed_forward = 0xCB ,
    ramp_set_current = 0x25 }
• enum motor_controller_repeating_time { none = 0 , one_hundred_ms = 0x64 }
• enum motor_controller_limp_mode { N_lim = 0x34 , N_lim_plus = 0x3F , N_lim_minus = 0x3E }
• enum motor_controller_startup { clear_errors = 0x8E , firmware_version = 0x1B }

```

Functions

- void **MC_Startup** (void *args)
Initializes the motor controller.
- uint16_t **sendTorqueToMotorController** (float T_filtered)
Sends a filtered torque command to the motor controller over CAN.
- void **MC_Request_Data** (uint8_t RegID)
Sends a CAN request to retrieve a specific register from the motor controller.
- void **ProcessMotorControllerResponse** (uv_CAN_msg *msg)
Processes a CAN response from the motor controller.
- void **Parse_Bamocar_Response** (uv_CAN_msg *msg)
Parses a 32-bit value from a CAN message in little-endian format.
- void **MC_Shutdown** (void)
Safely shuts down the motor controller.

Variables

- int16_t mc_speed_rpm
- int16_t mc_current
- int16_t mc_torque_cmd
- int16_t mc_motor_temp
- int16_t mc_igbt_temp
- TickType_t last_driver_input_time
- motor_controller_settings mc_default_settings

7.25.1 Macro Definition Documentation

7.25.1.1 FIRMWARE_VERSION_REGISTER

```
#define FIRMWARE_VERSION_REGISTER 0x1B
```

Definition at line 177 of file [motor_controller.h](#).

Referenced by [MC_Startup\(\)](#).

7.25.1.2 N_set_cmd

```
#define N_set_cmd N_set
```

Definition at line 181 of file [motor_controller.h](#).

7.25.1.3 SERIAL_NUMBER_REGISTER

```
#define SERIAL_NUMBER_REGISTER 0x62
```

Definition at line 176 of file [motor_controller.h](#).

Referenced by [MC_Startup\(\)](#).

7.25.2 Typedef Documentation

7.25.2.1 motor_controller_settings

```
typedef struct motor_controller_settings motor_controller_settings
```

Definition at line 10 of file [motor_controller.h](#).

7.25.3 Enumeration Type Documentation

7.25.3.1 motor_controller_current_parameters

```
enum motor_controller_current_parameters
```

Enumerator

CURRENT_ACTUAL		
----------------	--	--

Definition at line 45 of file [motor_controller.h](#).

7.25.3.2 motor_controller_io

```
enum motor_controller_io
```

Enumerator

todo6969		
----------	--	--

Definition at line 136 of file [motor_controller.h](#).

7.25.3.3 motor_controller_limp_mode

```
enum motor_controller_limp_mode
```

Enumerator

N_lim	
N_lim_plus	
N_lim_minus	

Definition at line 160 of file [motor_controller.h](#).

7.25.3.4 motor_controller_measurements

```
enum motor_controller_measurements
```

Enumerator

DC_bus_voltage	
----------------	--

Definition at line 89 of file [motor_controller.h](#).

7.25.3.5 motor_controller_motor_constants

```
enum motor_controller_motor_constants
```

Enumerator

nominal_motor_frequency	
nominal_motor_voltage	
power_factor	
motor_max_current	
motor_continuous_current	
motor_pole_number	
motor_kt_constant	
motor_ke_constant	
rated_motor_speed	
motor_temperature_switch_off_point	
stator_leakage_inductance	
nominal_magnitizing_current	
motor_magnetising_inductance	
rotor_resistance	
minimum_magnetising_current	
time_constant_rotor	
leakage_inductance_ph_ph	
stator_resistance_ph_ph	
time_constant_stator	

Definition at line 50 of file [motor_controller.h](#).

7.25.3.6 motor_controller_PI_values

```
enum motor_controller_PI_values
```

Enumerator

accelerate_ramp	
dismantling_ramp	
recuperation_ramp	
proportional_gain	
integral_time_constant	
integral_memory_max	
proportional_gain_2	
current_feed_forward	
ramp_set_current	

Definition at line 141 of file [motor_controller.h](#).

7.25.3.7 motor_controller_repeating_time

```
enum motor_controller_repeating_time
```

Enumerator

none	
one_hundred_ms	

Definition at line 154 of file [motor_controller.h](#).

7.25.3.8 motor_controller_speed_parameters

```
enum motor_controller_speed_parameters
```

Enumerator

N_actual	
N_set	
M_set	
N_cmd	
N_error	
M_out	

Definition at line 26 of file [motor_controller.h](#).

7.25.3.9 motor_controller_startup

```
enum motor_controller_startup
```

Enumerator

clear_errors	
firmware_version	

Definition at line 167 of file [motor_controller.h](#).

7.25.3.10 motor_controller_status

```
enum motor_controller_status
```

Enumerator

LOGIMAP_ERRORS	
LOGIMAP_IO	
POS_ACTUAL	

Definition at line 37 of file [motor_controller.h](#).

7.25.3.11 motor_controller_status_information_errors_warnings

```
enum motor_controller_status_information_errors_warnings
```

Enumerator

motor_controller_errors_warnings	
eprom_read_error	
hardware_fault	
rotate_field_enable_not_present_run	
CAN_timeout_error	
feedback_signal_error	
mains_voltage_min_limit	
motor_temp_max_limit	
IGBT_temp_max_limit	
mains_voltage_max_limit	
critical_AC_current	
race_away_detected	
ecode_timeout_error	
watchdog_reset	
AC_current_offset_fault	
internal_hardware_voltage_problem	
bleed_resistor_overload	
parameter_conflict_detected	
special_CPU_fault	
rotate_field_enable_not_present_norun	
auxiliary_voltage_min_limit	
feedback_signal_problem	
warning_5	
motor_temperature_warning	
IGBT_temperature_warning	
Vout_saturation_max_limit	
warning_9	
speed_actual_resolution_limit	
check_ecode_ID	
tripzone_glitch_detected	
ADC_sequencer_problem	
ADC_measurement_problem	
bleeder_resistor_warning	

Definition at line 94 of file [motor_controller.h](#).

7.25.3.12 motor_controller_temperatures

```
enum motor_controller_temperatures
```

Enumerator

igbt_temperature	
max_motor_temp	
warning_motor_temp	
motor_temperature	
air_temperature	
current_derate_temperature	
temp_sensor_pt1	
temp_sensor_pt2	
temp_sensor_pt3	
temp_sensor_pt4	

Definition at line 73 of file [motor_controller.h](#).

7.25.4 Function Documentation

7.25.4.1 MC_Request_Data()

```
void MC_Request_Data (
    uint8_t RegID)
```

Sends a CAN request to retrieve a specific register from the motor controller.

The request message is formatted as: [0x3D, RegID, 0], which should trigger an immediate reply.

Definition at line 110 of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [mc_settings](#), [uv_CAN_msg::msg_id](#), [UV_OK](#), and [uvSendCanMSG\(\)](#).

Referenced by [MC_SetAndVerify_Param\(\)](#), [MC_Shutdown\(\)](#), and [MC_Startup\(\)](#).

Here is the call graph for this function:

7.25.4.2 MC_Shutdown()

```
void MC_Shutdown (
    void )
```

Safely shuts down the motor controller.

This function stops all periodic CAN feedback, zeroes the torque or speed command (depending on control mode), disables the motor controller, and optionally requests the error/warning register for post-shutdown diagnostics.

It ensures the controller is left in a known, safe state after driving stops. Use this before powering down or transitioning to an inactive state.

Definition at line 548 of file [motor_controller.c](#).

References [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), and [motor_controller_errors_warnings](#).

Referenced by [performSafetyChecks\(\)](#), and [uvSecureVehicle\(\)](#).

Here is the call graph for this function:

7.25.4.3 MC_Startup()

```
void MC_Startup (
    void * args)
```

Initializes the motor controller.

This function performs all necessary startup steps to prepare the Bamocar motor controller for operation. It is typically called during system initialization from [uvfr_utils.c](#).

The routine performs the following actions:

- Toggles a GPIO pin for debug indication
- Registers a CAN RX handler for controller responses
- Enables cyclic transmission of critical feedback parameters
- Sends initialization commands (based on Bamocar Example 11)
- Requests metadata (serial number, firmware version)
- Optionally sets and verifies controller parameters
- Sends status to the system init queue
- Suspends itself after completion

Parameters

<i>args</i>	Pointer to uv_init_task_args , used to send back init status.
-------------	---

Definition at line [458](#) of file [motor_controller.c](#).

References [uv_init_task_response::device](#), [FIRMWARE_VERSION_REGISTER](#), [uv_init_task_args::init_info_queue](#), [insertCANMessageHandler\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [mc_settings](#), [MOTOR_CONTROLLER](#), [ProcessMotorControllerResponse\(\)](#), [SERIAL_NUMBER_REGISTER](#), [uv_init_task_response::status](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.25.4.4 Parse_Bamocar_Response()

```
void Parse_Bamocar_Response (
    uv_CAN_msg * msg)
```

Parses a 32-bit value from a CAN message in little-endian format.

This example assumes that the data bytes are stored as: data[0] = LSB, data[3] = MSB.

Definition at line [221](#) of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), and [uv_CAN_msg::dlc](#).

7.25.4.5 ProcessMotorControllerResponse()

```
void ProcessMotorControllerResponse (
    uv_CAN_msg * msg)
```

Processes a CAN response from the motor controller.

This function decodes a received CAN message by examining the register ID in the first byte (data[0]) and handling the response accordingly.

It performs little-endian parsing to extract values for speed, current, torque, temperatures, and error flags. Critical errors will trigger uvPanic() via the error handler.

The full message is also stored in a global buffer (last_mc_response) for later access and verification routines.

Parameters

<i>msg</i>	Pointer to the received CAN message from the motor controller.
------------	--

Definition at line 324 of file [motor_controller.c](#).

References [CURRENT_ACTUAL](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [igbt_temperature](#), [last_mc_response](#), [LOGIMAP_ERRORS](#), [LOGIMAP_IO](#), [M_out](#), [mc_current](#), [mc_igbt_temp](#), [mc_motor_temp](#), [mc_speed_rpm](#), [mc_torque_cmd](#), [motor_controller_errors_warnings](#), [motor_temperature](#), [N_actual](#), and [POS_ACTUAL](#).

Referenced by [MC_Startup\(\)](#).

7.25.4.6 sendTorqueToMotorController()

```
uint16_t sendTorqueToMotorController (
    float T_filtered)
```

Sends a filtered torque command to the motor controller over CAN.

This function scales the input torque (in Nm) to the Bamocar's expected format, clamps it within the valid operating range, and transmits it as a direct torque command using the N_set register. The final 16-bit value is sent via CAN using the configured transmit ID.

Parameters

<i>T_filtered</i>	The final torque value in Nm after filtering (typically 0–230 Nm).
-------------------	--

Returns

0 if successful, 1 if transmission failed.

Definition at line 69 of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [mc_settings](#), [uv_CAN_msg::msg_id](#), [N_set](#), [UV_OK](#), and [uvSendCanMSG\(\)](#).

Referenced by [StartDrivingLoop\(\)](#).

Here is the call graph for this function:

7.25.5 Variable Documentation

7.25.5.1 last_driver_input_time

```
TickType_t last_driver_input_time [extern]
```

Definition at line 24 of file [motor_controller.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.25.5.2 mc_current

```
int16_t mc_current [extern]
```

Definition at line 28 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.25.5.3 mc_default_settings

```
motor_controller_settings mc_default_settings [extern]
```

Definition at line 38 of file [motor_controller.c](#).

7.25.5.4 mc_igbt_temp

```
int16_t mc_igbt_temp [extern]
```

Definition at line 31 of file [motor_controller.c](#).

7.25.5.5 mc_motor_temp

```
int16_t mc_motor_temp [extern]
```

Definition at line 30 of file [motor_controller.c](#).

7.25.5.6 mc_speed_rpm

```
int16_t mc_speed_rpm [extern]
```

Definition at line 27 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.25.5.7 mc_torque_cmd

```
int16_t mc_torque_cmd [extern]
```

Definition at line 29 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.26 motor_controller.h

[Go to the documentation of this file.](#)

```

00001 #ifndef __MOTOR_CONTROLLER_H__
00002 #define __MOTOR_CONTROLLER_H__
00003
00004 #include "main.h"
00005 #include "freeRTOS.h"
00006 #include "uvfr_utils.h"
00007 #include "uvfr_settings.h"
00008 #include "can.h"
00009
00010 typedef struct motor_controller_settings motor_controller_settings;
00011 typedef struct uv_CAN_msg uv_CAN_msg;
00012
00013 //cyclic parameters for other files
00014 extern int16_t mc_speed_rpm;
00015 extern int16_t mc_current;
00016 extern int16_t mc_torque_cmd;
00017 extern int16_t mc_motor_temp;
00018 extern int16_t mc_igbt_temp;
00019
00020
00021 extern TickType_t last_driver_input_time;
00022
00023 /* Enums for CAN register IDs and other constants */
00024
00025 /* Speed parameters: */
00026 enum motor_controller_speed_parameters {
00027     N_actual = 0x30, // actual motor speed in rpm (16-bit, little-endian)
00028     N_set = 0x31, // setpoint (used for speed command in our case)
00029     M_set = 0x90, //setpoint (used for torque command in our case)
00030     N_cmd = 0x32, // command speed after ramp
00031
00032     N_error = 0x33, // speed error
00033     M_out = 0xA0 // actual active current scaled
00034 };
00035
00036
00037 enum motor_controller_status {
00038     LOGIMAP_ERRORS = 0x82, // Error bitfield
00039     LOGIMAP_IO = 0x83, // I/O status bitfield
00040     POS_ACTUAL = 0x86, // 32-bit position
00041     //motor_controller_errors_warnings = 0x8F // Some controllers use 0x8F for combined
00042     errors/warnings
00043 };
00044 /* Current parameters */
00045 enum motor_controller_current_parameters {
00046     CURRENT_ACTUAL = 0x69
00047 };
00048
00049 /* Motor constants */
00050 enum motor_controller_motor_constants {
00051     nominal_motor_frequency = 0x05,
00052     nominal_motor_voltage = 0x06,
00053     power_factor = 0x0E,
00054     motor_max_current = 0x4D,
00055     motor_continuous_current = 0x4E,
00056     motor_pole_number = 0x4F,
00057     motor_kt_constant = 0x87, // low
00058     motor_ke_constant = 0x87, // high, back emf
00059     rated_motor_speed = 0x59,
00060     motor_temperature_switch_off_point = 0xA3,
00061     stator_leakage_inductance = 0xB1,
00062     nominal_magnetizing_current = 0xB2,
00063     motor_magnetising_inductance = 0xB3,
00064     rotor_resistance = 0xB4,
00065     minimum_magnetising_current = 0xB5,
00066     time_constant_rotor = 0xB6,
00067     leakage_inductance_ph_ph = 0xBB,
00068     stator_resistance_ph_ph = 0xBC,
00069     time_constant_stator = 0xBD
00070 };
00071
00072 /* Temperatures */
00073 enum motor_controller_temperatures {
00074     igtb_temperature = 0x4A,
00075     //M-temp
00076     max_motor_temp = 0xA3, //set new limit for max motor temp
00077     warning_motor_temp = 0xA2, //motor temp limit for warnings
00078     motor_temperature = 0x49, //current motor temp
00079
00080     air_temperature = 0x4B,
00081     current_dereate_temperature = 0x4C,

```

```

00082     temp_sensor_pt1      = 0x9C,
00083     temp_sensor_pt2      = 0x9D,
00084     temp_sensor_pt3      = 0x9E,
00085     temp_sensor_pt4      = 0x9F
00086 };
00087
00088 /* Measurements */
00089 enum motor_controller_measurements {
00090     DC_bus_voltage = 0xEB
00091 };
00092
00093 /* Status, error, and warning flags */
00094 enum motor_controller_status_information_errors_warnings {
00095     motor_controller_errors_warnings = 0x8F, // used as register ID for error/warning response
00096
00097     /* Errors (assume low 16 bits) */
00098     eeprom_read_error          = 1 << 8,
00099     hardware_fault            = 1 << 9,
00100     rotate_field_enable_not_present_run= 1 << 10,
00101     CAN_timeout_error         = 1 << 11,
00102     feedback_signal_error     = 1 << 12,
00103     mains_voltage_min_limit   = 1 << 13,
00104     motor_temp_max_limit      = 1 << 14,
00105     IGBT_temp_max_limit       = 1 << 15,
00106
00107     /* Additional errors/warnings (often in high 16 bits) */
00108     mains_voltage_max_limit    = 1,
00109     critical_AC_current        = 1 << 1,
00110     race_away_detected         = 1 << 2,
00111     ecode_timeout_error        = 1 << 3,
00112     watchdog_reset            = 1 << 4,
00113     AC_current_offset_fault    = 1 << 5,
00114     internal_hardware_voltage_problem = 1 << 6,
00115     bleed_resistor_overload    = 1 << 7,
00116     parameter_conflict_detected = 1 << 8,
00117     special_CPU_fault          = 1 << 9,
00118     rotate_field_enable_not_present_norun = 1 << 10,
00119     auxiliary_voltage_min_limit = 1 << 11,
00120     feedback_signal_problem    = 1 << 12,
00121     warning_5                 = 1 << 13,
00122     motor_temperature_warning  = 1 << 14,
00123     IGBT_temperature_warning   = 1 << 15,
00124     Vout_saturation_max_limit = 1,
00125     warning_9                 = 1 << 1,
00126     speed_actual_resolution_limit = 1 << 2,
00127     check_ecode_ID             = 1 << 3,
00128     tripzone_glitch_detected  = 1 << 4,
00129     ADC_sequencer_problem     = 1 << 5,
00130     ADC_measurement_problem   = 1 << 6,
00131     bleeder_resistor_warning  = 1 << 7
00132 };
00133 };
00134
00135 /* I/O enum (placeholder) */
00136 enum motor_controller_io {
00137     todo6969 = 6969
00138 };
00139
00140 /* PI control values */
00141 enum motor_controller_PI_values {
00142     accelerate_ramp          = 0x35,
00143     dismantling_ramp          = 0xED,
00144     recuperation_ramp         = 0xC7,
00145     proportional_gain          = 0x1C, // may change based on mode
00146     integral_time_constant    = 0x1D,
00147     integral_memory_max        = 0x2B,
00148     proportional_gain_2         = 0xC9,
00149     current_feed_forward       = 0xCB,
00150     ramp_set_current           = 0x25
00151 };
00152
00153 /* Repeating time values */
00154 enum motor_controller_repeating_time {
00155     none                      = 0,
00156     one_hundred_ms             = 0x64
00157 };
00158
00159 /* Limp mode values */
00160 enum motor_controller_limp_mode {
00161     N_lim                     = 0x34,
00162     N_lim_plus                 = 0x3F,
00163     N_lim_minus                = 0x3E
00164 };
00165
00166 /* Startup-related registers */
00167 enum motor_controller_startup {
00168     clear_errors               = 0x8E,

```

```

00169     firmware_version = 0x1B,
00170 //    serial_number = 0x62
00171 };
00172
00173 /* Additional register IDs for requests */
00174 //#define SERIAL_NUMBER_REGISTER      0x1B // adjust if needed
00175 //#define FIRMWARE_VERSION_REGISTER  0x1E // adjust if needed
00176 #define SERIAL_NUMBER_REGISTER      0x62 // adjust if needed
00177 #define FIRMWARE_VERSION_REGISTER   0x1B // adjust if needed
00178
00179 /* Command identifier used for torque command.
00180   (For example, you might use N_set from motor_controller_speed_parameters.) */
00181 #define N_set_cmd  N_set
00182
00183 /*
00184  * Declare an extern for your global mc_default_settings so other files can use it.
00185  * The actual definition will be in motor_controller.c.
00186  */
00187 extern motor_controller_settings mc_default_settings;
00188
00189
00190 /* Function prototypes that will be called by other modules (driving_loop, init, etc.) */
00191 void MC_Startup(void* args);
00192 uint16_t sendTorqueToMotorController(float T_filtered);
00193 void MC_Request_Data(uint8_t RegID);
00194 void ProcessMotorControllerResponse(uv_CAN_msg* msg);
00195 void Parse_Bamocar_Response(uv_CAN_msg* msg);
00196
00197 void MC_Shutdown(void);
00198
00199
00200 #endif /* __MOTOR_CONTROLLER_H__ */

```

7.27 Core/Inc/odometer.h File Reference

#include "uvfr_utils.h"

Include dependency graph for odometer.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [uv_persistent_data_frame](#)

TypeDefs

- typedef struct uv_persistent_data_frame [uv_persistent_data_frame](#)

Functions

- [uv_status initOdometer \(void *args\)](#)
- [void odometerTask \(void *args\)](#)
, gotta know what the distance travelled is fam

7.27.1 Typedef Documentation

7.27.1.1 [uv_persistent_data_frame](#)

typedef struct uv_persistent_data_frame uv_persistent_data_frame

7.27.2 Function Documentation

7.27.2.1 initOdometer()

```
uv_status initOdometer (
    void * args)
```

Definition at line 11 of file [odometer.c](#).

References `_UV_DEFAULT_TASK_STACK_SIZE`, `uv_task_info::active_states`, `uv_task_info::deletion_states`, `odometerTask()`, `PROGRAMMING`, `uv_task_info::stack_size`, `uv_task_info::suspension_states`, `uv_task_info::task_args`, `uv_task_info::task_function`, `uv_task_info::task_name`, `uv_task_info::task_period`, `uv_task_info::task_priority`, `UV_DRIVING`, `UV_ERROR`, `UV_ERROR_STATE`, `UV_LAUNCH_CONTROL`, `UV_OK`, `UV_READY`, and `uvCreateTask()`.

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.27.2.2 odometerTask()

```
void odometerTask (
    void * args)
```

, gotta know what the distance travelled is fam

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
/*
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = xTaskGetTickCount();
/**
```

Definition at line 46 of file [odometer.c](#).

References `uv_task_info::cmd_data`, `killSelf()`, `suspendSelf()`, `uv_task_info::task_period`, `UV_KILL_CMD`, and `UV_SUSPEND_CMD`.

Referenced by [initOdometer\(\)](#).

Here is the call graph for this function:

7.28 odometer.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * odometer.h
00003  *
00004  * Created on: Nov 7, 2024
00005  *      Author: byo10
00006 */
00007
00008 #ifndef INC_ODOMETER_H_
00009 #define INC_ODOMETER_H_
00010
00011 #include"uvfr_utils.h"
00012
00013 typedef struct uv_persistent_data_frame{
00014     uint64_t total_vehicle_uptime;
00015     uint64_t total_time_driving;
00016     uint64_t total_distance_cm;
00017 }uv_persistent_data_frame;
00018
00019 uv_status initOdometer(void* args);
00020
00021 void odometerTask(void* args);
00022
00023
00024 #endif /* INC_ODOMETER_H_ */
```

7.29 Core/Inc/pdu.h File Reference

```
#include "main.h"
#include "uvfr_utils.h"
```

Include dependency graph for pdu.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [uv19_pdu_settings](#)

TypeDefs

- typedef struct abstract_conifer_channel [abstract_conifer_channel](#)
- typedef struct uv19_pdu_settings [uv19_pdu_settings](#)
- typedef enum [u19_PDU_ch](#) [u19_PDU_ch](#)

Enumerations

- enum [u19_PDU_ch](#) {
 [U19_PDU_5A_1](#) = 0x00 , [U19_PDU_5A_2](#) = 0x01 , [U19_PDU_5A_3](#) = 0x02 , [U19_PDU_5A_4](#) = 0x03 ,
 [U19_PDU_5A_5](#) = 0x07 , [U19_PDU_5A_6](#) = 0x06 , [U19_PDU_5A_7](#) = 0x05 , [U19_PDU_5A_8](#) = 0x04 ,
 [U19_PDU_5A_9](#) = 0x0B , [U19_PDU_5A_10](#) = 0x0A , [U19_PDU_5A_11](#) = 0x09 , [U19_PDU_5A_12](#) = 0x08 ,
 [U19_PDU_5A_13](#) = 0x0C , [U19_PDU_5A_14](#) = 0x0D , [U19_PDU_5A_15](#) = 0x0E , [U19_PDU_5A_16](#) = 0x0F
 ,
 [U19_PDU_20A_1](#) = 0x23 , [U19_PDU_20A_2](#) = 0x24 , [U19_PDU_20A_3](#) = 0x21 , [U19_PDU_20A_4](#) = 0x22 ,
 [U19_PDU_20A_5](#) = 0x27 , [U19_PDU_20A_6](#) = 0x28 , [U19_PDU_20A_7](#) = 0x25 , [U19_PDU_20A_8](#) = 0x26 }

Functions

- [uv_status u19updatePduChannel \(abstract_conifer_channel *ch_ptr, uint32_t *ecode\)](#)
- [uv_status initPDU \(uint32_t *ecode\)](#)

7.29.1 Typedef Documentation

7.29.1.1 abstract_conifer_channel

```
typedef struct abstract_conifer_channel abstract_conifer_channel
```

Definition at line 9 of file [pdu.h](#).

7.29.1.2 u19_PDU_ch

```
typedef enum u19_PDU_ch u19_PDU_ch
```

7.29.1.3 uv19_pdu_settings

```
typedef struct uv19_pdu_settings uv19_pdu_settings
```

7.29.2 Enumeration Type Documentation

7.29.2.1 u19_PDU_ch

```
enum u19_PDU_ch
```

Enumerator

U19_PDU_5A_1
U19_PDU_5A_2
U19_PDU_5A_3
U19_PDU_5A_4
U19_PDU_5A_5
U19_PDU_5A_6
U19_PDU_5A_7
U19_PDU_5A_8
U19_PDU_5A_9
U19_PDU_5A_10
U19_PDU_5A_11
U19_PDU_5A_12
U19_PDU_5A_13
U19_PDU_5A_14
U19_PDU_5A_15
U19_PDU_5A_16
U19_PDU_20A ₁
U19_PDU_20A ₂
U19_PDU_20A ₃
U19_PDU_20A ₄
U19_PDU_20A ₅
U19_PDU_20A ₆
U19_PDU_20A ₇
U19_PDU_20A ₈

Definition at line 22 of file [pdu.h](#).

7.29.3 Function Documentation

7.29.3.1 initPDU()

```
uv_status initPDU (
    uint32_t * ecode)
```

Definition at line 100 of file [pdu.c](#).

References [CONIFER_DRV_INIT_ERR](#), [conifer_params](#), [uv_CAN_msg::flags](#), [msg_to_PDU](#), [pdu_tx_mutex](#), [UV_ERROR](#), and [UV_OK](#).

Referenced by [coniferInit\(\)](#).

7.29.3.2 u19updatePduChannel()

```
uv_status u19updatePduChannel (
    abstract_conifer_channel * ch_ptr,
    uint32_t * ecode)
```

Definition at line 37 of file [pdu.c](#).

References CONIFER_CH_EN_BIT, CONIFER_CH_FLT_BIT, CONIFER_CH_LS_ACTIVE, CONIFER_DRV_INVALID_HW_CH_ID, uv_CAN_msg::data, FIRST_20A_CH, abstract_conifer_channel::hardware_mapping, HIGHEST_CH, msg_to_PDU, pdu_tx_mutex, set_20A_vals, set_5A_vals, abstract_conifer_channel::status_control_reg, U19_PDU_20A_BIT, U19_PDU_EN_BIT, UV_OK, and [uvSendCanMSG\(\)](#).

Referenced by [coniferInit\(\)](#).

Here is the call graph for this function:

7.30 pdu.h

[Go to the documentation of this file.](#)

```
00001 // Code to make human readable CAN messages for the device
00002
00003 // This file will define message data as human readable stuff
00004
00005
00006 #include "main.h"
00007 #include "uvfr_utils.h"
00008
00009 typedef struct abstract_conifer_channel abstract_conifer_channel;
00010
00011 // Can ID: 0x710
00012
00013 #ifndef __PDU_H__
00014 #define __PDU_H__
00015
00016 typedef struct uv19_pdu_settings{
00017     uint32_t PDU_rx_addr;
00018     uint32_t PDU_tx_addr;
00019     uint32_t expected_period;
00020 }uv19_pdu_settings;
00021
00022 typedef enum u19_PDU_ch{
00023     U19_PDU_5A_1 = 0x00,
00024     U19_PDU_5A_2 = 0x01,
00025     U19_PDU_5A_3 = 0x02,
00026     U19_PDU_5A_4 = 0x03,
00027     U19_PDU_5A_5 = 0x07,
00028     U19_PDU_5A_6 = 0x06,
00029     U19_PDU_5A_7 = 0x05,
00030     U19_PDU_5A_8 = 0x04,
00031     U19_PDU_5A_9 = 0x0B,
00032     U19_PDU_5A_10 = 0x0A,
00033     U19_PDU_5A_11 = 0x09,
00034     U19_PDU_5A_12 = 0x08,
00035     U19_PDU_5A_13 = 0x0C,
00036     U19_PDU_5A_14 = 0x0D,
00037     U19_PDU_5A_15 = 0x0E,
00038     U19_PDU_5A_16 = 0x0F,
00039     U19_PDU_20A_1 = 0x23,
00040     U19_PDU_20A_2 = 0x24,
00041     U19_PDU_20A_3 = 0x21,
00042     U19_PDU_20A_4 = 0x22,
00043     U19_PDU_20A_5 = 0x27,
00044     U19_PDU_20A_6 = 0x28,
00045     U19_PDU_20A_7 = 0x25,
00046     U19_PDU_20A_8 = 0x26
00047 }u19_PDU_ch;
00048
00049
00050 //Update a PDU channel to match it's conifer abstraction
00051 uv_status u19updatePduChannel(abstract_conifer_channel* ch_ptr, uint32_t* ecode);
00052
00053 uv_status initPDU(uint32_t* ecode);
00054
00055 #endif
```

7.31 Core/Inc/rb_tree.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- struct `rbnode`
Node of a Red-Black binary search tree.
- struct `rbtree`
struct representing a binary search tree

Macros

- `#define RB_DUP 1`
- `#define RB_MIN 1`
- `#define RED 0`
- `#define BLACK 1`
- `#define RB_ROOT(rbt)`
- `#define RB NIL(rbt)`
- `#define RB_FIRST(rbt)`
- `#define RB_MINIMAL(rbt)`
- `#define RB_ISEMPY(rbt)`
- `#define RB_APPLY(rbt, f, c, o)`

Typedefs

- `typedef struct rbnode rbnode`
Node of a Red-Black binary search tree.

Enumerations

- enum `rbtraversal { PREORDER , INORDER , POSTORDER }`
Evil traversal method specifier for traversing the tree.

Functions

- `rbtree * rbCreate (int(*compare_func)(const void *, const void *), void(*destroy_func)(void *))`
Create and initialize a binary search tree.
- `void rbDestroy (rbtree *rbt)`
Destroy the tree, and de-allocate it's elements.
- `rbnode * rbFind (rbtree *rbt, void *data)`
Find a node of the tree based off the data you provide the tree.
- `rbnode * rbSuccessor (rbtree *rbt, rbnode *node)`
- `int rbApplyNode (rbtree *rbt, rbnode *node, int(*func)(void *, void *), void *cookie, enum rbtraversal order)`
- `void rbPrint (rbtree *rbt, void(*print_func)(void *))`
Function used to print the contents of the tree.
- `rbnode * rblInsert (rbtree *rbt, void *data)`
Function that inserts data into the tree, and creates a new node.
- `void * rbDelete (rbtree *rbt, rbnode *node, int keep)`
Deletes a node from the tree.
- `int rbCheckOrder (rbtree *rbt, void *min, void *max)`
Function that validates that the order of the nodes in the tree is correct.
- `int rbCheckBlackHeight (rbtree *rbt)`
Function that Checks the height of black nodes.

7.31.1 Macro Definition Documentation

7.31.1.1 BLACK

```
#define BLACK 1
```

Definition at line 13 of file [rb_tree.h](#).

Referenced by [rbCreate\(\)](#), [rbDelete\(\)](#), and [rbInsert\(\)](#).

7.31.1.2 RB_APPLY

```
#define RB_APPLY(
    rbt,
    f,
    c,
    o)
```

Value:

```
rbapply_node((rbt), (rbt)->root.left, (f), (c), (o))
```

Definition at line 63 of file [rb_tree.h](#).

7.31.1.3 RB_DUP

```
#define RB_DUP 1
```

Definition at line 9 of file [rb_tree.h](#).

7.31.1.4 RB_FIRST

```
#define RB_FIRST(
    rbt)
```

Value:

```
((rbt)->root.left)
```

Definition at line 59 of file [rb_tree.h](#).

Referenced by [rbCheckBlackHeight\(\)](#), [rbCheckOrder\(\)](#), [rbDelete\(\)](#), [rbDestroy\(\)](#), [rbFind\(\)](#), [rbInsert\(\)](#), and [rbPrint\(\)](#).

7.31.1.5 RB_ISEMPTY

```
#define RB_ISEMPTY(
    rbt)
```

Value:

```
((rbt)->root.left == &(rbt)->nil && (rbt)->root.right == &(rbt)->nil)
```

Definition at line 62 of file [rb_tree.h](#).

7.31.1.6 RB_MIN

```
#define RB_MIN 1
```

Definition at line 10 of file [rb_tree.h](#).

7.31.1.7 RB_MINIMAL

```
#define RB_MINIMAL( rbt )
```

Value:

```
((rbt)->min)
```

Definition at line 60 of file [rb_tree.h](#).

7.31.1.8 RB NIL

```
#define RB NIL( rbt )
```

Value:

```
(&(rbt)->nill)
```

Definition at line 58 of file [rb_tree.h](#).

Referenced by [rbApplyNode\(\)](#), [rbCheckBlackHeight\(\)](#), [rbCreate\(\)](#), [rbDelete\(\)](#), [rbFind\(\)](#), [rbInsert\(\)](#), and [rbSuccessor\(\)](#).

7.31.1.9 RB_ROOT

```
#define RB_ROOT( rbt )
```

Value:

```
(&(rbt)->root)
```

Definition at line 57 of file [rb_tree.h](#).

Referenced by [rbCheckBlackHeight\(\)](#), [rbInsert\(\)](#), and [rbSuccessor\(\)](#).

7.31.1.10 RED

```
#define RED 0
```

Definition at line 12 of file [rb_tree.h](#).

Referenced by [rbCheckBlackHeight\(\)](#), [rbDelete\(\)](#), and [rbInsert\(\)](#).

7.31.2 Typedef Documentation

7.31.2.1 rbnod

```
typedef struct rbnod rbnod
```

Node of a Red-Black binary search tree.

7.31.3 Enumeration Type Documentation

7.31.3.1 rbtravers

```
enum rbtravers
```

Evil traversal method specifier for traversing the tree.

Function that applies some function to a subtree.

Parameters

<i>rbt</i>	Pointer to an the <code>rbtree</code> you wish to apply the functions to
<i>node</i>	Pointer to the node that is the root of the tree you wish to apply functions to. This can be a subtree of another tree if needed
<i>func</i>	The function you would like to apply to the nodes. This takes in two parameters. The first is a pointer to the data of the node. The second parameter is a pointer to the shared cookie, which allows information to be preserved between different calls to <code>func</code> . This function returns an integer value, with 0 being ok, and the others being various error states.
<i>cookie</i>	This is a pointer to some space in memory that all of the calls to <code>@func</code> share, to preserve information.
<i>order</i>	This is a member of that specifies the order in which the tree will be traversed.

Attention

DANGER!! RECURSION!! Beware of stack memory usage, since most tasks are memory limited.

Enumerator

PREORDER	
INORDER	
POSTORDER	

Definition at line 18 of file `rb_tree.h`.

7.31.4 Function Documentation

7.31.4.1 rbApplyNode()

```
int rbApplyNode (
    rbtree * rbt,
    rbnodes * node,
    int(* func )(void *, void *),
    void * cookie,
    enum rbtraversal order)
```

Definition at line 116 of file [rb_tree.c](#).

References [rbnode::data](#), [INORDER](#), [rbnode::left](#), [POSTORDER](#), [PREORDER](#), [RB_NIL](#), [rbApplyNode\(\)](#), and [rbnode::right](#).

Referenced by [rbApplyNode\(\)](#).

Here is the call graph for this function:

7.31.4.2 rbCheckBlackHeight()

```
int rbCheckBlackHeight (
    rbtree * rbt)
```

Function that Checks the height of black nodes.

Attention

DANGER!! THIS FUNCTION IS RECURSIVE. This is intended to be used in a laptop debugging context. The VCU simply does not have enough memory to deal with recursion of this manner.

Deprecated Leftovers from laptop unit tests.

Definition at line 571 of file [rb_tree.c](#).

References [RB_FIRST](#), [RB_NIL](#), [RB_ROOT](#), and [RED](#).

Referenced by [rbPrint\(\)](#).

7.31.4.3 rbCheckOrder()

```
int rbCheckOrder (
    rbtree * rbt,
    void * min,
    void * max)
```

Function that validates that the order of the nodes in the tree is correct.

Attention

DANGER!! THIS FUNCTION IS RECURSIVE. This is intended to be used in a laptop debugging context. The VCU simply does not have enough memory to deal with recursion of this manner.

Deprecated Leftovers from laptop unit tests.

Definition at line 545 of file [rb_tree.c](#).

References [RB_FIRST](#).

7.31.4.4 rbCreate()

```
rbtree * rbCreate (
    int(* compare_func )(const void *, const void *),
    void(* destroy_func )(void *))
```

Create and initialize a binary search tree.

Parameters

<i>compare_func</i>	A function that compares the data of two nodes. Accepts pointers to the data as parameters
<i>destroy_func</i>	The destructor function for the data, for safe disposal of dynamically allocated data

Definition at line 28 of file [rb_tree.c](#).

References [BLACK](#), [rbnode::color](#), [rbtree::compare](#), [rbtree::count](#), [rbnode::data](#), [rbtree::destroy](#), [rbnode::left](#), [rbtree::min](#), [rbtree::nil](#), [rbnode::parent](#), [RB_NIL](#), [rbnode::right](#), and [rbtree::root](#).

7.31.4.5 rbDelete()

```
void * rbDelete (
    rbtree * rbt,
    rbnode * node,
    int keep)
```

Deletes a node from the tree.

Parameters

<i>rbt</i>	Instance of a <code>rbtree</code> that we are removing the node from.
<i>node</i>	Pointer to the node that we would like to remove
<i>keep</i>	If <code>keep</code> is a truthy value, a pointer to the data of the node will be returned. Otherwise, the node and its data will be destroyed.

Return values

<i>If</i>	<code>keep</code> is "true", this will return a pointer to the data held by the deleted node. Otherwise it will return NULL.
-----------	--

Definition at line 360 of file [rb_tree.c](#).

References [BLACK](#), [rbnode::color](#), [rbtree::count](#), [rbnode::data](#), [rbtree::destroy](#), [rbnode::left](#), [rbtree::min](#), [rbnode::parent](#), [RB_FIRST](#), [RB_NIL](#), [rbSuccessor\(\)](#), [RED](#), and [rbnode::right](#).

Here is the call graph for this function:

7.31.4.6 rbDestroy()

```
void rbDestroy (
    rbtree * rbt)
```

Destroy the tree, and de-allocate its elements.

Definition at line 61 of file [rb_tree.c](#).

References [RB_FIRST](#).

7.31.4.7 rbFind()

```
rbnode * rbFind (
    rbtree * rbt,
    void * data)
```

Find a node of the tree based off the data you provide the tree.

Parameters

<i>rbt</i>	Pointer to the tree in which we would like to find the node.
------------	--

Return values

<i>Returns</i>	a pointer to the node if the node is present in the tree. Otherwise, it will return NULL to indicate the node could not be found.
----------------	---

Definition at line 71 of file [rb_tree.c](#).

References [rbtree::compare](#), [rbnode::data](#), [rbnode::left](#), [RB_FIRST](#), [RB NIL](#), and [rbnode::right](#).

7.31.4.8 rbInsert()

```
rbnode * rbInsert (
    rbtree * rbt,
    void * data)
```

Function that inserts data into the tree, and creates a new node.

Parameters

<i>rbt</i>	Instance of a <code>rbtree</code> that we would like to insert data into
<i>data</i>	Pointer to the data we wish to insert

Return values

<i>This</i>	function returns a pointer to the <code>rbnode</code> that was added. The function will return NULL if the system is out of memory, or is otherwise unable to insert the node.
-------------	--

Definition at line 207 of file [rb_tree.c](#).

References [BLACK](#), [rbnode::color](#), [rbtree::compare](#), [rbtree::count](#), [rbnode::data](#), [rbtree::destroy](#), [rbnode::left](#), [rbtree::min](#), [rbnode::parent](#), [RB_FIRST](#), [RB NIL](#), [RB_ROOT](#), [RED](#), and [rbnode::right](#).

7.31.4.9 rbPrint()

```
void rbPrint (
    rbtree * rbt,
    void(* print_func )(void *))
```

Function used to print the contents of the tree.

Parameters

<i>rbt</i>	a pointer to the <code>rbtree</code> you wish to print.
<i>print_func</i>	A pointer to a print function specific to the data.

Attention

DANGER!! RECURSION!!

Deprecated Leftovers from laptop unit tests. Sorta useless, cause like what are we gonna print to?

Definition at line 607 of file [rb_tree.c](#).

References [RB_FIRST](#), and [rbCheckBlackHeight\(\)](#).

Here is the call graph for this function:

7.31.4.10 rbSuccessor()

```
rbnode * rbSuccessor (
    rbtree * rbt,
    rbnod
```

Definition at line 92 of file [rb_tree.c](#).

References [rbnode::left](#), [rbnode::parent](#), [RB NIL](#), [RB ROOT](#), and [rbnode::right](#).

Referenced by [rbDelete\(\)](#).

7.32 rb_tree.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * Copyright (c) 2019 xieqing. https://github.com/xieqing
00003  * May be freely redistributed, but copyright notice must be retained.
00004 */
00005
00006 #ifndef _RB_HEADER
00007 #define _RB_HEADER
00008
00009 #define RB_DUP 1
00010 #define RB_MIN 1
00011
00012 #define RED 0
00013 #define BLACK 1
00014
00018 enum rbtraversal {
00019     PREORDER,
00020     INORDER,
00021     POSTORDER
00022 };
00023
00027 typedef struct rbnod
00028 {
00029     struct rbnod *left;
00030     struct rbnod *right;
00031     struct rbnod *parent;
00032     void *data;
00033     char color;
00034 } rbnod;
00035
00039 typedef struct {
```

```

00040     int (*compare)(const void *, const void *);
00041     void (*print)(void *);
00042     void (*destroy)(void *);
00044     rbnodes root;
00045     rbnodes nil;
00049     #ifndef RB_MIN
00050     rbnodes *min;
00051     #endif
00052
00053     int count;
00054 } rbtree;
00055
00056 //Internal macros
00057 #define RB_ROOT(rbt) (&(rbt)->root)
00058 #define RB NIL(rbt) (&(rbt)->nil)
00059 #define RB FIRST(rbt) ((rbt)->root.left)
00060 #define RB_MINIMAL(rbt) ((rbt)->min)
00061
00062 #define RB_ISEMPTY(rbt) ((rbt)->root.left == &(rbt)->nil && (rbt)->root.right == &(rbt)->nil)
00063 #define RB_APPLY(rbt, f, c, o) rbapply_node((rbt), (rbt)->root.left, (f), (c), (o))
00064
00071 rbtree *rbCreate(int (*compare_func)(const void *, const void *), void (*destroy_func)(void *));
00072
00076 void rbDestroy(rbtree *rbt);
00077
00086 rbnodes *rbFind(rbtree *rbt, void *data);
00087 rbnodes *rbSuccessor(rbtree *rbt, rbnodes *node);
00088
00105 int rbApplyNode(rbtree *rbt, rbnodes *node, int (*func)(void *, void *), void *cookie, enum rbtraversal
order);
00106
00117 void rbPrint(rbtree *rbt, void (*print_func)(void *));
00118
00128 rbnodes *rbInsert(rbtree *rbt, void *data);
00129
00143 void *rbDelete(rbtree *rbt, rbnodes *node, int keep);
00144
00153 int rbCheckOrder(rbtree *rbt, void *min, void *max);
00154
00163 int rbCheckBlackHeight(rbtree *rbt);
00164
00165 #endif /* _RB_HEADER */

```

7.33 Core/Inc/stm32f4xx_hal_conf.h File Reference

HAL configuration template file. This file should be copied to the application folder and renamed to [stm32f4xx_hal_conf.h](#).

```
#include "stm32f4xx_hal_rcc.h"
#include "stm32f4xx_hal_gpio.h"
#include "stm32f4xx_hal_exti.h"
#include "stm32f4xx_hal_dma.h"
#include "stm32f4xx_hal_cortex.h"
#include "stm32f4xx_hal_adc.h"
#include "stm32f4xx_hal_can.h"
#include "stm32f4xx_hal_flash.h"
#include "stm32f4xx_hal_pwr.h"
#include "stm32f4xx_hal_tim.h"
```

Include dependency graph for `stm32f4xx_hal_conf.h`: This graph shows which files directly or indirectly include this file:

Macros

- `#define HAL_MODULE_ENABLED`
- This is the list of modules to be used in the HAL driver.*
- `#define HAL_ADC_MODULE_ENABLED`
 - `#define HAL_CAN_MODULE_ENABLED`
 - `#define HAL_TIM_MODULE_ENABLED`

- #define HAL_GPIO_MODULE_ENABLED
- #define HAL_EXTI_MODULE_ENABLED
- #define HAL_DMA_MODULE_ENABLED
- #define HAL_RCC_MODULE_ENABLED
- #define HAL_FLASH_MODULE_ENABLED
- #define HAL_PWR_MODULE_ENABLED
- #define HAL_CORTEX_MODULE_ENABLED
- #define HSE_VALUE 25000000U

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

- #define HSE_STARTUP_TIMEOUT 100U
- #define HSI_VALUE ((uint32_t)16000000U)

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

- #define LSI_VALUE 32000U

Internal Low Speed oscillator (LSI) value.

- #define LSE_VALUE 32768U

External Low Speed oscillator (LSE) value.

- #define LSE_STARTUP_TIMEOUT 5000U
- #define EXTERNAL_CLOCK_VALUE 12288000U

External clock source for I2S peripheral This value is used by the I2S HAL module to compute the I2S clock source frequency, this source is inserted directly through I2S_CKIN pad.

- #define VDD_VALUE 3300U

This is the HAL system configuration section.

- #define TICK_INT_PRIORITY 15U
- #define USERTOS 0U
- #define PREFETCH_ENABLE 1U
- #define INSTRUCTION_CACHE_ENABLE 1U
- #define DATA_CACHE_ENABLE 1U
- #define USE_HAL_ADC_REGISTER_CALLBACKS 0U /* ADC register callback disabled */
- #define USE_HAL_CAN_REGISTER_CALLBACKS 0U /* CAN register callback disabled */
- #define USE_HAL_CEC_REGISTER_CALLBACKS 0U /* CEC register callback disabled */
- #define USE_HAL_CRYP_REGISTER_CALLBACKS 0U /* CRYP register callback disabled */
- #define USE_HAL_DAC_REGISTER_CALLBACKS 0U /* DAC register callback disabled */
- #define USE_HAL_DCMI_REGISTER_CALLBACKS 0U /* DCMI register callback disabled */
- #define USE_HAL_DFSDM_REGISTER_CALLBACKS 0U /* DFSDM register callback disabled */
- #define USE_HAL_DMA2D_REGISTER_CALLBACKS 0U /* DMA2D register callback disabled */
- #define USE_HAL_DSI_REGISTER_CALLBACKS 0U /* DSI register callback disabled */
- #define USE_HAL_ETH_REGISTER_CALLBACKS 0U /* ETH register callback disabled */
- #define USE_HAL_HASH_REGISTER_CALLBACKS 0U /* HASH register callback disabled */
- #define USE_HAL_HCD_REGISTER_CALLBACKS 0U /* HCD register callback disabled */
- #define USE_HAL_I2C_REGISTER_CALLBACKS 0U /* I2C register callback disabled */
- #define USE_HAL_FMPI2C_REGISTER_CALLBACKS 0U /* FMPI2C register callback disabled */
- #define USE_HAL_FMP SMBUS_REGISTER_CALLBACKS 0U /* FMP SMBUS register callback disabled */
- #define USE_HAL_I2S_REGISTER_CALLBACKS 0U /* I2S register callback disabled */
- #define USE_HAL_IRDA_REGISTER_CALLBACKS 0U /* IRDA register callback disabled */
- #define USE_HAL_LPTIM_REGISTER_CALLBACKS 0U /* LPTIM register callback disabled */
- #define USE_HAL_LTDC_REGISTER_CALLBACKS 0U /* LTDC register callback disabled */
- #define USE_HAL_MMC_REGISTER_CALLBACKS 0U /* MMC register callback disabled */
- #define USE_HAL_NAND_REGISTER_CALLBACKS 0U /* NAND register callback disabled */
- #define USE_HAL_NOR_REGISTER_CALLBACKS 0U /* NOR register callback disabled */
- #define USE_HAL_PCCARD_REGISTER_CALLBACKS 0U /* PCCARD register callback disabled */
- #define USE_HAL_PCD_REGISTER_CALLBACKS 0U /* PCD register callback disabled */
- #define USE_HAL_QSPI_REGISTER_CALLBACKS 0U /* QSPI register callback disabled */

- #define USE_HAL_RNG_REGISTER_CALLBACKS 0U /* RNG register callback disabled */
- #define USE_HAL_RTC_REGISTER_CALLBACKS 0U /* RTC register callback disabled */
- #define USE_HAL_SAI_REGISTER_CALLBACKS 0U /* SAI register callback disabled */
- #define USE_HAL_SD_REGISTER_CALLBACKS 0U /* SD register callback disabled */
- #define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U /* SMARTCARD register callback disabled */
- #define USE_HAL_SDRAM_REGISTER_CALLBACKS 0U /* SDRAM register callback disabled */
- #define USE_HAL_SRAM_REGISTER_CALLBACKS 0U /* SRAM register callback disabled */
- #define USE_HAL_SPDIFRX_REGISTER_CALLBACKS 0U /* SPDIFRX register callback disabled */
- #define USE_HAL_SMBUS_REGISTER_CALLBACKS 0U /* SMBUS register callback disabled */
- #define USE_HAL_SPI_REGISTER_CALLBACKS 0U /* SPI register callback disabled */
- #define USE_HAL_TIM_REGISTER_CALLBACKS 0U /* TIM register callback disabled */
- #define USE_HAL_UART_REGISTER_CALLBACKS 0U /* UART register callback disabled */
- #define USE_HAL_USART_REGISTER_CALLBACKS 0U /* USART register callback disabled */
- #define USE_HAL_WWDG_REGISTER_CALLBACKS 0U /* WWDG register callback disabled */
- #define MAC_ADDR0 2U

Uncomment the line below to expand the "assert_param" macro in the HAL drivers code.

- #define MAC_ADDR1 0U
- #define MAC_ADDR2 0U
- #define MAC_ADDR3 0U
- #define MAC_ADDR4 0U
- #define MAC_ADDR5 0U
- #define ETH_RX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for receive */
- #define ETH_TX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for transmit */
- #define ETH_RXBUFN 4U /* 4 Rx buffers of size ETH_RX_BUF_SIZE */
- #define ETH_TXBUFN 4U /* 4 Tx buffers of size ETH_TX_BUF_SIZE */
- #define DP83848_PHY_ADDRESS
- #define PHY_RESET_DELAY 0x000000FFU
- #define PHY_CONFIG_DELAY 0x00000FFFU
- #define PHY_READ_TO 0x0000FFFFU
- #define PHY_WRITE_TO 0x0000FFFFU
- #define PHY_BCR ((uint16_t)0x0000U)
- #define PHY_BSR ((uint16_t)0x0001U)
- #define PHY_RESET ((uint16_t)0x8000U)
- #define PHY_LOOPBACK ((uint16_t)0x4000U)
- #define PHY_FULLDUPLEX_100M ((uint16_t)0x2100U)
- #define PHY_HALFDUPLEX_100M ((uint16_t)0x2000U)
- #define PHY_FULLDUPLEX_10M ((uint16_t)0x0100U)
- #define PHY_HALFDUPLEX_10M ((uint16_t)0x0000U)
- #define PHY_AUTONEGOTIATION ((uint16_t)0x1000U)
- #define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200U)
- #define PHY_POWERDOWN ((uint16_t)0x0800U)
- #define PHY_ISOLATE ((uint16_t)0x0400U)
- #define PHY_AUTONEGO_COMPLETE ((uint16_t)0x0020U)
- #define PHY_LINKED_STATUS ((uint16_t)0x0004U)
- #define PHY_JABBER_DETECTION ((uint16_t)0x0002U)
- #define PHY_SR ((uint16_t))
- #define PHY_SPEED_STATUS ((uint16_t))
- #define PHY_DUPLEX_STATUS ((uint16_t))
- #define USE_SPI_CRC 0U
- #define assert_param(expr)

Include module's header file.

7.33.1 Detailed Description

HAL configuration template file. This file should be copied to the application folder and renamed to `stm32f4xx_hal_conf.h`.

Author

MCD Application Team

Attention

Copyright (c) 2017 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [stm32f4xx_hal_conf.h](#).

7.33.2 Macro Definition Documentation

7.33.2.1 assert_param

```
#define assert_param(  
    expr)
```

Value:

```
((void) 0U)
```

Include module's header file.

Definition at line [488](#) of file [stm32f4xx_hal_conf.h](#).

7.33.2.2 DATA_CACHE_ENABLE

```
#define DATA_CACHE_ENABLE 1U
```

Definition at line [155](#) of file [stm32f4xx_hal_conf.h](#).

7.33.2.3 DP83848_PHY_ADDRESS

```
#define DP83848_PHY_ADDRESS
```

Definition at line [225](#) of file [stm32f4xx_hal_conf.h](#).

7.33.2.4 ETH_RX_BUF_SIZE

```
#define ETH_RX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for receive */
```

Definition at line [217](#) of file [stm32f4xx_hal_conf.h](#).

7.33.2.5 ETH_RXBUFN

```
#define ETH_RXBUFN 4U /* 4 Rx buffers of size ETH_RX_BUF_SIZE */
```

Definition at line 219 of file [stm32f4xx_hal_conf.h](#).

7.33.2.6 ETH_TX_BUF_SIZE

```
#define ETH_TX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for transmit */
```

Definition at line 218 of file [stm32f4xx_hal_conf.h](#).

7.33.2.7 ETH_TXBUFN

```
#define ETH_TXBUFN 4U /* 4 Tx buffers of size ETH_TX_BUF_SIZE */
```

Definition at line 220 of file [stm32f4xx_hal_conf.h](#).

7.33.2.8 EXTERNAL_CLOCK_VALUE

```
#define EXTERNAL_CLOCK_VALUE 12288000U
```

External clock source for I2S peripheral This value is used by the I2S HAL module to compute the I2S clock source frequency, this source is inserted directly through I2S_CKIN pad.

Value of the External audio frequency in Hz

Definition at line 140 of file [stm32f4xx_hal_conf.h](#).

7.33.2.9 HAL_ADC_MODULE_ENABLED

```
#define HAL_ADC_MODULE_ENABLED
```

Definition at line 41 of file [stm32f4xx_hal_conf.h](#).

7.33.2.10 HAL_CAN_MODULE_ENABLED

```
#define HAL_CAN_MODULE_ENABLED
```

Definition at line 42 of file [stm32f4xx_hal_conf.h](#).

7.33.2.11 HAL_CORTEX_MODULE_ENABLED

```
#define HAL_CORTEX_MODULE_ENABLED
```

Definition at line 90 of file [stm32f4xx_hal_conf.h](#).

7.33.2.12 HAL_DMA_MODULE_ENABLED

```
#define HAL_DMA_MODULE_ENABLED
```

Definition at line 86 of file [stm32f4xx_hal_conf.h](#).

7.33.2.13 HAL_EXTI_MODULE_ENABLED

```
#define HAL_EXTI_MODULE_ENABLED
```

Definition at line 85 of file [stm32f4xx_hal_conf.h](#).

7.33.2.14 HAL_FLASH_MODULE_ENABLED

```
#define HAL_FLASH_MODULE_ENABLED
```

Definition at line 88 of file [stm32f4xx_hal_conf.h](#).

7.33.2.15 HAL_GPIO_MODULE_ENABLED

```
#define HAL_GPIO_MODULE_ENABLED
```

Definition at line 84 of file [stm32f4xx_hal_conf.h](#).

7.33.2.16 HAL_MODULE_ENABLED

```
#define HAL_MODULE_ENABLED
```

This is the list of modules to be used in the HAL driver.

Definition at line 38 of file [stm32f4xx_hal_conf.h](#).

7.33.2.17 HAL_PWR_MODULE_ENABLED

```
#define HAL_PWR_MODULE_ENABLED
```

Definition at line 89 of file [stm32f4xx_hal_conf.h](#).

7.33.2.18 HAL_RCC_MODULE_ENABLED

```
#define HAL_RCC_MODULE_ENABLED
```

Definition at line 87 of file [stm32f4xx_hal_conf.h](#).

7.33.2.19 HAL_TIM_MODULE_ENABLED

```
#define HAL_TIM_MODULE_ENABLED
```

Definition at line 66 of file [stm32f4xx_hal_conf.h](#).

7.33.2.20 HSE_STARTUP_TIMEOUT

```
#define HSE_STARTUP_TIMEOUT 100U
```

Time out for HSE start up, in ms

Definition at line 103 of file [stm32f4xx_hal_conf.h](#).

7.33.2.21 HSE_VALUE

```
#define HSE_VALUE 25000000U
```

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

Value of the External oscillator in Hz

Definition at line 99 of file [stm32f4xx_hal_conf.h](#).

7.33.2.22 HSI_VALUE

```
#define HSI_VALUE ((uint32_t)16000000U)
```

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

Value of the Internal oscillator in Hz

Definition at line 112 of file [stm32f4xx_hal_conf.h](#).

7.33.2.23 INSTRUCTION_CACHE_ENABLE

```
#define INSTRUCTION_CACHE_ENABLE 1U
```

Definition at line 154 of file [stm32f4xx_hal_conf.h](#).

7.33.2.24 LSE_STARTUP_TIMEOUT

```
#define LSE_STARTUP_TIMEOUT 5000U
```

Time out for LSE start up, in ms

Definition at line 131 of file [stm32f4xx_hal_conf.h](#).

7.33.2.25 LSE_VALUE

```
#define LSE_VALUE 32768U
```

External Low Speed oscillator (LSE) value.

< Value of the Internal Low Speed oscillator in Hz The real value may vary depending on the variations in voltage and temperature. Value of the External Low Speed oscillator in Hz

Definition at line 127 of file [stm32f4xx_hal_conf.h](#).

7.33.2.26 LSI_VALUE

```
#define LSI_VALUE 32000U
```

Internal Low Speed oscillator (LSI) value.

LSI Typical Value in Hz

Definition at line 119 of file [stm32f4xx_hal_conf.h](#).

7.33.2.27 MAC_ADDR0

```
#define MAC_ADDR0 2U
```

Uncomment the line below to expand the "assert_param" macro in the HAL drivers code.

Definition at line 209 of file [stm32f4xx_hal_conf.h](#).

7.33.2.28 MAC_ADDR1

```
#define MAC_ADDR1 0U
```

Definition at line 210 of file [stm32f4xx_hal_conf.h](#).

7.33.2.29 MAC_ADDR2

```
#define MAC_ADDR2 0U
```

Definition at line 211 of file [stm32f4xx_hal_conf.h](#).

7.33.2.30 MAC_ADDR3

```
#define MAC_ADDR3 0U
```

Definition at line 212 of file [stm32f4xx_hal_conf.h](#).

7.33.2.31 MAC_ADDR4

```
#define MAC_ADDR4 0U
```

Definition at line 213 of file [stm32f4xx_hal_conf.h](#).

7.33.2.32 MAC_ADDR5

```
#define MAC_ADDR5 0U
```

Definition at line 214 of file [stm32f4xx_hal_conf.h](#).

7.33.2.33 PHY_AUTONEGO_COMPLETE

```
#define PHY_AUTONEGO_COMPLETE ((uint16_t)0x0020U)
```

Auto-Negotiation process completed

Definition at line 250 of file [stm32f4xx_hal_conf.h](#).

7.33.2.34 PHY_AUTONEGOTIATION

```
#define PHY_AUTONEGOTIATION ((uint16_t)0x1000U)
```

Enable auto-negotiation function

Definition at line 245 of file [stm32f4xx_hal_conf.h](#).

7.33.2.35 PHY_BCR

```
#define PHY_BCR ((uint16_t)0x0000U)
```

Transceiver Basic Control Register

Definition at line 236 of file [stm32f4xx_hal_conf.h](#).

7.33.2.36 PHY_BSR

```
#define PHY_BSR ((uint16_t)0x0001U)
```

Transceiver Basic Status Register

Definition at line 237 of file [stm32f4xx_hal_conf.h](#).

7.33.2.37 PHY_CONFIG_DELAY

```
#define PHY_CONFIG_DELAY 0x00000FFFU
```

Definition at line 229 of file [stm32f4xx_hal_conf.h](#).

7.33.2.38 PHY_DUPLEX_STATUS

```
#define PHY_DUPLEX_STATUS ((uint16_t))
```

PHY Duplex mask

Definition at line 258 of file [stm32f4xx_hal_conf.h](#).

7.33.2.39 PHY_FULLDUPLEX_100M

```
#define PHY_FULLDUPLEX_100M ((uint16_t)0x2100U)
```

Set the full-duplex mode at 100 Mb/s

Definition at line 241 of file [stm32f4xx_hal_conf.h](#).

7.33.2.40 PHY_FULLDUPLEX_10M

```
#define PHY_FULLDUPLEX_10M ((uint16_t)0x0100U)
```

Set the full-duplex mode at 10 Mb/s

Definition at line 243 of file [stm32f4xx_hal_conf.h](#).

7.33.2.41 PHY_HALFDUPLEX_100M

```
#define PHY_HALFDUPLEX_100M ((uint16_t)0x2000U)
```

Set the half-duplex mode at 100 Mb/s

Definition at line 242 of file [stm32f4xx_hal_conf.h](#).

7.33.2.42 PHY_HALFDUPLEX_10M

```
#define PHY_HALFDUPLEX_10M ((uint16_t)0x0000U)
```

Set the half-duplex mode at 10 Mb/s

Definition at line 244 of file [stm32f4xx_hal_conf.h](#).

7.33.2.43 PHY_ISOLATE

```
#define PHY_ISOLATE ((uint16_t)0x0400U)
```

Isolate PHY from MII

Definition at line 248 of file [stm32f4xx_hal_conf.h](#).

7.33.2.44 PHY_JABBER_DETECTION

```
#define PHY_JABBER_DETECTION ((uint16_t)0x0002U)
```

Jabber condition detected

Definition at line 252 of file [stm32f4xx_hal_conf.h](#).

7.33.2.45 PHY_LINKED_STATUS

```
#define PHY_LINKED_STATUS ((uint16_t)0x0004U)
```

Valid link established

Definition at line 251 of file [stm32f4xx_hal_conf.h](#).

7.33.2.46 PHY_LOOPBACK

```
#define PHY_LOOPBACK ((uint16_t)0x4000U)
```

Select loop-back mode

Definition at line 240 of file [stm32f4xx_hal_conf.h](#).

7.33.2.47 PHY_POWERDOWN

```
#define PHY_POWERDOWN ((uint16_t)0x0800U)
```

Select the power down mode

Definition at line 247 of file [stm32f4xx_hal_conf.h](#).

7.33.2.48 PHY_READ_TO

```
#define PHY_READ_TO 0x0000FFFFU
```

Definition at line 231 of file [stm32f4xx_hal_conf.h](#).

7.33.2.49 PHY_RESET

```
#define PHY_RESET ((uint16_t)0x8000U)
```

PHY Reset

Definition at line 239 of file [stm32f4xx_hal_conf.h](#).

7.33.2.50 PHY_RESET_DELAY

```
#define PHY_RESET_DELAY 0x000000FFU
```

Definition at line 227 of file [stm32f4xx_hal_conf.h](#).

7.33.2.51 PHY_RESTART_AUTONEGOTIATION

```
#define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200U)
```

Restart auto-negotiation function

Definition at line 246 of file [stm32f4xx_hal_conf.h](#).

7.33.2.52 PHY_SPEED_STATUS

```
#define PHY_SPEED_STATUS ((uint16_t))
```

PHY Speed mask

Definition at line 257 of file [stm32f4xx_hal_conf.h](#).

7.33.2.53 PHY_SR

```
#define PHY_SR ((uint16_t))
```

PHY status register Offset

Definition at line 255 of file [stm32f4xx_hal_conf.h](#).

7.33.2.54 PHY_WRITE_TO

```
#define PHY_WRITE_TO 0x0000FFFFU
```

Definition at line 232 of file [stm32f4xx_hal_conf.h](#).

7.33.2.55 PREFETCH_ENABLE

```
#define PREFETCH_ENABLE 1U
```

Definition at line 153 of file [stm32f4xx_hal_conf.h](#).

7.33.2.56 TICK_INT_PRIORITY

```
#define TICK_INT_PRIORITY 15U
```

tick interrupt priority

Definition at line 151 of file [stm32f4xx_hal_conf.h](#).

7.33.2.57 USE_HAL_ADC_REGISTER_CALLBACKS

```
#define USE_HAL_ADC_REGISTER_CALLBACKS 0U /* ADC register callback disabled */
```

Definition at line 157 of file [stm32f4xx_hal_conf.h](#).

7.33.2.58 USE_HAL_CAN_REGISTER_CALLBACKS

```
#define USE_HAL_CAN_REGISTER_CALLBACKS 0U /* CAN register callback disabled */
```

Definition at line 158 of file [stm32f4xx_hal_conf.h](#).

7.33.2.59 USE_HAL_CEC_REGISTER_CALLBACKS

```
#define USE_HAL_CEC_REGISTER_CALLBACKS 0U /* CEC register callback disabled */
```

Definition at line 159 of file [stm32f4xx_hal_conf.h](#).

7.33.2.60 USE_HAL_CRYP_REGISTER_CALLBACKS

```
#define USE_HAL_CRYP_REGISTER_CALLBACKS 0U /* CRYP register callback disabled */
```

Definition at line 160 of file [stm32f4xx_hal_conf.h](#).

7.33.2.61 USE_HAL_DAC_REGISTER_CALLBACKS

```
#define USE_HAL_DAC_REGISTER_CALLBACKS 0U /* DAC register callback disabled */
```

Definition at line 161 of file [stm32f4xx_hal_conf.h](#).

7.33.2.62 USE_HAL_DCMI_REGISTER_CALLBACKS

```
#define USE_HAL_DCMI_REGISTER_CALLBACKS 0U /* DCMI register callback disabled */
```

Definition at line 162 of file [stm32f4xx_hal_conf.h](#).

7.33.2.63 USE_HAL_DFSDM_REGISTER_CALLBACKS

```
#define USE_HAL_DFSDM_REGISTER_CALLBACKS 0U /* DFSDM register callback disabled */
```

Definition at line 163 of file [stm32f4xx_hal_conf.h](#).

7.33.2.64 USE_HAL_DMA2D_REGISTER_CALLBACKS

```
#define USE_HAL_DMA2D_REGISTER_CALLBACKS 0U /* DMA2D register callback disabled */
```

Definition at line 164 of file [stm32f4xx_hal_conf.h](#).

7.33.2.65 USE_HAL_DSI_REGISTER_CALLBACKS

```
#define USE_HAL_DSI_REGISTER_CALLBACKS 0U /* DSI register callback disabled */
```

Definition at line 165 of file [stm32f4xx_hal_conf.h](#).

7.33.2.66 USE_HAL_ETH_REGISTER_CALLBACKS

```
#define USE_HAL_ETH_REGISTER_CALLBACKS 0U /* ETH register callback disabled */
```

Definition at line 166 of file [stm32f4xx_hal_conf.h](#).

7.33.2.67 USE_HAL_FMPI2C_REGISTER_CALLBACKS

```
#define USE_HAL_FMPI2C_REGISTER_CALLBACKS 0U /* FMPI2C register callback disabled */
```

Definition at line 170 of file [stm32f4xx_hal_conf.h](#).

7.33.2.68 USE_HAL_FMP SMBUS REGISTER_CALLBACKS

```
#define USE_HAL_FMP SMBUS REGISTER_CALLBACKS 0U /* FMP SMBUS register callback disabled */
```

Definition at line 171 of file [stm32f4xx_hal_conf.h](#).

7.33.2.69 USE_HAL_HASH_REGISTER_CALLBACKS

```
#define USE_HAL_HASH_REGISTER_CALLBACKS 0U /* HASH register callback disabled */
```

Definition at line 167 of file [stm32f4xx_hal_conf.h](#).

7.33.2.70 USE_HAL_HCD_REGISTER_CALLBACKS

```
#define USE_HAL_HCD_REGISTER_CALLBACKS 0U /* HCD register callback disabled */
```

Definition at line 168 of file [stm32f4xx_hal_conf.h](#).

7.33.2.71 USE_HAL_I2C_REGISTER_CALLBACKS

```
#define USE_HAL_I2C_REGISTER_CALLBACKS 0U /* I2C register callback disabled */
```

Definition at line 169 of file [stm32f4xx_hal_conf.h](#).

7.33.2.72 USE_HAL_I2S_REGISTER_CALLBACKS

```
#define USE_HAL_I2S_REGISTER_CALLBACKS 0U /* I2S register callback disabled */
```

Definition at line 172 of file [stm32f4xx_hal_conf.h](#).

7.33.2.73 USE_HAL_IRDA_REGISTER_CALLBACKS

```
#define USE_HAL_IRDA_REGISTER_CALLBACKS 0U /* IRDA register callback disabled */
```

Definition at line 173 of file [stm32f4xx_hal_conf.h](#).

7.33.2.74 USE_HAL_LPTIM_REGISTER_CALLBACKS

```
#define USE_HAL_LPTIM_REGISTER_CALLBACKS 0U /* LPTIM register callback disabled */
```

Definition at line 174 of file [stm32f4xx_hal_conf.h](#).

7.33.2.75 USE_HAL_LTDC_REGISTER_CALLBACKS

```
#define USE_HAL_LTDC_REGISTER_CALLBACKS 0U /* LTDC register callback disabled */
```

Definition at line 175 of file [stm32f4xx_hal_conf.h](#).

7.33.2.76 USE_HAL_MMC_REGISTER_CALLBACKS

```
#define USE_HAL_MMC_REGISTER_CALLBACKS 0U /* MMC register callback disabled */
```

Definition at line 176 of file [stm32f4xx_hal_conf.h](#).

7.33.2.77 USE_HAL_NAND_REGISTER_CALLBACKS

```
#define USE_HAL_NAND_REGISTER_CALLBACKS 0U /* NAND register callback disabled */
```

Definition at line 177 of file [stm32f4xx_hal_conf.h](#).

7.33.2.78 USE_HAL_NOR_REGISTER_CALLBACKS

```
#define USE_HAL_NOR_REGISTER_CALLBACKS 0U /* NOR register callback disabled */
```

Definition at line 178 of file [stm32f4xx_hal_conf.h](#).

7.33.2.79 USE_HAL_PCCARD_REGISTER_CALLBACKS

```
#define USE_HAL_PCCARD_REGISTER_CALLBACKS 0U /* PCCARD register callback disabled */
```

Definition at line 179 of file [stm32f4xx_hal_conf.h](#).

7.33.2.80 USE_HAL_PCD_REGISTER_CALLBACKS

```
#define USE_HAL_PCD_REGISTER_CALLBACKS 0U /* PCD register callback disabled */
```

Definition at line 180 of file [stm32f4xx_hal_conf.h](#).

7.33.2.81 USE_HAL_QSPI_REGISTER_CALLBACKS

```
#define USE_HAL_QSPI_REGISTER_CALLBACKS 0U /* QSPI register callback disabled */
```

Definition at line 181 of file [stm32f4xx_hal_conf.h](#).

7.33.2.82 USE_HAL_RNG_REGISTER_CALLBACKS

```
#define USE_HAL_RNG_REGISTER_CALLBACKS 0U /* RNG register callback disabled */
```

Definition at line 182 of file [stm32f4xx_hal_conf.h](#).

7.33.2.83 USE_HAL_RTC_REGISTER_CALLBACKS

```
#define USE_HAL_RTC_REGISTER_CALLBACKS 0U /* RTC register callback disabled */
```

Definition at line 183 of file [stm32f4xx_hal_conf.h](#).

7.33.2.84 USE_HAL_SAI_REGISTER_CALLBACKS

```
#define USE_HAL_SAI_REGISTER_CALLBACKS 0U /* SAI register callback disabled */
```

Definition at line 184 of file [stm32f4xx_hal_conf.h](#).

7.33.2.85 USE_HAL_SD_REGISTER_CALLBACKS

```
#define USE_HAL_SD_REGISTER_CALLBACKS 0U /* SD register callback disabled */
```

Definition at line 185 of file [stm32f4xx_hal_conf.h](#).

7.33.2.86 USE_HAL_SDRAM_REGISTER_CALLBACKS

```
#define USE_HAL_SDRAM_REGISTER_CALLBACKS 0U /* SDRAM register callback disabled */
```

Definition at line 187 of file [stm32f4xx_hal_conf.h](#).

7.33.2.87 USE_HAL_SMARTCARD_REGISTER_CALLBACKS

```
#define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U /* SMARTCARD register callback disabled */
```

Definition at line 186 of file [stm32f4xx_hal_conf.h](#).

7.33.2.88 USE_HAL_SMBUS_REGISTER_CALLBACKS

```
#define USE_HAL_SMBUS_REGISTER_CALLBACKS 0U /* SMBUS register callback disabled */
```

Definition at line 190 of file [stm32f4xx_hal_conf.h](#).

7.33.2.89 USE_HAL_SPDIFRX_REGISTER_CALLBACKS

```
#define USE_HAL_SPDIFRX_REGISTER_CALLBACKS 0U /* SPDIFRX register callback disabled */
```

Definition at line 189 of file [stm32f4xx_hal_conf.h](#).

7.33.2.90 USE_HAL_SPI_REGISTER_CALLBACKS

```
#define USE_HAL_SPI_REGISTER_CALLBACKS 0U /* SPI register callback disabled */
```

Definition at line 191 of file [stm32f4xx_hal_conf.h](#).

7.33.2.91 USE_HAL_SRAM_REGISTER_CALLBACKS

```
#define USE_HAL_SRAM_REGISTER_CALLBACKS 0U /* SRAM register callback disabled */
```

Definition at line 188 of file [stm32f4xx_hal_conf.h](#).

7.33.2.92 USE_HAL_TIM_REGISTER_CALLBACKS

```
#define USE_HAL_TIM_REGISTER_CALLBACKS 0U /* TIM register callback disabled */
```

Definition at line 192 of file [stm32f4xx_hal_conf.h](#).

7.33.2.93 USE_HAL_UART_REGISTER_CALLBACKS

```
#define USE_HAL_UART_REGISTER_CALLBACKS 0U /* UART register callback disabled */
```

Definition at line 193 of file [stm32f4xx_hal_conf.h](#).

7.33.2.94 USE_HAL_USART_REGISTER_CALLBACKS

```
#define USE_HAL_USART_REGISTER_CALLBACKS 0U /* USART register callback disabled */
```

Definition at line 194 of file [stm32f4xx_hal_conf.h](#).

7.33.2.95 USE_HAL_WWDG_REGISTER_CALLBACKS

```
#define USE_HAL_WWDG_REGISTER_CALLBACKS 0U /* WWDG register callback disabled */
```

Definition at line 195 of file [stm32f4xx_hal_conf.h](#).

7.33.2.96 USE_RTOS

```
#define USE_RTOS 0U
```

Definition at line 152 of file [stm32f4xx_hal_conf.h](#).

7.33.2.97 USE_SPI_CRC

```
#define USE_SPI_CRC 0U
```

Definition at line 267 of file [stm32f4xx_hal_conf.h](#).

7.33.2.98 VDD_VALUE

```
#define VDD_VALUE 3300U
```

This is the HAL system configuration section.

Value of VDD in mv

Definition at line 150 of file [stm32f4xx_hal_conf.h](#).

7.34 stm32f4xx_hal_conf.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00021 /* USER CODE END Header */
00022
00023 /* Define to prevent recursive inclusion -----*/
00024 #ifndef __STM32F4XX_HAL_CONF_H
00025 #define __STM32F4XX_HAL_CONF_H
00026
00027 #ifdef __cplusplus
00028 extern "C" {
00029 #endif
00030
00031 /* Exported types -----*/
00032 /* Exported constants -----*/
00033
00034 /* ##### Module Selection ##### */
00038 #define HAL_MODULE_ENABLED
00039
00040 /* #define HAL_CRYPT_MODULE_ENABLED */
00041 #define HAL_ADC_MODULE_ENABLED
00042 #define HAL_CAN_MODULE_ENABLED
00043 /* #define HAL_CRC_MODULE_ENABLED */
00044 /* #define HAL_CAN_LEGACY_MODULE_ENABLED */
00045 /* #define HAL_DAC_MODULE_ENABLED */
00046 /* #define HAL_DCMI_MODULE_ENABLED */
00047 /* #define HAL_DMA2D_MODULE_ENABLED */
00048 /* #define HAL_ETH_MODULE_ENABLED */
00049 /* #define HAL_ETH_LEGACY_MODULE_ENABLED */
00050 /* #define HAL_NAND_MODULE_ENABLED */
00051 /* #define HAL_NOR_MODULE_ENABLED */
00052 /* #define HAL_PCCARD_MODULE_ENABLED */
00053 /* #define HAL_SRAM_MODULE_ENABLED */
00054 /* #define HAL_SDRAM_MODULE_ENABLED */
00055 /* #define HAL_HASH_MODULE_ENABLED */
00056 /* #define HAL_I2C_MODULE_ENABLED */
00057 /* #define HAL_I2S_MODULE_ENABLED */
00058 /* #define HAL_IWDG_MODULE_ENABLED */
00059 /* #define HAL_LTDC_MODULE_ENABLED */
00060 /* #define HAL_RNG_MODULE_ENABLED */
00061 /* #define HAL_RTC_MODULE_ENABLED */
00062 /* #define HAL_SAI_MODULE_ENABLED */
00063 /* #define HAL_SD_MODULE_ENABLED */
00064 /* #define HAL_MMC_MODULE_ENABLED */
00065 /* #define HAL_SPI_MODULE_ENABLED */
00066 #define HAL_TIM_MODULE_ENABLED
00067 /* #define HAL_UART_MODULE_ENABLED */
00068 /* #define HAL_USART_MODULE_ENABLED */
00069 /* #define HAL_IRDA_MODULE_ENABLED */
00070 /* #define HAL_SMARTCARD_MODULE_ENABLED */
00071 /* #define HAL_SMBUS_MODULE_ENABLED */
00072 /* #define HAL_WWDG_MODULE_ENABLED */
00073 /* #define HAL_PCD_MODULE_ENABLED */
00074 /* #define HAL_HCD_MODULE_ENABLED */
00075 /* #define HAL_DSI_MODULE_ENABLED */
00076 /* #define HAL_QSPI_MODULE_ENABLED */
00077 /* #define HAL_QSPI_MODULE_ENABLED */
00078 /* #define HAL_CEC_MODULE_ENABLED */
00079 /* #define HAL_FMPI2C_MODULE_ENABLED */
00080 /* #define HAL_FMPFSMBUS_MODULE_ENABLED */
00081 /* #define HAL_SPDIFRX_MODULE_ENABLED */
00082 /* #define HAL_DFSDM_MODULE_ENABLED */
00083 /* #define HAL_LPTIM_MODULE_ENABLED */
00084 #define HAL_GPIO_MODULE_ENABLED
00085 #define HAL_EXTI_MODULE_ENABLED
00086 #define HAL_DMA_MODULE_ENABLED
00087 #define HAL_RCC_MODULE_ENABLED
00088 #define HAL_FLASH_MODULE_ENABLED
00089 #define HAL_PWR_MODULE_ENABLED
00090 #define HAL_CORTEX_MODULE_ENABLED
00091
00092 /* ##### HSE/HSI Values adaptation ##### */
00098 #if !defined (HSE_VALUE)
00099     #define HSE_VALUE    25000000U
00100 #endif /* HSE_VALUE */
00101
00102 #if !defined (HSE_STARTUP_TIMEOUT)
00103     #define HSE_STARTUP_TIMEOUT    100U
00104 #endif /* HSE_STARTUP_TIMEOUT */
00105
00111 #if !defined (HSI_VALUE)
00112     #define HSI_VALUE    ((uint32_t)16000000U)
00113 #endif /* HSI_VALUE */
00114

```

```

00118 #if !defined (LSI_VALUE)
00119 #define LSI_VALUE 32000U
00120 #endif /* LSI_VALUE */
00126 #if !defined (LSE_VALUE)
00127 #define LSE_VALUE 32768U
00128 #endif /* LSE_VALUE */
00129
00130 #if !defined (LSE_STARTUP_TIMEOUT)
00131 #define LSE_STARTUP_TIMEOUT 5000U
00132 #endif /* LSE_STARTUP_TIMEOUT */
00133
00139 #if !defined (EXTERNAL_CLOCK_VALUE)
00140 #define EXTERNAL_CLOCK_VALUE 12288000U
00141 #endif /* EXTERNAL_CLOCK_VALUE */
00142
00143 /* Tip: To avoid modifying this file each time you need to use different HSE,
00144 === you can define the HSE value in your toolchain compiler preprocessor. */
00145
00146 /* ##### System Configuration #####
00150 #define VDD_VALUE 3300U
00151 #define TICK_INT_PRIORITY 15U
00152 #define USERTOS 0U
00153 #define PREFETCH_ENABLE 1U
00154 #define INSTRUCTION_CACHE_ENABLE 1U
00155 #define DATA_CACHE_ENABLE 1U
00156
00157 #define USE_HAL_ADC_REGISTER_CALLBACKS 0U /* ADC register callback disabled */
00158 #define USE_HAL_CAN_REGISTER_CALLBACKS 0U /* CAN register callback disabled */
00159 #define USE_HAL_CEC_REGISTER_CALLBACKS 0U /* CEC register callback disabled */
00160 #define USE_HAL_CRYPT_REGISTER_CALLBACKS 0U /* CRYPT register callback disabled */
00161 #define USE_HAL_DAC_REGISTER_CALLBACKS 0U /* DAC register callback disabled */
00162 #define USE_HAL_DCMI_REGISTER_CALLBACKS 0U /* DCMI register callback disabled */
00163 #define USE_HAL_DFSDM_REGISTER_CALLBACKS 0U /* DFSDM register callback disabled */
00164 #define USE_HAL_DMA2D_REGISTER_CALLBACKS 0U /* DMA2D register callback disabled */
00165 #define USE_HAL_DSI_REGISTER_CALLBACKS 0U /* DSI register callback disabled */
00166 #define USE_HAL_ETH_REGISTER_CALLBACKS 0U /* ETH register callback disabled */
00167 #define USE_HAL_HASH_REGISTER_CALLBACKS 0U /* HASH register callback disabled */
00168 #define USE_HAL_HCD_REGISTER_CALLBACKS 0U /* HCD register callback disabled */
00169 #define USE_HAL_I2C_REGISTER_CALLBACKS 0U /* I2C register callback disabled */
00170 #define USE_HAL_FMPI2C_REGISTER_CALLBACKS 0U /* FMPPI2C register callback disabled */
00171 #define USE_HAL_FMPSMBUS_REGISTER_CALLBACKS 0U /* FMPSMBUS register callback disabled */
00172 #define USE_HAL_I2S_REGISTER_CALLBACKS 0U /* I2S register callback disabled */
00173 #define USE_HAL_IRDA_REGISTER_CALLBACKS 0U /* IRDA register callback disabled */
00174 #define USE_HAL_LPTIM_REGISTER_CALLBACKS 0U /* LPTIM register callback disabled */
00175 #define USE_HAL_LTDC_REGISTER_CALLBACKS 0U /* LTDC register callback disabled */
00176 #define USE_HAL_MMCHS_REGISTER_CALLBACKS 0U /* MMC register callback disabled */
00177 #define USE_HAL_NAND_REGISTER_CALLBACKS 0U /* NAND register callback disabled */
00178 #define USE_HAL_NOR_REGISTER_CALLBACKS 0U /* NOR register callback disabled */
00179 #define USE_HAL_PCCARD_REGISTER_CALLBACKS 0U /* PCCARD register callback disabled */
00180 #define USE_HAL_PCD_REGISTER_CALLBACKS 0U /* PCD register callback disabled */
00181 #define USE_HAL_QSPI_REGISTER_CALLBACKS 0U /* QSPI register callback disabled */
00182 #define USE_HAL_RNG_REGISTER_CALLBACKS 0U /* RNG register callback disabled */
00183 #define USE_HAL_RTC_REGISTER_CALLBACKS 0U /* RTC register callback disabled */
00184 #define USE_HAL_SAI_REGISTER_CALLBACKS 0U /* SAI register callback disabled */
00185 #define USE_HAL_SD_REGISTER_CALLBACKS 0U /* SD register callback disabled */
00186 #define USE_HAL_SMARTCARD_REGISTER_CALLBACKS 0U /* SMARTCARD register callback disabled */
00187 #define USE_HAL_SDRAM_REGISTER_CALLBACKS 0U /* SDRAM register callback disabled */
00188 #define USE_HAL_SRAM_REGISTER_CALLBACKS 0U /* SRAM register callback disabled */
00189 #define USE_HAL_SPDIFRX_REGISTER_CALLBACKS 0U /* SPDIFRX register callback disabled */
00190 #define USE_HAL_SMBUS_REGISTER_CALLBACKS 0U /* SMBUS register callback disabled */
00191 #define USE_HAL_SPI_REGISTER_CALLBACKS 0U /* SPI register callback disabled */
00192 #define USE_HAL_TIM_REGISTER_CALLBACKS 0U /* TIM register callback disabled */
00193 #define USE_HAL_UART_REGISTER_CALLBACKS 0U /* UART register callback disabled */
00194 #define USE_HAL_USART_REGISTER_CALLBACKS 0U /* USART register callback disabled */
00195 #define USE_HAL_WWDG_REGISTER_CALLBACKS 0U /* WWDG register callback disabled */
00196
00197 /* ##### Assert Selection #####
00202 /* #define USE_FULL_ASSERT 1U */
00203
00204 /* ##### Ethernet peripheral configuration #####
00205
00206 /* Section 1 : Ethernet peripheral configuration */
00207
00208 /* MAC ADDRESS: MAC_ADDR0:MAC_ADDR1:MAC_ADDR2:MAC_ADDR3:MAC_ADDR4:MAC_ADDR5 */
00209 #define MAC_ADDR0 2U
00210 #define MAC_ADDR1 0U
00211 #define MAC_ADDR2 0U
00212 #define MAC_ADDR3 0U
00213 #define MAC_ADDR4 0U
00214 #define MAC_ADDR5 0U
00215
00216 /* Definition of the Ethernet driver buffers size and count */
00217 #define ETH_RX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for receive */
00218 #define ETH_TX_BUF_SIZE ETH_MAX_PACKET_SIZE /* buffer size for transmit */
00219 #define ETH_RXBUFN 4U /* 4 Rx buffers of size ETH_RX_BUF_SIZE */
00220 #define ETH_TXBUFN 4U /* 4 Tx buffers of size ETH_TX_BUF_SIZE */
00221

```

```
00222 /* Section 2: PHY configuration section */
00223
00224 /* DP83848_PHY_ADDRESS Address*/
00225 #define DP83848_PHY_ADDRESS
00226 /* PHY Reset delay these values are based on a 1 ms Systick interrupt*/
00227 #define PHY_RESET_DELAY          0x000000FFU
00228 /* PHY Configuration delay */
00229 #define PHY_CONFIG_DELAY         0x00000FFFU
00230
00231 #define PHY_READ_TO              0x0000FFFFU
00232 #define PHY_WRITE_TO             0x00000FFFFU
00233
00234 /* Section 3: Common PHY Registers */
00235
00236 #define PHY_BCR                  ((uint16_t)0x0000U)
00237 #define PHY_BSR                  ((uint16_t)0x0001U)
00238 #define PHY_RESET                ((uint16_t)0x8000U)
00239
00240 #define PHY_LOOPBACK             ((uint16_t)0x4000U)
00241 #define PHY_FULLDUPLEX_100M      ((uint16_t)0x2100U)
00242 #define PHY_HALFDUPLEX_100M      ((uint16_t)0x2000U)
00243 #define PHY_FULLDUPLEX_10M       ((uint16_t)0x0100U)
00244 #define PHY_HALFDUPLEX_10M       ((uint16_t)0x0000U)
00245 #define PHY_AUTONEGOTIATION     ((uint16_t)0x1000U)
00246 #define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200U)
00247 #define PHY_POWERDOWN            ((uint16_t)0x0800U)
00248 #define PHY_ISOLATE              ((uint16_t)0x0400U)
00249 #define PHY_AUTONEGO_COMPLETE    ((uint16_t)0x0020U)
00250 #define PHY_LINKED_STATUS        ((uint16_t)0x0004U)
00251 #define PHY_JABBER_DETECTION     ((uint16_t)0x0002U)
00252
00253 /* Section 4: Extended PHY Registers */
00254
00255 #define PHY_SR                  ((uint16_t))
00256 #define PHY_SPEED_STATUS          ((uint16_t))
00257 #define PHY_DUPLEX_STATUS         ((uint16_t))
00258
00259 /* ##### SPI peripheral configuration ##### */
00260
00261
00262 /* CRC FEATURE: Use to activate CRC feature inside HAL SPI Driver
00263 * Activated: CRC code is present inside driver
00264 * Deactivated: CRC code cleaned from driver
00265 */
00266
00267 #define USE_SPI_CRC             0U
00268
00269 /* Includes ----- */
00270
00271 #ifdef HAL_RCC_MODULE_ENABLED
00272     #include "stm32f4xx_hal_rcc.h"
00273 #endif /* HAL_RCC_MODULE_ENABLED */
00274
00275 #ifdef HAL_GPIO_MODULE_ENABLED
00276     #include "stm32f4xx_hal_gpio.h"
00277 #endif /* HAL_GPIO_MODULE_ENABLED */
00278
00279 #ifdef HAL_EXTI_MODULE_ENABLED
00280     #include "stm32f4xx_hal_exti.h"
00281 #endif /* HAL_EXTI_MODULE_ENABLED */
00282
00283 #ifdef HAL_DMA_MODULE_ENABLED
00284     #include "stm32f4xx_hal_dma.h"
00285 #endif /* HAL_DMA_MODULE_ENABLED */
00286
00287 #ifdef HAL_CORTEX_MODULE_ENABLED
00288     #include "stm32f4xx_hal_cortex.h"
00289 #endif /* HAL_CORTEX_MODULE_ENABLED */
00290
00291 #ifdef HAL_ADC_MODULE_ENABLED
00292     #include "stm32f4xx_hal_adc.h"
00293 #endif /* HAL_ADC_MODULE_ENABLED */
00294
00295 #ifdef HAL_CAN_MODULE_ENABLED
00296     #include "stm32f4xx_hal_can.h"
00297 #endif /* HAL_CAN_MODULE_ENABLED */
00298
00299 #ifdef HAL_CAN_LEGACY_MODULE_ENABLED
00300     #include "stm32f4xx_hal_can_legacy.h"
00301 #endif /* HAL_CAN_LEGACY_MODULE_ENABLED */
00302
00303 #ifdef HAL_CRC_MODULE_ENABLED
00304     #include "stm32f4xx_hal_crc.h"
00305 #endif /* HAL_CRC_MODULE_ENABLED */
00306
00307 #ifdef HAL_CRYPT_MODULE_ENABLED
00308     #include "stm32f4xx_hal_crypt.h"
00309 #endif /* HAL_CRYPT_MODULE_ENABLED */
00310
00311 #ifdef HAL_DMA2D_MODULE_ENABLED
00312     #include "stm32f4xx_hal_dma2d.h"
00313 #endif /* HAL_DMA2D_MODULE_ENABLED */
00314
00315 #ifdef HAL_DMA2D_MODULE_ENABLED
00316     #include "stm32f4xx_hal_dma2d.h"
00317 #endif /* HAL_DMA2D_MODULE_ENABLED */
```

```
00318 #ifdef HAL_DAC_MODULE_ENABLED
00319     #include "stm32f4xx_hal_dac.h"
00320 #endif /* HAL_DAC_MODULE_ENABLED */
00321
00322 #ifdef HAL_DCMI_MODULE_ENABLED
00323     #include "stm32f4xx_hal_dcmi.h"
00324 #endif /* HAL_DCMI_MODULE_ENABLED */
00325
00326 #ifdef HAL_ETH_MODULE_ENABLED
00327     #include "stm32f4xx_hal_eth.h"
00328 #endif /* HAL_ETH_MODULE_ENABLED */
00329
00330 #ifdef HAL_ETH_LEGACY_MODULE_ENABLED
00331     #include "stm32f4xx_hal_eth_legacy.h"
00332 #endif /* HAL_ETH_LEGACY_MODULE_ENABLED */
00333
00334 #ifdef HAL_FLASH_MODULE_ENABLED
00335     #include "stm32f4xx_hal_flash.h"
00336 #endif /* HAL_FLASH_MODULE_ENABLED */
00337
00338 #ifdef HAL_SRAM_MODULE_ENABLED
00339     #include "stm32f4xx_hal_sram.h"
00340 #endif /* HAL_SRAM_MODULE_ENABLED */
00341
00342 #ifdef HAL_NOR_MODULE_ENABLED
00343     #include "stm32f4xx_hal_nor.h"
00344 #endif /* HAL_NOR_MODULE_ENABLED */
00345
00346 #ifdef HAL_NAND_MODULE_ENABLED
00347     #include "stm32f4xx_hal_nand.h"
00348 #endif /* HAL_NAND_MODULE_ENABLED */
00349
00350 #ifdef HAL_PCCARD_MODULE_ENABLED
00351     #include "stm32f4xx_hal_pccard.h"
00352 #endif /* HAL_PCCARD_MODULE_ENABLED */
00353
00354 #ifdef HAL_SDRAM_MODULE_ENABLED
00355     #include "stm32f4xx_hal_sdram.h"
00356 #endif /* HAL_SDRAM_MODULE_ENABLED */
00357
00358 #ifdef HAL_HASH_MODULE_ENABLED
00359     #include "stm32f4xx_hal_hash.h"
00360 #endif /* HAL_HASH_MODULE_ENABLED */
00361
00362 #ifdef HAL_I2C_MODULE_ENABLED
00363     #include "stm32f4xx_hal_i2c.h"
00364 #endif /* HAL_I2C_MODULE_ENABLED */
00365
00366 #ifdef HAL_SMBUS_MODULE_ENABLED
00367     #include "stm32f4xx_hal_smbus.h"
00368 #endif /* HAL_SMBUS_MODULE_ENABLED */
00369
00370 #ifdef HAL_I2S_MODULE_ENABLED
00371     #include "stm32f4xx_hal_i2s.h"
00372 #endif /* HAL_I2S_MODULE_ENABLED */
00373
00374 #ifdef HAL_IWDG_MODULE_ENABLED
00375     #include "stm32f4xx_hal_iwdg.h"
00376 #endif /* HAL_IWDG_MODULE_ENABLED */
00377
00378 #ifdef HAL_LTDC_MODULE_ENABLED
00379     #include "stm32f4xx_hal_ltdc.h"
00380 #endif /* HAL_LTDC_MODULE_ENABLED */
00381
00382 #ifdef HAL_PWR_MODULE_ENABLED
00383     #include "stm32f4xx_hal_pwr.h"
00384 #endif /* HAL_PWR_MODULE_ENABLED */
00385
00386 #ifdef HAL_RNG_MODULE_ENABLED
00387     #include "stm32f4xx_hal_rng.h"
00388 #endif /* HAL_RNG_MODULE_ENABLED */
00389
00390 #ifdef HAL_RTC_MODULE_ENABLED
00391     #include "stm32f4xx_hal_rtc.h"
00392 #endif /* HAL_RTC_MODULE_ENABLED */
00393
00394 #ifdef HAL_SAI_MODULE_ENABLED
00395     #include "stm32f4xx_hal_sai.h"
00396 #endif /* HAL_SAI_MODULE_ENABLED */
00397
00398 #ifdef HAL_SD_MODULE_ENABLED
00399     #include "stm32f4xx_hal_sd.h"
00400 #endif /* HAL_SD_MODULE_ENABLED */
00401
00402 #ifdef HAL_SPI_MODULE_ENABLED
00403     #include "stm32f4xx_hal_spi.h"
00404 #endif /* HAL_SPI_MODULE_ENABLED */
```

```
00405
00406 #ifdef HAL_TIM_MODULE_ENABLED
00407 #include "stm32f4xx_hal_tim.h"
00408 #endif /* HAL_TIM_MODULE_ENABLED */
00409
00410 #ifdef HAL_UART_MODULE_ENABLED
00411 #include "stm32f4xx_hal_uart.h"
00412 #endif /* HAL_UART_MODULE_ENABLED */
00413
00414 #ifdef HAL_USART_MODULE_ENABLED
00415 #include "stm32f4xx_hal_usart.h"
00416 #endif /* HAL_USART_MODULE_ENABLED */
00417
00418 #ifdef HAL_IRDA_MODULE_ENABLED
00419 #include "stm32f4xx_hal_irda.h"
00420 #endif /* HAL_IRDA_MODULE_ENABLED */
00421
00422 #ifdef HAL_SMARTCARD_MODULE_ENABLED
00423 #include "stm32f4xx_hal_smartcard.h"
00424 #endif /* HAL_SMARTCARD_MODULE_ENABLED */
00425
00426 #ifdef HAL_WWDG_MODULE_ENABLED
00427 #include "stm32f4xx_hal_wwdg.h"
00428 #endif /* HAL_WWDG_MODULE_ENABLED */
00429
00430 #ifdef HAL_PCD_MODULE_ENABLED
00431 #include "stm32f4xx_hal_pcd.h"
00432 #endif /* HAL_PCD_MODULE_ENABLED */
00433
00434 #ifdef HAL_HCD_MODULE_ENABLED
00435 #include "stm32f4xx_hal_hcd.h"
00436 #endif /* HAL_HCD_MODULE_ENABLED */
00437
00438 #ifdef HAL_DSI_MODULE_ENABLED
00439 #include "stm32f4xx_hal_dsi.h"
00440 #endif /* HAL_DSI_MODULE_ENABLED */
00441
00442 #ifdef HAL_QSPI_MODULE_ENABLED
00443 #include "stm32f4xx_hal_qspi.h"
00444 #endif /* HAL_QSPI_MODULE_ENABLED */
00445
00446 #ifdef HAL_CEC_MODULE_ENABLED
00447 #include "stm32f4xx_hal_cec.h"
00448 #endif /* HAL_CEC_MODULE_ENABLED */
00449
00450 #ifdef HAL_FMPI2C_MODULE_ENABLED
00451 #include "stm32f4xx_hal_fmpi2c.h"
00452 #endif /* HAL_FMPI2C_MODULE_ENABLED */
00453
00454 #ifdef HAL_FMPSEMBUS_MODULE_ENABLED
00455 #include "stm32f4xx_hal_fmpsembus.h"
00456 #endif /* HAL_FMPSEMBUS_MODULE_ENABLED */
00457
00458 #ifdef HAL_SPDIFRX_MODULE_ENABLED
00459 #include "stm32f4xx_hal_spdifrx.h"
00460 #endif /* HAL_SPDIFRX_MODULE_ENABLED */
00461
00462 #ifdef HAL_DFSDM_MODULE_ENABLED
00463 #include "stm32f4xx_hal_dfsm.h"
00464 #endif /* HAL_DFSDM_MODULE_ENABLED */
00465
00466 #ifdef HAL_LPTIM_MODULE_ENABLED
00467 #include "stm32f4xx_hal_lptim.h"
00468 #endif /* HAL_LPTIM_MODULE_ENABLED */
00469
00470 #ifdef HAL_MMC_MODULE_ENABLED
00471 #include "stm32f4xx_hal_mmc.h"
00472 #endif /* HAL_MMC_MODULE_ENABLED */
00473
00474 /* Exported macro -----*/
00475 #ifdef USE_FULL_ASSERT
00476 #define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FILE__, __LINE__))
00477 /* Exported functions -----*/
00478 void assert_failed(uint8_t* file, uint32_t line);
00479 #else
00480 #define assert_param(expr) ((void)0U)
00481 #endif /* USE_FULL_ASSERT */
00482
00483 #ifdef __cplusplus
00484 }
00485 #endif
00486
00487 #endif /* __STM32F4xx_HAL_CONF_H */
```

7.35 Core\Inc\stm32f4xx_it.h File Reference

This file contains the headers of the interrupt handlers.

This graph shows which files directly or indirectly include this file:

Functions

- void [NMI_Handler](#) (void)
This function handles Non maskable interrupt.
- void [HardFault_Handler](#) (void)
This function handles Hard fault interrupt.
- void [MemManage_Handler](#) (void)
This function handles Memory management fault.
- void [BusFault_Handler](#) (void)
This function handles Pre-fetch fault, memory access fault.
- void [UsageFault_Handler](#) (void)
This function handles Undefined instruction or illegal state.
- void [SVC_Handler](#) (void)
This function handles System service call via SWI instruction.
- void [DebugMon_Handler](#) (void)
This function handles Debug monitor.
- void [PendSV_Handler](#) (void)
This function handles Pendable request for system service.
- void [SysTick_Handler](#) (void)
This function handles System tick timer.
- void [ADC_IRQHandler](#) (void)
This function handles ADC1, ADC2 and ADC3 global interrupts.
- void [CAN1_TX_IRQHandler](#) (void)
This function handles CAN1 TX interrupts.
- void [CAN1_RX0_IRQHandler](#) (void)
This function handles CAN1 RX0 interrupts.
- void [CAN1_RX1_IRQHandler](#) (void)
This function handles CAN1 RX1 interrupt.
- void [CAN1_SCE_IRQHandler](#) (void)
This function handles CAN1 SCE interrupt.
- void [TIM1_UP_TIM10_IRQHandler](#) (void)
This function handles TIM1 update interrupt and TIM10 global interrupt.
- void [EXTI15_10_IRQHandler](#) (void)
This function handles EXTI line[15:10] interrupts.
- void [DMA2_Stream0_IRQHandler](#) (void)
This function handles DMA2 stream0 global interrupt.
- void [DMA2_Stream1_IRQHandler](#) (void)
This function handles DMA2 stream1 global interrupt.
- void [DMA2_Stream2_IRQHandler](#) (void)
This function handles DMA2 stream2 global interrupt.
- void [CAN2_TX_IRQHandler](#) (void)
This function handles CAN2 TX interrupts.
- void [CAN2_RX0_IRQHandler](#) (void)
This function handles CAN2 RX0 interrupts.
- void [CAN2_RX1_IRQHandler](#) (void)
This function handles CAN2 RX1 interrupt.
- void [CAN2_SCE_IRQHandler](#) (void)
This function handles CAN2 SCE interrupt.

7.35.1 Detailed Description

This file contains the headers of the interrupt handlers.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [stm32f4xx_it.h](#).

7.35.2 Function Documentation

7.35.2.1 ADC_IRQHandler()

```
void ADC_IRQHandler (
    void )
```

This function handles ADC1, ADC2 and ADC3 global interrupts.

Definition at line [213](#) of file [stm32f4xx_it.c](#).

References [hadc1](#), [hadc2](#), and [hadc3](#).

7.35.2.2 BusFault_Handler()

```
void BusFault_Handler (
    void )
```

This function handles Pre-fetch fault, memory access fault.

Definition at line [123](#) of file [stm32f4xx_it.c](#).

7.35.2.3 CAN1_RX0_IRQHandler()

```
void CAN1_RX0_IRQHandler (
    void )
```

This function handles CAN1 RX0 interrupts.

Definition at line [243](#) of file [stm32f4xx_it.c](#).

References [hcan1](#).

7.35.2.4 CAN1_RX1_IRQHandler()

```
void CAN1_RX1_IRQHandler (
    void )
```

This function handles CAN1 RX1 interrupt.

Definition at line 257 of file [stm32f4xx_it.c](#).

References [hcan1](#).

7.35.2.5 CAN1_SCE_IRQHandler()

```
void CAN1_SCE_IRQHandler (
    void )
```

This function handles CAN1 SCE interrupt.

Definition at line 271 of file [stm32f4xx_it.c](#).

References [hcan1](#).

7.35.2.6 CAN1_TX_IRQHandler()

```
void CAN1_TX_IRQHandler (
    void )
```

This function handles CAN1 TX interrupts.

Definition at line 229 of file [stm32f4xx_it.c](#).

References [hcan1](#).

7.35.2.7 CAN2_RX0_IRQHandler()

```
void CAN2_RX0_IRQHandler (
    void )
```

This function handles CAN2 RX0 interrupts.

Definition at line 372 of file [stm32f4xx_it.c](#).

References [hcan2](#).

7.35.2.8 CAN2_RX1_IRQHandler()

```
void CAN2_RX1_IRQHandler (
    void )
```

This function handles CAN2 RX1 interrupt.

Definition at line 386 of file [stm32f4xx_it.c](#).

References [hcan2](#).

7.35.2.9 CAN2_SCE_IRQHandler()

```
void CAN2_SCE_IRQHandler (
    void )
```

This function handles CAN2 SCE interrupt.

Definition at line 400 of file [stm32f4xx_it.c](#).

References [hcan2](#).

7.35.2.10 CAN2_TX_IRQHandler()

```
void CAN2_TX_IRQHandler (
    void )
```

This function handles CAN2 TX interrupts.

Definition at line 358 of file [stm32f4xx_it.c](#).

References [hcan2](#).

7.35.2.11 DebugMon_Handler()

```
void DebugMon_Handler (
    void )
```

This function handles Debug monitor.

Definition at line 166 of file [stm32f4xx_it.c](#).

7.35.2.12 DMA2_Stream0_IRQHandler()

```
void DMA2_Stream0_IRQHandler (
    void )
```

This function handles DMA2 stream0 global interrupt.

Definition at line 316 of file [stm32f4xx_it.c](#).

References [hdma_adc1](#).

7.35.2.13 DMA2_Stream1_IRQHandler()

```
void DMA2_Stream1_IRQHandler (
    void )
```

This function handles DMA2 stream1 global interrupt.

Definition at line 330 of file [stm32f4xx_it.c](#).

References [hdma_adc3](#).

7.35.2.14 DMA2_Stream2_IRQHandler()

```
void DMA2_Stream2_IRQHandler (
    void )
```

This function handles DMA2 stream2 global interrupt.

Definition at line [344](#) of file [stm32f4xx_it.c](#).

References [hdma_adc2](#).

7.35.2.15 EXTI15_10_IRQHandler()

```
void EXTI15_10_IRQHandler (
    void )
```

This function handles EXTI line[15:10] interrupts.

Definition at line [299](#) of file [stm32f4xx_it.c](#).

7.35.2.16 HardFault_Handler()

```
void HardFault_Handler (
    void )
```

This function handles Hard fault interrupt.

Definition at line [93](#) of file [stm32f4xx_it.c](#).

7.35.2.17 MemManage_Handler()

```
void MemManage_Handler (
    void )
```

This function handles Memory management fault.

Definition at line [108](#) of file [stm32f4xx_it.c](#).

7.35.2.18 NMI_Handler()

```
void NMI_Handler (
    void )
```

This function handles Non maskable interrupt.

Definition at line [78](#) of file [stm32f4xx_it.c](#).

7.35.2.19 PendSV_Handler()

```
void PendSV_Handler (
    void )
```

This function handles Pendable request for system service.

Definition at line 179 of file [stm32f4xx_it.c](#).

7.35.2.20 SVC_Handler()

```
void SVC_Handler (
    void )
```

This function handles System service call via SWI instruction.

Definition at line 153 of file [stm32f4xx_it.c](#).

7.35.2.21 SysTick_Handler()

```
void SysTick_Handler (
    void )
```

This function handles System tick timer.

Definition at line 192 of file [stm32f4xx_it.c](#).

7.35.2.22 TIM1_UP_TIM10_IRQHandler()

```
void TIM1_UP_TIM10_IRQHandler (
    void )
```

This function handles TIM1 update interrupt and TIM10 global interrupt.

Definition at line 285 of file [stm32f4xx_it.c](#).

References [htim1](#).

7.35.2.23 UsageFault_Handler()

```
void UsageFault_Handler (
    void )
```

This function handles Undefined instruction or illegal state.

Definition at line 138 of file [stm32f4xx_it.c](#).

7.36 stm32f4xx_it.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Define to prevent recursive inclusion -----*/
00021 #ifndef __STM32F4XX_IT_H
00022 #define __STM32F4XX_IT_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Private includes -----*/
00029 /* USER CODE BEGIN Includes */
00030
00031 /* USER CODE END Includes */
00032
00033 /* Exported types -----*/
00034 /* USER CODE BEGIN ET */
00035
00036 /* USER CODE END ET */
00037
00038 /* Exported constants -----*/
00039 /* USER CODE BEGIN EC */
00040
00041 /* USER CODE END EC */
00042
00043 /* Exported macro -----*/
00044 /* USER CODE BEGIN EM */
00045
00046 /* USER CODE END EM */
00047
00048 /* Exported functions prototypes -----*/
00049 void NMI_Handler(void);
00050 void HardFault_Handler(void);
00051 void MemManage_Handler(void);
00052 void BusFault_Handler(void);
00053 void UsageFault_Handler(void);
00054 void SVC_Handler(void);
00055 void DebugMon_Handler(void);
00056 void PendSV_Handler(void);
00057 void SysTick_Handler(void);
00058 void ADC_IRQHandler(void);
00059 void CAN1_TX_IRQHandler(void);
00060 void CAN1_RX0_IRQHandler(void);
00061 void CAN1_RX1_IRQHandler(void);
00062 void CAN1_SCE_IRQHandler(void);
00063 void TIM1_UP_TIM10_IRQHandler(void);
00064 void EXTI15_10_IRQHandler(void);
00065 void DMA2_Stream0_IRQHandler(void);
00066 void DMA2_Stream1_IRQHandler(void);
00067 void DMA2_Stream2_IRQHandler(void);
00068 void CAN2_TX_IRQHandler(void);
00069 void CAN2_RX0_IRQHandler(void);
00070 void CAN2_RX1_IRQHandler(void);
00071 void CAN2_SCE_IRQHandler(void);
00072 /* USER CODE BEGIN EFP */
00073
00074 /* USER CODE END EFP */
00075
00076 #ifdef __cplusplus
00077 }
00078 #endif
00079
00080 #endif /* __STM32F4XX_IT_H */

```

7.37 Core/Inc/temp_monitoring.h File Reference

```
#include "uvfr_utils.h"
```

Include dependency graph for temp_monitoring.h: This graph shows which files directly or indirectly include this file:

Functions

- [uv_status initTempMonitor \(void *args\)](#)
- [void tempMonitorTask \(void *args\)](#)

7.37.1 Function Documentation

7.37.1.1 initTempMonitor()

```
uv_status initTempMonitor (
    void * args)
```

Definition at line 17 of file [temp_monitoring.c](#).

References `_UV_DEFAULT_TASK_STACK_SIZE`, `uv_task_info::active_states`, `uv_task_info::deletion_states`, `PROGRAMMING`, `uv_task_info::stack_size`, `uv_task_info::suspension_states`, `uv_task_info::task_args`, `uv_task_info::task_function`, `uv_task_info::task_name`, `uv_task_info::task_period`, `uv_task_info::task_priority`, `tempMonitorTask()`, `UV_DRIVING`, `UV_ERROR`, `UV_ERROR_STATE`, `UV_LAUNCH_CONTROL`, `UV_OK`, `UV_READY`, and `uvCreateTask()`.

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.37.1.2 tempMonitorTask()

```
void tempMonitorTask (
    void * args)
```

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
*/
```

```
insertCANMessageHandler(0x64, testfunc2, CAN_BUS_2);
//insertCANMessageHandler(0x6B1, BMS_msg2, CAN_BUS_2);
```

```
uv_CAN_msg temp1;
uv_CAN_msg temp2;

temp1.data[0] = 1;
temp1.data[1] = 2;
temp1.data[2] = 3;
temp1.dlc = 3;
temp1.msg_id = 0x85;
temp1.flags = CAN_BUS_1;

temp2.data[0] = 9;
temp2.data[1] = 10;
temp2.data[2] = 11;
temp2.dlc = 3;
temp2.msg_id = 0xF5;
temp2.flags = 0x00;
```

```
uvSendCanMSG(&temp1);
uvSendCanMSG(&temp2);
```

```
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = 0;
/**
```

This is an example of a task control point, which is the spot in the task where the task decides what needs to be done, based on the commands it has received from the task manager and the SCD

Definition at line 112 of file [temp_monitoring.c](#).

References `CAN_BUS_1`, `CAN_BUS_2`, `uv_task_info::cmd_data`, `uv_CAN_msg::data`, `uv_CAN_msg::dlc`, `uv_CAN_msg::flags`, `insertCANMessageHandler()`, `killSelf()`, `uv_CAN_msg::msg_id`, `suspendSelf()`, `uv_task_info::task_period`, `testfunc()`, `testfunc2()`, `testfunc3()`, `UV_KILL_CMD`, `UV_SUSPEND_CMD`, `uvSendCanMSG()`, and `uvTaskDelayUntil()`.

Referenced by [initTempMonitor\(\)](#).

Here is the call graph for this function:

7.38 temp_monitoring.h

[Go to the documentation of this file.](#)

```
00001 /*  
00002 * temp_monitoring.h  
00003 *  
00004 * Created on: Oct 31, 2024  
00005 * Author: byo10  
00006 */  
00007  
00008 #ifndef INC_TEMP_MONITORING_H_  
00009 #define INC_TEMP_MONITORING_H_  
00010  
00011 #include "uvfr_utils.h"  
00012  
00013 uv_status initTempMonitor(void* args);  
00014  
00015 void tempMonitorTask(void* args);  
00016  
00017 #endif /* INC_TEMP_MONITORING_H_ */
```

7.39 Core/Inc/tim.h File Reference

This file contains all the function prototypes for the [tim.c](#) file.

```
#include "main.h"
```

Include dependency graph for tim.h: This graph shows which files directly or indirectly include this file:

Functions

- void [MX_TIM3_Init](#) (void)
- void [MX_TIM11_Init](#) (void)

Variables

- TIM_HandleTypeDef [htim3](#)
- TIM_HandleTypeDef [htim11](#)

7.39.1 Detailed Description

This file contains all the function prototypes for the [tim.c](#) file.

Attention

Copyright (c) 2024 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [tim.h](#).

7.39.2 Function Documentation

7.39.2.1 MX_TIM11_Init()

```
void MX_TIM11_Init (
    void )
```

Definition at line 71 of file [tim.c](#).

References [Error_Handler\(\)](#), and [htim11](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.39.2.2 MX_TIM3_Init()

```
void MX_TIM3_Init (
    void )
```

Definition at line 31 of file [tim.c](#).

References [Error_Handler\(\)](#), and [htim3](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.39.3 Variable Documentation

7.39.3.1 htim11

```
TIM_HandleTypeDef htim11 [extern]
```

Definition at line 28 of file [tim.c](#).

Referenced by [MX_TIM11_Init\(\)](#).

7.39.3.2 htim3

```
TIM_HandleTypeDef htim3 [extern]
```

Definition at line 27 of file [tim.c](#).

Referenced by [HAL_TIM_PeriodElapsedCallback\(\)](#), and [MX_TIM3_Init\(\)](#).

7.40 tim.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Define to prevent recursive inclusion -----*/
00021 #ifndef __TIM_H__
00022 #define __TIM_H__
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 /* Includes -----*/
00029 #include "main.h"
00030
00031 /* USER CODE BEGIN Includes */
00032
00033 /* USER CODE END Includes */
00034
00035 extern TIM_HandleTypeDef htim3;
00036
00037 extern TIM_HandleTypeDef htim11;
00038
00039 /* USER CODE BEGIN Private defines */
00040
00041 /* USER CODE END Private defines */
00042
00043 void MX_TIM3_Init(void);
00044 void MX_TIM11_Init(void);
00045
00046 /* USER CODE BEGIN Prototypes */
00047
00048 /* USER CODE END Prototypes */
00049
00050 #ifdef __cplusplus
00051 }
00052 #endif
00053
00054 #endif /* __TIM_H__ */
00055

```

7.41 Core/Inc/uvfr_conifer.h File Reference

```
#include "stdlib.h"
#include "stdint.h"
```

Include dependency graph for uvfr_conifer.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [abstract_conifer_channel](#)
- struct [conifer_ch_setting](#)
- struct [conifer_settings](#)
- struct [conifer_state](#)
- struct [conifer_hw_channel](#)

Macros

- [#define conifer_params current_vehicle_settings->conifer_settings](#)
- [#define CONIFER_CH_EN_BIT \(0x0001U<<0\)](#)
- [#define CONIFER_CH_IS_CRIT_BIT \(0x0001U<<1\)](#)
- [#define CONIFER_CH_PSRC_BIT \(0x0001U<<2\)](#)
- [#define CONIFER_CH_HB_DIR_MASK \(\(0x0001U<<3\)|\(0x0001U<<4\)\)](#)
- [#define CONIFER_CH_EN_OCP \(0x0001U<<5\)](#)

- #define CONIFER_CH_FLT_BIT (0x0001U<<6)
- #define CONIFER_CH_ILIM_BIT (0x0001U<<7)
- #define CONIFER_CH_ALLOW_LOAD_SHED (0x0001U<<8)
- #define CONIFER_CH_LS_ACTIVE (0x0001U<<9)
- #define CONIFER_CH_IN_USE (0x0001U<<15)
- #define CONIFER_CH_LOC_MASK (0x0700U)
- #define CONIFER_HW_CH_ID_MASK (0x00FF)
- #define IS_CONIFER_CH_USED(ch)
- #define GET_CONIFER_CH_LOC(ch)
- #define EXPECT_UV19PDU (0x01U<<0)
- #define EXPECT_ECUMASTER_PMU16 (0x01U<<1)
- #define EN_SW_OCP (0x01U<<2)
- #define EN_DYNAMIC_LOAD_SHEDDING (0x01U<<3)
- #define ROUND_PWM_ON_BINARY_CHANNELS (0x01U<<8)
- #define USE_ABS_VAL_MAPPING_HBRIDGE_TO_PWM (0x01U<<9)
- #define CONIFER_CH_TYPE_MASK 0x0007

Typedefs

- typedef struct conifer_ch_setting conifer_ch_setting
- typedef enum CONIFER_OUTPUT conifer_output_channel
- typedef enum conifer_driver_ecode conifer_driver_ecode
- typedef struct conifer_settings conifer_settings
- typedef enum conifer_ch_location conifer_ch_location
- typedef struct conifer_state conifer_state
- typedef enum conifer_hw_ch_type conifer_hw_ch_type
- typedef struct conifer_hw_channel conifer_hw_channel

Enumerations

- enum CONIFER_OUTPUT {
 VCU_SAFETY , BAMO_RFE , BAMO_RUN , BAMO_PWR ,
 SDC_BOARD_PWR , HVIL_PWR , DCDC_EN , COOLANT_PUMP1 ,
 COOLANT_PUMP2 , RAD_FANS1 , RAD_FANS2 , RAD_FANS3 ,
 RAD_FANS4 , ACCU_FANS1 , ACCU_FANS2 , BRAKE_LIGHT ,
 HORN , IMD_PWR , GENERAL_ACCU_PWR1 , GENERAL_ACCU_PWR2 ,
 VCU_PWR , RTML_PWR , TSSI_PWR , BSPD_PWR ,
 DASH_PWR , FINAL_CONIFER_OUTPUT }
- enum conifer_driver_ecode {
 CONIFER_DRV_OK , CONIFER_DRV_INVALID_HW_LOC , CONIFER_DRV_INVALID_HW_CH_ID ,
 CONIFER_DRV_INVALID_ABSTRACT_CH ,
 CONIFER_EXT_DEVICE_NOT_FOUND , CONIFER_EXT_DEVICE_NOR_RESPOND , CONIFER_DRV_INIT_ERR
 }
- enum conifer_ch_location {
 LOCAL_CH = 0b000 , UV19_PDU_CH = 0b001 , ECUMASTER_PMU16_CH = 0b010 , RESERVED_CH_POS1
 = 0b011 ,
 RESERVED_CH_POS2 = 0b100 , RESERVED_CH_POS3 = 0b101 , RESERVED_CH_POS4 = 0b110 ,
 INVALID_MAPPING = 0b111 }
- enum conifer_hw_ch_type {
 BIN_CH = 0x00 , DUAL_BUS_CH = 0x01 , PWM_CH = 0x02 , DUAL_BUS_PWM_CH = 0x03 ,
 H_BRIDGE = 0x04 , INDEPENDENT_REG_CH = 0x05 , EXT_BIN_CH = 0x06 , SPECIAL = 0x07 }

Functions

- [uv_status coniferInit \(\)](#)
Initializes the conifer library.
- [uv_status coniferDeInit \(\)](#)
- [uv_status coniferEnChannel \(conifer_output_channel ch\)](#)
Enables a conifer output channel.
- [uv_status coniferDisChannel \(conifer_output_channel ch\)](#)
Disables a conifer output channel.
- [uv_status coniferToggleChannel \(conifer_output_channel ch\)](#)
Toggles a conifer output channel.
- [uv_status coniferSetDutyCycle \(conifer_output_channel ch, uint8_t duty\)](#)
Sets Duty Cycle of a conifer managed PWM output.
- [uv_status coniferSetDirection \(conifer_output_channel ch, uint8_t dir\)](#)
Sets direction of a conifer managed H-bridge output.
- [uv_status coniferSetDutyCycleAndDirection \(conifer_output_channel ch, uint8_t duty, uint8_t dir\)](#)
Set duty cycle and direction of a conifer managed.
- [uint16_t coniferGetChannelCurrent \(conifer_output_channel ch\)](#)
- [uint16_t coniferGetChannelFbck \(conifer_output_channel ch\)](#)
- [uint16_t coniferGetChannelFaults \(conifer_output_channel ch\)](#)

7.41.1 Macro Definition Documentation

7.41.1.1 CONIFER_CH_ALLOW_LOAD_SHED

```
#define CONIFER_CH_ALLOW_LOAD_SHED (0x0001U<<8)
```

Definition at line 27 of file [uvfr_conifer.h](#).

Referenced by [__attribute__\(\)](#), and [coniferEnLoadShedding\(\)](#).

7.41.1.2 CONIFER_CH_EN_BIT

```
#define CONIFER_CH_EN_BIT (0x0001U<<0)
```

Definition at line 20 of file [uvfr_conifer.h](#).

Referenced by [__attribute__\(\)](#), [coniferEnChannel\(\)](#), [coniferToggleChannel\(\)](#), and [u19updatePduChannel\(\)](#).

7.41.1.3 CONIFER_CH_EN_OCP

```
#define CONIFER_CH_EN_OCP (0x0001U<<5)
```

Definition at line 24 of file [uvfr_conifer.h](#).

7.41.1.4 CONIFER_CH_FLT_BIT

```
#define CONIFER_CH_FLT_BIT (0x0001U<<6)
```

Definition at line 25 of file [uvfr_conifer.h](#).

Referenced by [u19updatePduChannel\(\)](#).

7.41.1.5 CONIFER_CH_HB_DIR_MASK

```
#define CONIFER_CH_HB_DIR_MASK ((0x0001U<<3)|(0x0001U<<4))
```

Definition at line 23 of file [uvfr_conifer.h](#).

7.41.1.6 CONIFER_CH_ILIM_BIT

```
#define CONIFER_CH_ILIM_BIT (0x0001U<<7)
```

Definition at line 26 of file [uvfr_conifer.h](#).

7.41.1.7 CONIFER_CH_IN_USE

```
#define CONIFER_CH_IN_USE (0x0001U<<15)
```

Definition at line 31 of file [uvfr_conifer.h](#).

Referenced by [__attribute__\(\)](#).

7.41.1.8 CONIFER_CH_IS_CRIT_BIT

```
#define CONIFER_CH_IS_CRIT_BIT (0x0001U<<1)
```

Definition at line 21 of file [uvfr_conifer.h](#).

Referenced by [__attribute__\(\)](#).

7.41.1.9 CONIFER_CH_LOC_MASK

```
#define CONIFER_CH_LOC_MASK (0x0700U)
```

Definition at line 32 of file [uvfr_conifer.h](#).

Referenced by [coniferGetChannelCurrent\(\)](#), and [coniferGetChannelFbck\(\)](#).

7.41.1.10 CONIFER_CH_LS_ACTIVE

```
#define CONIFER_CH_LS_ACTIVE (0x0001U<<9)
```

Definition at line 28 of file [uvfr_conifer.h](#).

Referenced by [coniferDisLoadShedding\(\)](#), [coniferEnLoadShedding\(\)](#), and [u19updatePduChannel\(\)](#).

7.41.1.11 CONIFER_CH_PSRC_BIT

```
#define CONIFER_CH_PSRC_BIT (0x0001U<<2)
```

Definition at line 22 of file [uvfr_conifer.h](#).

7.41.1.12 CONIFER_CH_TYPE_MASK

```
#define CONIFER_CH_TYPE_MASK 0x0007
```

Definition at line 188 of file [uvfr_conifer.h](#).

7.41.1.13 CONIFER_HW_CH_ID_MASK

```
#define CONIFER_HW_CH_ID_MASK (0x00FF)
```

Definition at line 33 of file [uvfr_conifer.h](#).

7.41.1.14 conifer_params

```
#define conifer_params current_vehicle_settings->conifer_settings
```

Definition at line 7 of file [uvfr_conifer.h](#).

Referenced by [coniferInit\(\)](#), and [initPDU\(\)](#).

7.41.1.15 EN_DYNAMIC_LOAD_SHEDDING

```
#define EN_DYNAMIC_LOAD_SHEDDING (0x01U<<3)
```

Definition at line 97 of file [uvfr_conifer.h](#).

7.41.1.16 EN_SW_OCP

```
#define EN_SW_OCP (0x01U<<2)
```

Definition at line 96 of file [uvfr_conifer.h](#).

7.41.1.17 EXPECT_ECUMASTER_PMU16

```
#define EXPECT_ECUMASTER_PMU16 (0x01U<<1)
```

Definition at line 95 of file [uvfr_conifer.h](#).

7.41.1.18 EXPECT_UV19PDU

```
#define EXPECT_UV19PDU (0x01U<<0)
```

Definition at line 94 of file [uvfr_conifer.h](#).

Referenced by [coniferInit\(\)](#).

7.41.1.19 GET_CONIFER_CH_LOC

```
#define GET_CONIFER_CH_LOC(  
    ch)
```

Value:

```
((ch->hardware_mapping&CONIFER_CH_LOC_MASK) >> 8)
```

Definition at line 36 of file [uvfr_conifer.h](#).

7.41.1.20 IS_CONIFER_CH_USED

```
#define IS_CONIFER_CH_USED(  
    ch)
```

Value:

```
(ch->hardware_mapping&CONIFER_CH_IN_USE)
```

Definition at line 35 of file [uvfr_conifer.h](#).

Referenced by [coniferDisChannel\(\)](#), [coniferEnChannel\(\)](#), [coniferGetChannelCurrent\(\)](#), [coniferInit\(\)](#), [coniferSetDirection\(\)](#), [coniferSetDutyCycle\(\)](#), [coniferSetDutyCycleAndDirection\(\)](#), and [coniferToggleChannel\(\)](#).

7.41.1.21 ROUND_PWM_ON_BINARY_CHANNELS

```
#define ROUND_PWM_ON_BINARY_CHANNELS (0x01U<<8)
```

Definition at line 99 of file [uvfr_conifer.h](#).

7.41.1.22 USE_ABS_VAL_MAPPING_HBRIDGE_TO_PWM

```
#define USE_ABS_VAL_MAPPING_HBRIDGE_TO_PWM (0x01U<<9)
```

Definition at line 100 of file [uvfr_conifer.h](#).

7.41.2 Typedef Documentation

7.41.2.1 `conifer_ch_location`

```
typedef enum conifer_ch_location conifer_ch_location
```

7.41.2.2 `conifer_ch_setting`

```
typedef struct conifer_ch_setting conifer_ch_setting
```

7.41.2.3 `conifer_driver_ecode`

```
typedef enum conifer_driver_ecode conifer_driver_ecode
```

7.41.2.4 `conifer_hw_ch_type`

```
typedef enum conifer_hw_ch_type conifer_hw_ch_type
```

7.41.2.5 `conifer_hw_channel`

```
typedef struct conifer_hw_channel conifer_hw_channel
```

7.41.2.6 `conifer_output_channel`

```
typedef enum CONIFER_OUTPUT conifer_output_channel
```

7.41.2.7 `conifer_settings`

```
typedef struct conifer_settings conifer_settings
```

7.41.2.8 `conifer_state`

```
typedef struct conifer_state conifer_state
```

7.41.3 Enumeration Type Documentation

7.41.3.1 `conifer_ch_location`

```
enum conifer_ch_location
```

Enumerator

LOCAL_CH
UV19_PDU_CH
ECUMASTER_PMU16_CH
RESERVED_CH_POS1
RESERVED_CH_POS2
RESERVED_CH_POS3
RESERVED_CH_POS4
INVALID_MAPPING

Definition at line 149 of file [uvfr_conifer.h](#).

7.41.3.2 conifer_driver_ecode

```
enum conifer_driver_ecode
```

Enumerator

CONIFER_DRV_OK
CONIFER_DRV_INVALID_HW_LOC
CONIFER_DRV_INVALID_HW_CH_ID
CONIFER_DRV_INVALID_ABSTRACT_CH
CONIFER_EXT_DEVICE_NOT_FOUND
CONIFER_EXT_DEVICE_NOR_RESPOND
CONIFER_DRV_INIT_ERR

Definition at line 84 of file [uvfr_conifer.h](#).

7.41.3.3 conifer_hw_ch_type

```
enum conifer_hw_ch_type
```

Enumerator

BIN_CH
DUAL_BUS_CH
PWM_CH
DUAL_BUS_PWM_CH
H_BRIDGE
INDEPENDENT_REG_CH
EXT_BIN_CH
SPECIAL

Definition at line 169 of file [uvfr_conifer.h](#).

7.41.3.4 CONIFER_OUTPUT

```
enum CONIFER_OUTPUT
```

Enumerator

VCU_SAFETY
BAMO_RFE
BAMO_RUN
BAMO_PWR
SDC_BOARD_PWR
HVIL_PWR
DCDC_EN
COOLANT_PUMP1
COOLANT_PUMP2
RAD_FANS1
RAD_FANS2
RAD_FANS3
RAD_FANS4
ACCU_FANS1
ACCU_FANS2
BRAKE_LIGHT
HORN
IMD_PWR
GENERAL_ACCU_PWR1
GENERAL_ACCU_PWR2
VCU_PWR
RTML_PWR
TSSI_PWR
BSPD_PWR
DASH_PWR
FINAL_CONIFER_OUTPUT

Definition at line 45 of file [uvfr_conifer.h](#).

7.41.4 Function Documentation

7.41.4.1 coniferDeInit()

```
uv_status coniferDeInit ()
```

Definition at line 232 of file [uvfr_conifer.c](#).

References [UV_OK](#).

7.41.4.2 coniferDisChannel()

```
uv_status coniferDisChannel (
    conifer_output_channel ch)
```

Disables a conifer output channel.

Definition at line 300 of file [uvfr_conifer.c](#).

References [IS_CONIFER_CH_USED](#), [abstract_conifer_channel::status_control_reg](#), and [UV_ABORTED](#).

Referenced by [BeepBeepMotherFucker\(\)](#), and [uvSecureVehicle\(\)](#).

7.41.4.3 coniferEnChannel()

```
uv_status coniferEnChannel (
    conifer_output_channel ch)
```

Enables a conifer output channel.

Definition at line 285 of file [uvfr_conifer.c](#).

References [CONIFER_CH_EN_BIT](#), [IS_CONIFER_CH_USED](#), [abstract_conifer_channel::status_control_reg](#), and [UV_ABORTED](#).

Referenced by [BeepBeepMotherFucker\(\)](#), and [uvInit\(\)](#).

7.41.4.4 coniferGetChannelCurrent()

```
uint16_t coniferGetChannelCurrent (
    conifer_output_channel ch)
```

Definition at line 362 of file [uvfr_conifer.c](#).

References [CONIFER_CH_LOC_MASK](#), [ECUMASTER_PMU16_CH](#), [abstract_conifer_channel::hardware_mapping](#), [IS_CONIFER_CH_USED](#), [LOCAL_CH](#), [UV19_PDU_CH](#), and [UV_ABORTED](#).

7.41.4.5 coniferGetChannelFaults()

```
uint16_t coniferGetChannelFaults (
    conifer_output_channel ch)
```

Definition at line 407 of file [uvfr_conifer.c](#).

7.41.4.6 coniferGetChannelFbck()

```
uint16_t coniferGetChannelFbck (
    conifer_output_channel ch)
```

Definition at line 386 of file [uvfr_conifer.c](#).

References [CONIFER_CH_LOC_MASK](#), [ECUMASTER_PMU16_CH](#), [abstract_conifer_channel::hardware_mapping](#), [LOCAL_CH](#), and [UV19_PDU_CH](#).

7.41.4.7 coniferInit()

```
uv_status coniferInit ()
```

Initializes the conifer library.

Definition at line 173 of file [uvfr_conifer.c](#).

References [ch_updaters](#), [conifer_params](#), [EXPECT_UV19PDU](#), [FINAL_CONIFER_OUTPUT](#), [abstract_conifer_channel::hardware_mappingPDU\(\)](#), [IS_CONIFER_CH_USED](#), [abstract_conifer_channel::status_control_reg](#), [u19updatePduChannel\(\)](#), [UV19_PDU_CH](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.41.4.8 coniferSetDirection()

```
uv_status coniferSetDirection (
    conifer_output_channel ch,
    uint8_t dir)
```

Sets direction of a conifer managed H-bridge output.

Definition at line 340 of file [uvfr_conifer.c](#).

References [IS_CONIFER_CH_USED](#), [UV_ABORTED](#), and [UV_ERROR](#).

7.41.4.9 coniferSetDutyCycle()

```
uv_status coniferSetDutyCycle (
    conifer_output_channel ch,
    uint8_t duty)
```

Sets Duty Cycle of a conifer managed PWM output.

Definition at line 328 of file [uvfr_conifer.c](#).

References [IS_CONIFER_CH_USED](#), [UV_ABORTED](#), and [UV_ERROR](#).

7.41.4.10 coniferSetDutyCycleAndDirection()

```
uv_status coniferSetDutyCycleAndDirection (
    conifer_output_channel ch,
    uint8_t duty,
    uint8_t dir)
```

Set duty cycle and direction of a conifer managed.

Definition at line 352 of file [uvfr_conifer.c](#).

References [IS_CONIFER_CH_USED](#), [UV_ABORTED](#), and [UV_ERROR](#).

7.41.4.11 coniferToggleChannel()

```
uv_status coniferToggleChannel (
    conifer_output_channel ch)
```

Toggles a conifer output channel.

Definition at line 314 of file [uvfr_conifer.c](#).

References [CONIFER_CH_EN_BIT](#), [IS_CONIFER_CH_USED](#), [abstract_conifer_channel::status_control_reg](#), and [UV_ABORTED](#).

Referenced by [BeepBeepMotherFucker\(\)](#).

7.42 uvfr_conifer.h

[Go to the documentation of this file.](#)

```

00001 #ifndef CONIFER_H
00002 #define CONIFER_H
00003
00004 #include "stdlib.h"
00005 #include "stdint.h"
00006
00007 #define conifer_params current_vehicle_settings->conifer_settings
00008
00009 typedef enum uv_status_t uv_status;
00010
00011 //This is the abstract layer of a conifer channel
00012 typedef struct abstract_conifer_channel{
00013     uint16_t status_control_reg;
00014     uint16_t hardware_mapping;
00015     uint16_t ilim;
00016     uint16_t duty;
00017 }abstract_conifer_channel;
00018
00019 //Macros for use with the conifer_status_control_reg
00020 #define CONIFER_CH_EN_BIT (0x0001U<<0)
00021 #define CONIFER_CH_IS_CRIT_BIT (0x0001U<<1)
00022 #define CONIFER_CH_PSRC_BIT (0x0001U<<2)
00023 #define CONIFER_CH_HB_DIR_MASK ((0x0001U<<3) | (0x0001U<<4))
00024 #define CONIFER_CH_EN_OCP (0x0001U<<5)
00025 #define CONIFER_CH_FLT_BIT (0x0001U<<6)
00026 #define CONIFER_CH_ILIM_BIT (0x0001U<<7)
00027 #define CONIFER_CH_ALLOW_LOAD_SHED (0x0001U<<8)
00028 #define CONIFER_CH_LS_ACTIVE (0x0001U<<9)
00029
00030 //Macros for usage with the hardware_mapping field of the abstract_conifer_channel
00031 #define CONIFER_CH_IN_USE (0x0001U<<15)
00032 #define CONIFER_CH_LOC_MASK (0x0700U)
00033 #define CONIFER_HW_CH_ID_MASK (0x00FF)
00034
00035 #define IS_CONIFER_CH_USED(ch) (ch->hardware_mapping&CONIFER_CH_IN_USE)
00036 #define GET_CONIFER_CH_LOC(ch) ((ch->hardware_mapping&CONIFER_CH_LOC_MASK)>>8)
00037
00038 typedef struct conifer_ch_setting{
00039     struct abstract_conifer_channel ch_dat;
00040     uint16_t ch;
00041     uint16_t reserved;
00042 }conifer_ch_setting;
00043
00044 //Enum that is the human readable names of the abstract conifer outputs
00045 typedef enum CONIFER_OUTPUT{
00046     VCU_SAFETY,
00047     BAMO_RFE,
00048     BAMO_RUN,
00049     BAMO_PWR,
00050     SDC_BOARD_PWR,
00051     HVIL_PWR,
00052     DCDC_EN,
00053     COOLANT_PUMP1,
00054     COOLANT_PUMP2,
00055     RAD_FANS1,
00056     RAD_FANS2,
00057     RAD_FANS3,
00058     RAD_FANS4,
00059     ACCU_FANS1,
00060     ACCU_FANS2,
00061     BRAKE_LIGHT,
00062     HORN,
00063     //DRS_FL1,
00064     //DRS_FL2,
00065     //DRS_FR1,
00066     //DRS_FR2,
00067     //DRS_R1,
00068     //DRS_R2,
00069     //DRS_R3,
00070     //BMS_ALWAYS_ON,
00071     //BMS_DISCHARGE_PWR,
00072     IMD_PWR,
00073     GENERAL_ACCU_PWR1,
00074     GENERAL_ACCU_PWR2,
00075     VCU_PWR, //Yes, the VCU can turn itself off technically. You should not however.
00076     RTML_PWR,
00077     TSSI_PWR,
00078     BSPD_PWR,
00079     DASH_PWR,
00080
00081     FINAL_CONIFER_OUTPUT
00082 }conifer_output_channel;

```

```

00083
00084 typedef enum conifer_driver_ecode{
00085     CONIFER_DRV_OK,
00086     CONIFER_DRV_INVALID_HW_LOC,
00087     CONIFER_DRV_INVALID_HW_CH_ID,
00088     CONIFER_DRV_INVALID_ABSTRACT_CH,
00089     CONIFER_EXT_DEVICE_NOT_FOUND,
00090     CONIFER_EXT_DEVICE_NOR_RESPOND,
00091     CONIFER_DRV_INIT_ERR
00092 }conifer_driver_ecode;
00093
00094 #define EXPECT_UV19PDU (0x01U<0) //if this bit is set, that means that there should be a pdu connected
00095 to the CANbus
00096 #define EXPECT_ECUMASTER_PMU16 (0x01U<1) //if this bit is set, that means there should be a PMU16
00097 somewhere
00098 #define EN_SW_OCP (0x01U<2) //Enables SW OCP
00099 #define EN_DYNAMIC_LOAD_SHEDDING (0x01U<3) //Enables dynamic load shedding
00100
00101
00102
00103 //The base config settings for CONIFER
00104 typedef struct conifer_settings{
00105     uint32_t conif_flags; //4
00106
00107     //These variables are from the future, you wouldnt understand them yet
00108     uint16_t OCP_flt_time; //ms
00109     uint16_t OCP_load_shed_time; //ms //8
00110
00111
00112     uint16_t sys_fault_current; //Fault current of entire sys
00113     uint16_t sys_cont_current; //Continuous Current Rating of system
00114     //12
00115
00116     uint16_t sys_excess_I2T; //in A*ms
00117     uint16_t sys_ocp_timeout;
00118     //16
00119
00120     uint16_t sys_max_voltage;
00121     uint16_t sys_lv_threshold_voltage; //If V_SYS drops below this value, it will trigger emergency
00122     load shedding
00123     //20
00124
00125     uint16_t sys_cutoff_voltage; //If V_SYS drops below this voltage
00126     uint16_t regbusA_target_voltage;
00127     //24
00128
00129     uint16_t regbusA_fault_current;
00130     uint16_t regbusA_continuous_current;
00131     //28
00132
00133     uint16_t regbusB_target_voltage;
00134     uint16_t regbusB_fault_current;
00135     //32
00136
00137     uint16_t regbusB_continuous_current;
00138     uint8_t n_ch;
00139     uint8_t pdu_bus;
00140     //36
00141
00142     uint8_t dpms_bus;
00143     uint8_t ecmaster_bus;
00144
00145     struct conifer_ch_setting ch_list[32]; //The 55 puts us to exactly 256 bytes
00146 }conifer_settings;
00147
00148 //Enum representing the physical location of a conifer channel
00149 typedef enum conifer_ch_location{
00150     LOCAL_CH = 0b000,
00151     UV19_PDU_CH = 0b001,
00152     ECUMASTER_PMU16_CH = 0b010,
00153     RESERVED_CH_POS1 = 0b011,
00154     RESERVED_CH_POS2 = 0b100,
00155     RESERVED_CH_POS3 = 0b101,
00156     RESERVED_CH_POS4 = 0b110,
00157     INVALID_MAPPING = 0b111
00158 }conifer_ch_location;
00159
00160 //typedef enum conifer_stat_flags{
00161 //
00162 //};
00163
00164 typedef struct conifer_state{
00165     uint32_t status_flags;
00166     uint8_t load_shedding;

```

```

00167 }conifer_state;
00168
00169 typedef enum conifer_hw_ch_type{ //This represents the type of a hardware channel
00170     BIN_CH = 0x00,
00171     DUAL_BUS_CH = 0x01,
00172     PWM_CH = 0x02,
00173     DUAL_BUS_PWM_CH = 0x03,
00174     H_BRIDGE = 0x04,
00175     INDEPENDENT_REG_CH = 0x05,
00176     EXT_BIN_CH = 0x06,
00177     SPECIAL = 0x07
00178 }conifer_hw_ch_type;
00179
00180 //This is a lower level hardware level thing
00181 typedef struct conifer_hw_channel{
00182     uint16_t ch_info_reg;
00183
00184 }conifer_hw_channel;
00185
00186
00187
00188 #define CONIFER_CH_TYPE_MASK 0x0007
00189
00190 //Initializes whatever conifer things need initialized
00191 uv_status coniferInit();
00192 uv_status coniferDeInit();
00193
00194
00195 uv_status coniferEnChannel(conifer_output_channel ch); //
00196 uv_status coniferDisChannel(conifer_output_channel ch);
00197 uv_status coniferToggleChannel(conifer_output_channel ch);
00198 uv_status coniferSetDutyCycle(conifer_output_channel ch, uint8_t duty);
00199 uv_status coniferSetDirection(conifer_output_channel ch, uint8_t dir);
00200 uv_status coniferSetDutyCycleAndDirection(conifer_output_channel ch, uint8_t duty, uint8_t dir);
00201
00202 uint16_t coniferGetChannelCurrent(conifer_output_channel ch);
00203 uint16_t coniferGetChannelFbck(conifer_output_channel ch);
00204 uint16_t coniferGetChannelFaults(conifer_output_channel ch);
00205
00206
00207
00208 #endif

```

7.43 Core/Inc/uvfr_diagnostics.h File Reference

#include <string.h>

Include dependency graph for uvfr_diagnostics.h: This graph shows which files directly or indirectly include this file:

Macros

- #define DIAGNOSTIC_RX_MSG_ID 0x421
- #define DIAGNOSTIC_TX_MSG_ID 0x521
- #define TEXTIFY(A)
- #define uvAssert(x)

Enumerations

- enum diagnostic_cmd {
 ENTER_DIAGNOSTICS_MODE , EXIT_DIAGNOSTICS_MODE , REQUEST_STATE_CHANGE ,
 FORCE_STATE_CHANGE ,
 dCLEAR_FAULTS }
- enum logged_event_type {
 LOG_FATAL_ERROR , LOG_ERROR , LOG_WARNING , LOG_STATE_CHANGE ,
 LOG_GENERIC_EVENT }

Functions

- [uv_status uvInitDiagnostics \(\)](#)
- [uv_status logDiagnosticEvent \(\)](#)

7.43.1 Macro Definition Documentation

7.43.1.1 DIAGNOSTIC_RX_MSG_ID

```
#define DIAGNOSTIC_RX_MSG_ID 0x421
```

Definition at line [11](#) of file [uvfr_diagnostics.h](#).

7.43.1.2 DIAGNOSTIC_TX_MSG_ID

```
#define DIAGNOSTIC_TX_MSG_ID 0x521
```

Definition at line [12](#) of file [uvfr_diagnostics.h](#).

7.43.1.3 TEXTIFY

```
#define TEXTIFY( A)
```

Value:

```
#A
```

Definition at line [19](#) of file [uvfr_diagnostics.h](#).

7.43.1.4 uvAssert

```
#define uvAssert( x)
```

Value:

```
if((x) == 0){\
    extern void uvAssertFailed(char* file, uint16_t line, TaskHandle_t task, char* condition);\
    TaskHandle_t ctask = xTaskGetCurrentTaskHandle();\
    uvAssertFailed(__UV_FILENAME__, __LINE__, ctask, TEXTIFY(x));\
}
```

Definition at line [24](#) of file [uvfr_diagnostics.h](#).

Referenced by [main\(\)](#).

7.43.2 Enumeration Type Documentation

7.43.2.1 diagnostic_cmd

```
enum diagnostic_cmd
```

Enumerator

ENTER_DIAGNOSTICS_MODE	
EXIT_DIAGNOSTICS_MODE	
REQUEST_STATE_CHANGE	
FORCE_STATE_CHANGE	
dCLEAR_FAULTS	

Definition at line 31 of file [uvfr_diagnostics.h](#).

7.43.2.2 logged_event_type

```
enum logged_event_type
```

Enumerator

LOG_FATAL_ERROR	
LOG_ERROR	
LOG_WARNING	
LOG_STATE_CHANGE	
LOG_GENERIC_EVENT	

Definition at line 39 of file [uvfr_diagnostics.h](#).

7.43.3 Function Documentation

7.43.3.1 logDiagnosticEvent()

```
uv_status logDiagnosticEvent ()
```

7.43.3.2 uvInitDiagnostics()

```
uv_status uvInitDiagnostics ()
```

Definition at line 18 of file [uvfr_diagnostics.c](#).

References [UV_OK](#).

Referenced by [uvInit\(\)](#).

7.44 uvfr_diagnostics.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * uvfr_diagnostics_system.h
00003  *
00004  * Created on: Jun 14, 2025
00005  * Author: byo10
00006 */
00007
00008 #ifndef INC_UVFR_DIAGNOSTICS_H_
00009 #define INC_UVFR_DIAGNOSTICS_H_
00010
00011 #define DIAGNOSTIC_RX_MSG_ID 0x421 //MSG from laptop or PCAN
00012 #define DIAGNOSTIC_TX_MSG_ID 0x521 //MSG from VCU
00013
00014
00015
00016 #include<string.h>
00017
00018 #ifndef TEXTIFY
00019     #define TEXTIFY(A) #A
00020 #endif
00021
00022
00023
00024 #define uvAssert( x ) if((x) == 0){\
00025     extern void uvAssertFailed(char* file, uint16_t line, TaskHandle_t task, char* condition);\
00026     TaskHandle_t ctask = xTaskGetCurrentTaskHandle();\
00027     uvAssertFailed(__UV_FILENAME__, __LINE__, ctask, TEXTIFY(x));\
00028 }
00029
00030
00031 typedef enum{
00032     ENTER_DIAGNOSTICS_MODE,
00033     EXIT_DIAGNOSTICS_MODE,
00034     REQUEST_STATE_CHANGE,
00035     FORCE_STATE_CHANGE,
00036     dCLEAR_FAULTS
00037 }diagnostic_cmd;
00038
00039 typedef enum{
00040     LOG_FATAL_ERROR,
00041     LOG_ERROR,
00042     LOG_WARNING,
00043     LOG_STATE_CHANGE,
00044     LOG_GENERIC_EVENT
00045 }logged_event_type;
00046
00047 uv_status uvInitDiagnostics();
00048
00049 uv_status logDiagnosticEvent(); //THIS NEEDS ARGS
00050
00051 #endif /* INC_UVFR_DIAGNOSTICS_H_ */

```

7.45 Core/Inc/uvfr_global_config.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define UV19_PDU 1
- #define ECUMASTER_PMU 0
- #define STM32_F407 1
- #define STM32_H7xx 0
- #define FIRMWARE_MAJOR_RELEASE 0
- #define FIRMWARE_MINOR_RELEASE 1
- #define FIRMWARE_PATCH_NUM 0
- #define UV_MALLOC_LIMIT ((size_t)1024)
- #define USE_OS_MEM_MGMT 1

7.45.1 Macro Definition Documentation

7.45.1.1 ECUMASTER_PMU

```
#define ECUMASTER_PMU 0
```

Definition at line 13 of file [uvfr_global_config.h](#).

7.45.1.2 FIRMWARE_MAJOR_RELEASE

```
#define FIRMWARE_MAJOR_RELEASE 0
```

Definition at line 18 of file [uvfr_global_config.h](#).

Referenced by [handleIncomingLaptopMsg\(\)](#), and [uvSaveSettingsToFlash\(\)](#).

7.45.1.3 FIRMWARE_MINOR_RELEASE

```
#define FIRMWARE_MINOR_RELEASE 1
```

Definition at line 19 of file [uvfr_global_config.h](#).

Referenced by [handleIncomingLaptopMsg\(\)](#), and [uvSaveSettingsToFlash\(\)](#).

7.45.1.4 FIRMWARE_PATCH_NUM

```
#define FIRMWARE_PATCH_NUM 0
```

Definition at line 20 of file [uvfr_global_config.h](#).

Referenced by [handleIncomingLaptopMsg\(\)](#), and [uvSaveSettingsToFlash\(\)](#).

7.45.1.5 STM32_F407

```
#define STM32_F407 1
```

Definition at line 15 of file [uvfr_global_config.h](#).

7.45.1.6 STM32_H7xx

```
#define STM32_H7xx 0
```

Definition at line 16 of file [uvfr_global_config.h](#).

7.45.1.7 USE_OS_MEM_MGMT

```
#define USE_OS_MEM_MGMT 1
```

Definition at line 30 of file [uvfr_global_config.h](#).

7.45.1.8 UV19_PDU

```
#define UV19_PDU 1
```

Definition at line 12 of file [uvfr_global_config.h](#).

7.45.1.9 UV_MALLOC_LIMIT

```
#define UV_MALLOC_LIMIT ((size_t)1024)
```

Definition at line 26 of file [uvfr_global_config.h](#).

7.46 uvfr_global_config.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * uvfr_global_config.h
00003  *
00004  *   Created on: Nov 27, 2024
00005  *       Author: byo10
00006 */
00007
00008 #ifndef INC_UVFR_GLOBAL_CONFIG_H_
00009 #define INC_UVFR_GLOBAL_CONFIG_H_
00010
00011 //VCU Port Configuration
00012 #define UV19_PDU 1 //if 1, then we still use the uv19 PMU
00013 #define ECUMASTER_PMU 0
00014
00015 #define STM32_F407 1
00016 #define STM32_H7xx 0
00017
00018 #define FIRMWARE_MAJOR_RELEASE 0
00019 #define FIRMWARE_MINOR_RELEASE 1
00020 #define FIRMWARE_PATCH_NUM 0
00021
00022
00023
00024 //memory management configuration
00025 #ifndef UV_MALLOC_LIMIT
00026 #define UV_MALLOC_LIMIT ((size_t)1024)
00027 #endif
00028
00029 #ifndef USE_OS_MEM_MGMT
00030 #define USE_OS_MEM_MGMT 1
00031 #endif
00032
00033
00034
00035
00036 #endif /* INC_UVFR_GLOBAL_CONFIG_H_ */
```

7.47 Core/Inc/uvfr_settings.h File Reference

```
#include "motor_controller.h"
#include "driving_loop.h"
#include "uvfr_utils.h"
#include "main.h"
#include "daq.h"
#include "bms.h"
```

Include dependency graph for uvfr_settings.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct `veh_gen_info`
- struct `uv_vehicle_settings`
- struct `motor_controller_settings`

Macros

- #define ENABLE_FLASH_SETTINGS 0
- #define START_OF_USER_FLASH &_s_uvdata
- #define TOP_OF_USER_FLASH &_e_uvdata
- #define TOP_OF_FLASH_SBLOCK (void*)0x080FFFFF
- #define SBLOCK_CSR_OFFSET 0x0C00
- #define SYS_DATA_OFFSET 0x00
- #define SIZE_OF_MGROUP 0x0100
- #define SBLOCK_CRC_REGION_OFFSET
- #define SIZEOF_USER_FLASH 4096
- #define SIZEOF_SBLOCK 4096
- #define FLASH_SBLOCK_START START_OF_USER_FLASH
- #define isPtrToFlash(p)
- #define GENERAL_VEH_INFO_MGROUP 0
- #define GENERAL_VEH_INFO_OFFSET 32
- #define OS_SETTINGS_MGROUP 0
- #define OS_SETTINGS_OFFSET 128
- #define OS_SETTINGS_ADDR ((void*)(START_OF_USER_FLASH + OS_SETTINGS_MGROUP*256 + OS_SETTINGS_OFFSET))
- #define MOTOR_MGROUP 1
- #define MOTOR_OFFSET 0
- #define MOTOR_ADDR (struct motor_controller_settings*)(START_OF_USER_FLASH + MOTOR_MGROUP*256 + MOTOR_OFFSET)
- #define DRIVING_MGROUP 2
- #define DRIVING_OFFSET 0
- #define DRIVING_ADDR ((driving_loop_args*)(START_OF_USER_FLASH + DRIVING_MGROUP + DRIVING_OFFSET))
- #define BMS_MGROUP 4
- #define BMS_OFFSET 0
- #define BMS_ADDR NULL
- #define IMD_MGROUP 4
- #define IMD_OFFSET 128
- #define IMD_ADDR ((void*)(START_OF_USER_FLASH + IMD_MGROUP*256 + IMD_OFFSET))
- #define CONIFER_MGROUP 5
- #define CONIFER_OFFSET 0
- #define CONIFER_ADDR ((void*)(START_OF_USER_FLASH + CONIFER_MGROUP*256 + CONIFER_OFFSET))

- #define DAQ_HEAD_MGROUP 6
- #define DAQ_HEAD_OFFSET 128
- #define DAQ_HEAD_ADDR ((daq_loop_args*)(START_OF_USER_FLASH + DAQ_HEAD_MGROUP*256 + DAQ_HEAD_OFFSET))
- #define DAQ_PARAMS1_MGROUP 7
- #define DAQ_PARAMS1_OFFSET 0
- #define DAQ_PARAMS1_ADDR ((void*)(START_OF_USER_FLASH + DAQ_PARAMS1_MGROUP*256 + DAQ_PARAMS1_OFFSET))
- #define DAQ_PARAMS2_MGROUP 8
- #define DAQ_PARAMS2_OFFSET 0
- #define DAQ_PARAMS2_ADDR
- #define DAQ_PARAMS3_MGROUP 9
- #define DAQ_PARAMS3_OFFSET 0
- #define DAQ_PARAMS3_ADDR
- #define PERSISTENT_DATA_MGROUP 13
- #define CRC_MGROUP1 14
- #define CRC_MGROUP2 15
- #define SETTING_BRANCH_SIZE (256*10)
- #define CRC_POLY 0x04C11DB7
- #define MAGIC_NUMBER 0xEFBEADDE

Typedefs

- typedef struct veh_gen_info **veh_gen_info**
- typedef struct uv_vehicle_settings **uv_vehicle_settings**

Enumerations

- enum **laptop_CMD** {

LAPTOP_HANDSHAKE = 0x01 , ENTER_PROGRAMMING_MODE = 0x02 , REQUEST_VCU_STATUS = 0x03 ,

CLEAR_FAULTS = 0x04 ,

REQUEST_VCU_FIRMWARE_VERSION = 0x05 , GENERIC_ACK = 0x10 , CANNOT_PERFORM_REQUEST = 0x11 ,

SET_SPECIFIC_PARAM = 0x20 ,

END_OF_SPECIFIC_PARAMS = 0x3F , REQUEST_ALL_SETTINGS = 0x40 , REQUEST_ALL_JOURNAL_ENTRIES = 0x41 ,

REQUEST_JOURNAL_ENTRIES_BY_TIME = 0x42 ,

REQUEST_SPECIFIC_SETTING = 0x43 , SAVE_AND_APPLY_NEW_SETTINGS = 0x80 , DISCARD_NEW_SETTINGS = 0x82 ,

DISCARD_NEW_SETTINGS_AND_EXIT = 0x83 ,

FORCE_RESTORE_FACTORY_DEFAULT = 0x84 , REQUEST_PEDAL_CALIBRATION = 0x90 ,

ADVANCE_CALIBRATION_SEQ = 0x91 , ABORT_CALIBRATION_SEQ = 0x92 ,

FINISH_CALIBRATION_SEQ = 0x93 }

Functions

- **uv_status uvConfigSettingTask (void *args)**

Function to setup the parameters of the setting setter task.
- **void nukeSettings (uv_vehicle_settings **settings_to_delete)**
- **uv_status uvValidateSettingsFromFlash ()**
- enum **uv_status_t uvSettingsInit ()**

this function does one thing, and one thing only, it checks if we have custom settings, then it attempts to get them. If it fails, then we revert to factory defaults.
- **uv_status setupDefaultSettings ()**

Function that allocates the neccessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.
- **uv_status uvSaveSettingsToFlash ()**
- **uv_status uvComputeMemRegionChecksum (void *sblock, uint32_t *csums, char mregion)**
- **uv_status uvComputeSettingsCheckSums (void *sblock)**
- **uv_status uvValidateChecksums (void *sblock)**

Variables

- `uv_vehicle_settings * current_vehicle_settings`

7.47.1 Macro Definition Documentation

7.47.1.1 BMS_ADDR

```
#define BMS_ADDR NULL
```

Definition at line 67 of file [uvfr_settings.h](#).

Referenced by [uvLoadSettingsFromFlash\(\)](#).

7.47.1.2 BMS_MGROUP

```
#define BMS_MGROUP 4
```

Definition at line 65 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.3 BMS_OFFSET

```
#define BMS_OFFSET 0
```

Definition at line 66 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.4 CONIFER_ADDR

```
#define CONIFER_ADDR ((void*) (START_OF_USER_FLASH + CONIFER_MGROUP*256 + CONIFER_OFFSET))
```

Definition at line 75 of file [uvfr_settings.h](#).

7.47.1.5 CONIFER_MGROUP

```
#define CONIFER_MGROUP 5
```

Definition at line 73 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.6 CONIFER_OFFSET

```
#define CONIFER_OFFSET 0
```

Definition at line 74 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.7 CRC_MGROUP1

```
#define CRC_MGROUP1 14
```

Definition at line 94 of file [uvfr_settings.h](#).

7.47.1.8 CRC_MGROUP2

```
#define CRC_MGROUP2 15
```

Definition at line 95 of file [uvfr_settings.h](#).

7.47.1.9 CRC_POLY

```
#define CRC_POLY 0x04C11DB7
```

Definition at line 100 of file [uvfr_settings.h](#).

Referenced by [uvComputeChunkChecksum\(\)](#).

7.47.1.10 DAQ_HEAD_ADDR

```
#define DAQ_HEAD_ADDR ((daq_loop_args*) (START_OF_USER_FLASH + DAQ_HEAD_MGROUP*256 + DAQ_HEAD_OFFSET))
```

Definition at line 79 of file [uvfr_settings.h](#).

Referenced by [uvLoadSettingsFromFlash\(\)](#).

7.47.1.11 DAQ_HEAD_MGROUP

```
#define DAQ_HEAD_MGROUP 6
```

Definition at line 77 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.12 DAQ_HEAD_OFFSET

```
#define DAQ_HEAD_OFFSET 128
```

Definition at line 78 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.13 DAQ_PARAMS1_ADDR

```
#define DAQ_PARAMS1_ADDR ((void*) (START_OF_USER_FLASH + DAQ_PARAMS1_MGROUP*256 + DAQ_PARAMS1_OFFSET))
```

Definition at line 83 of file [uvfr_settings.h](#).

Referenced by [uvLoadSettingsFromFlash\(\)](#).

7.47.1.14 DAQ_PARAMS1_MGROUP

```
#define DAQ_PARAMS1_MGROUP 7
```

Definition at line 81 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.15 DAQ_PARAMS1_OFFSET

```
#define DAQ_PARAMS1_OFFSET 0
```

Definition at line 82 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.16 DAQ_PARAMS2_ADDR

```
#define DAQ_PARAMS2_ADDR
```

Definition at line 87 of file [uvfr_settings.h](#).

7.47.1.17 DAQ_PARAMS2_MGROUP

```
#define DAQ_PARAMS2_MGROUP 8
```

Definition at line 85 of file [uvfr_settings.h](#).

7.47.1.18 DAQ_PARAMS2_OFFSET

```
#define DAQ_PARAMS2_OFFSET 0
```

Definition at line 86 of file [uvfr_settings.h](#).

7.47.1.19 DAQ_PARAMS3_ADDR

```
#define DAQ_PARAMS3_ADDR
```

Definition at line 91 of file [uvfr_settings.h](#).

7.47.1.20 DAQ_PARAMS3_MGROUP

```
#define DAQ_PARAMS3_MGROUP 9
```

Definition at line 89 of file [uvfr_settings.h](#).

7.47.1.21 DAQ_PARAMS3_OFFSET

```
#define DAQ_PARAMS3_OFFSET 0
```

Definition at line 90 of file [uvfr_settings.h](#).

7.47.1.22 DRIVING_ADDR

```
#define DRIVING_ADDR ((driving_loop_args*) (START_OF_USER_FLASH + DRIVING_MGROUP + DRIVING_OFFSET))
```

Definition at line 63 of file [uvfr_settings.h](#).

7.47.1.23 DRIVING_MGROUP

```
#define DRIVING_MGROUP 2
```

Definition at line 61 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.24 DRIVING_OFFSET

```
#define DRIVING_OFFSET 0
```

Definition at line 62 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.25 ENABLE_FLASH_SETTINGS

```
#define ENABLE_FLASH_SETTINGS 0
```

Definition at line 21 of file [uvfr_settings.h](#).

7.47.1.26 FLASH_SBLOCK_START

```
#define FLASH_SBLOCK_START START_OF_USER_FLASH
```

Definition at line 41 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvSaveSettingsToFlash\(\)](#), [uvSBlockCorruptionHandler\(\)](#), and [uvSettingsProgrammerTask\(\)](#).

7.47.1.27 GENERAL_VEH_INFO_MGROUP

```
#define GENERAL_VEH_INFO_MGROUP 0
```

Definition at line 47 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

7.47.1.28 GENERAL_VEH_INFO_OFFSET

```
#define GENERAL_VEH_INFO_OFFSET 32
```

Definition at line 48 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.29 IMD_ADDR

```
#define IMD_ADDR ((void*) (START_OF_USER_FLASH + IMD_MGROUP*256 + IMD_OFFSET))
```

Definition at line 71 of file [uvfr_settings.h](#).

Referenced by [uvLoadSettingsFromFlash\(\)](#).

7.47.1.30 IMD_MGROUP

```
#define IMD_MGROUP 4
```

Definition at line 69 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.31 IMD_OFFSET

```
#define IMD_OFFSET 128
```

Definition at line 70 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.32 isPtrToFlash

```
#define isPtrToFlash(  
    p)  
  
Value:  
( (p>=0x08000000) && (p<TOP\_OF\_USER\_FLASH) )
```

Definition at line [44](#) of file [uvfr_settings.h](#).

7.47.1.33 MAGIC_NUMBER

```
#define MAGIC_NUMBER 0xEFBEADDE
```

Definition at line [107](#) of file [uvfr_settings.h](#).

Referenced by [uvResetFlashToDefault\(\)](#), [uvSaveSettingsToFlash\(\)](#), and [uvValidateFlashSettings\(\)](#).

7.47.1.34 MOTOR_ADDR

```
#define MOTOR_ADDR (struct motor\_controller\_settings*) (START\_OF\_USER\_FLASH + MOTOR\_MGROUP*256 +  
MOTOR\_OFFSET)
```

Definition at line [58](#) of file [uvfr_settings.h](#).

7.47.1.35 MOTOR_MGROUP

```
#define MOTOR_MGROUP 1
```

Definition at line [55](#) of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.36 MOTOR_OFFSET

```
#define MOTOR_OFFSET 0
```

Definition at line [56](#) of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.37 OS_SETTINGS_ADDR

```
#define OS_SETTINGS_ADDR ((void*) (START\_OF\_USER\_FLASH + OS\_SETTINGS\_MGROUP*256 + OS\_SETTINGS\_OFFSET))
```

Definition at line [53](#) of file [uvfr_settings.h](#).

7.47.1.38 OS_SETTINGS_MGROUP

```
#define OS_SETTINGS_MGROUP 0
```

Definition at line 50 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), and [uvLoadSettingsFromFlash\(\)](#).

7.47.1.39 OS_SETTINGS_OFFSET

```
#define OS_SETTINGS_OFFSET 128
```

Definition at line 51 of file [uvfr_settings.h](#).

Referenced by [sendAllSettingsWorker\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.47.1.40 PERSISTENT_DATA_MGROUP

```
#define PERSISTENT_DATA_MGROUP 13
```

Definition at line 93 of file [uvfr_settings.h](#).

7.47.1.41 SBLOCK_CRC_REGION_OFFSET

```
#define SBLOCK_CRC_REGION_OFFSET
```

Definition at line 34 of file [uvfr_settings.h](#).

7.47.1.42 SBLOCK_CSR_OFFSET

```
#define SBLOCK_CSR_OFFSET 0x0C00
```

Definition at line 31 of file [uvfr_settings.h](#).

7.47.1.43 SETTING_BRANCH_SIZE

```
#define SETTING_BRANCH_SIZE (256*10)
```

Definition at line 98 of file [uvfr_settings.h](#).

Referenced by [uvCreateTmpSettingsCopy\(\)](#), [uvResetFlashToDefault\(\)](#), and [uvSettingsInit\(\)](#).

7.47.1.44 SIZE_OF_MGROUP

```
#define SIZE_OF_MGROUP 0x0100
```

Definition at line 33 of file [uvfr_settings.h](#).

7.47.1.45 SIZEOF_SBLOCK

```
#define SIZEOF_SBLOCK 4096
```

Definition at line 39 of file [uvfr_settings.h](#).

7.47.1.46 SIZEOF_USER_FLASH

```
#define SIZEOF_USER_FLASH 4096
```

Definition at line 37 of file [uvfr_settings.h](#).

7.47.1.47 START_OF_USER_FLASH

```
#define START_OF_USER_FLASH &_s_uvdata
```

Definition at line 27 of file [uvfr_settings.h](#).

Referenced by [uvCreateTmpSettingsCopy\(\)](#), [uvForceDefaultReversionUponDeviceReset\(\)](#), [uvLoadSettingsFromFlash\(\)](#), [uvSaveSettingsToFlash\(\)](#), [uvSendSpecificParam\(\)](#), [uvSetSettingResponseReminder\(\)](#), [uvSettingsInit\(\)](#), and [uvValidateFlashSettings\(\)](#).

7.47.1.48 SYS_DATA_OFFSET

```
#define SYS_DATA_OFFSET 0x00
```

Definition at line 32 of file [uvfr_settings.h](#).

7.47.1.49 TOP_OF_FLASH_SBLOCK

```
#define TOP_OF_FLASH_SBLOCK (void*) 0x080FFFFF
```

Definition at line 30 of file [uvfr_settings.h](#).

Referenced by [uvSaveSettingsToFlash\(\)](#).

7.47.1.50 TOP_OF_USER_FLASH

```
#define TOP_OF_USER_FLASH &_e_uvdata
```

Definition at line 28 of file [uvfr_settings.h](#).

7.47.2 Typedef Documentation

7.47.2.1 uv_vehicle_settings

```
typedef struct uv_vehicle_settings uv_vehicle_settings
```

7.47.2.2 veh_gen_info

```
typedef struct veh_gen_info veh_gen_info
```

7.47.3 Enumeration Type Documentation

7.47.3.1 laptop_CMD

```
enum laptop_CMD
```

Enumerator

LAPTOP_HANDSHAKE
ENTER_PROGRAMMING_MODE
REQUEST_VCU_STATUS
CLEARFAULTS
REQUEST_VCU_FIRMWARE_VERSION
GENERIC_ACK
CANNOT_PERFORM_REQUEST
SET_SPECIFIC_PARAM
END_OF_SPECIFIC_PARAMS
REQUEST_ALL_SETTINGS
REQUEST_ALL_JOURNAL_ENTRIES
REQUEST_JOURNAL_ENTRIES_BY_TIME
REQUEST_SPECIFIC_SETTING
SAVE_AND_APPLY_NEW_SETTINGS
DISCARD_NEW_SETTINGS
DISCARD_NEW_SETTINGS_AND_EXIT
FORCE_RESTORE_FACTORY_DEFAULT
REQUEST_PEDAL_CALIBRATION
ADVANCE_CALIBRATION_SEQ
ABORT_CALIBRATION_SEQ
FINISH_CALIBRATION_SEQ

Definition at line 126 of file [uvfr_settings.h](#).

7.47.4 Function Documentation

7.47.4.1 nukeSettings()

```
void nukeSettings (
    uv_vehicle_settings ** settings_to_delete)
```

Definition at line 280 of file [uvfr_settings.c](#).

7.47.4.2 setupDefaultSettings()

```
uv_status setupDefaultSettings ()
```

Function that allocates the neccessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.

Definition at line 250 of file [uvfr_settings.c](#).

References [uv_vehicle_settings::bms_settings](#), [uv_vehicle_settings::conifer_settings](#), [current_vehicle_settings](#), [uv_vehicle_settings::daq_param_list](#), [uv_vehicle_settings::daq_settings](#), [default_bms_settings](#), [default_conifer_settings](#), [default_daq_settings](#), [default_datapoints](#), [default_dl_settings](#), [default_os_settings](#), [uv_vehicle_settings::driving_loop_settings](#), [uv_vehicle_settings::flags](#), [uv_vehicle_settings::imd_settings](#), [mc_default_settings](#), [uv_vehicle_settings::mc_settings](#), [uv_vehicle_settings::os_settings](#), [UV_ERROR](#), and [UV_OK](#).

Referenced by [uvSettingsInit\(\)](#).

7.47.4.3 uvComputeMemRegionChecksum()

```
uv_status uvComputeMemRegionChecksum (
    void * sblock,
    uint32_t * csums,
    char mregion)
```

Definition at line 490 of file [uvfr_settings.c](#).

References [UV_OK](#), and [uvComputeChunkChecksum\(\)](#).

Referenced by [uvComputeSettingsChecksums\(\)](#).

Here is the call graph for this function:

7.47.4.4 uvComputeSettingsCheckSums()

```
uv_status uvComputeSettingsCheckSums (
    void * sblock)
```

7.47.4.5 uvConfigSettingTask()

```
uv_status uvConfigSettingTask (
    void * args)
```

Function to setup the parameters of the setting setter task.

Definition at line 328 of file [uvfr_settings.c](#).

References [ABOVE_NORMAL](#), [uv_task_info::active_states](#), [uv_task_info::deletion_states](#), [PROGRAMMING](#), [uv_task_info::stack_size](#), [uv_task_info::suspension_states](#), [uv_task_info::task_args](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [uv_task_info::task_period](#), [uv_task_info::task_priority](#), [UV_ERROR](#), [UV_OK](#), [uvCreateTask\(\)](#), and [uvSettingsProgrammerTask\(\)](#).

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.47.4.6 uvSaveSettingsToFlash()

```
uv_status uvSaveSettingsToFlash ()
```

Referenced by [uvResetFlashToDefault\(\)](#), and [uvSettingsProgrammerTask\(\)](#).

7.47.4.7 uvSettingsInit()

```
enum uv_status_t uvSettingsInit ()
```

this function does one thing, and one thing only, it checks if we have custom settings, then it attempts to get them. If it fails, then we revert to factory defaults.

Definition at line 359 of file [uvfr_settings.c](#).

References [__uvInitPanic\(\)](#), [CAN_BUS_1](#), [CAN_BUS_2](#), [current_vehicle_settings](#), [handleIncomingLaptopMsg\(\)](#), [insertCANMessageHandler\(\)](#), [SETTING_BRANCH_SIZE](#), [setupDefaultSettings\(\)](#), [START_OF_USER_FLASH](#), [UV_ERROR](#), [UV_OK](#), [uvLoadSettingsFromFlash\(\)](#), [uvResetFlashToDefault\(\)](#), [uvSendCanMSG\(\)](#), [uvValidateFlashSettings\(\)](#), and [vcu_ack_msg](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.47.4.8 uvValidateChecksums()

```
uv_status uvValidateChecksums (
    void * sblock)
```

Definition at line 522 of file [uvfr_settings.c](#).

References [UV_ERROR](#), [UV_OK](#), and [uvComputeSettingsChecksums\(\)](#).

Here is the call graph for this function:

7.47.4.9 uvValidateSettingsFromFlash()

```
uv_status uvValidateSettingsFromFlash ()
```

7.47.5 Variable Documentation

7.47.5.1 current_vehicle_settings

```
uv_vehicle_settings* current_vehicle_settings [extern]
```

Definition at line 25 of file [uvfr_settings.c](#).

Referenced by [configureDaqSubTasks\(\)](#), [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), [StartDrivingLoop\(\)](#), [uvInit\(\)](#), and [uvStartStateMachine\(\)](#).

7.48 uvfr_settings.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * uvfr_settings.h
00003 *
00004 * Created on: Oct 10, 2024
00005 * Author: byo10
00006 */
00007
00008 #ifndef INC_UVFR_SETTINGS_H_
00009 #define INC_UVFR_SETTINGS_H_
00010
00011
00012 #include "motor_controller.h"
00013 #include "driving_loop.h"
00014 #include "uvfr_utils.h"
00015 #include "main.h"
00016 #include "daq.h"
00017 #include "bms.h"
00018
00019
00020
00021 #define ENABLE_FLASH_SETTINGS 0
00022
00023
00024
00025 //This should nearly always work out to 0x080FE000, and goes 4 KB until it reaches the end of
00026 //the user flash region at 0xFFFFFFF
00027 #define START_OF_USER_FLASH &_s_uvdata
00028 #define TOP_OF_USER_FLASH &_e_uvdata
00029
00030 #define TOP_OF_FLASH_SBLOCK (void*)0x080FFFFF
00031 #define SBLOCK_CSR_OFFSET 0x0C00
00032 #define SYS_DATA_OFFSET 0x00
00033 #define SIZE_OF_MGROUP 0x0100
00034 #define SBLOCK_CRC_REGION_OFFSET // (START_OF_USER_FLASH - 0xFF)
00035
00036 //size of user flash in bytes
00037 #define SIZEOF_USER_FLASH 4096
00038
00039 #define SIZEOF_SBLOCK 4096
00040
00041 #define FLASH_SBLOCK_START START_OF_USER_FLASH
00042
00043
00044 #define isPtrToFlash(p) ((p>=0x08000000)&&(p<TOP_OF_USER_FLASH))
00045
00046 //Positions of different settings in SBLOCK
00047 #define GENERAL_VEH_INFO_MGROUP 0
00048 #define GENERAL_VEH_INFO_OFFSET 32 //Start at 32 since first 32 bytes reserved for settings integrity
checks
00049
00050 #define OS_SETTINGS_MGROUP 0
00051 #define OS_SETTINGS_OFFSET 128
00052
00053 #define OS_SETTINGS_ADDR ((void*)(START_OF_USER_FLASH + OS_SETTINGS_MGROUP*256 + OS_SETTINGS_OFFSET))
00054
00055 #define MOTOR_MGROUP 1
00056 #define MOTOR_OFFSET 0
00057
00058 #define MOTOR_ADDR (struct motor_controller_settings*) (START_OF_USER_FLASH + MOTOR_MGROUP*256 +
MOTOR_OFFSET)
00059
00060
00061 #define DRIVING_MGROUP 2
00062 #define DRIVING_OFFSET 0
00063 #define DRIVING_ADDR ((driving_loop_args*)(START_OF_USER_FLASH + DRIVING_MGROUP + DRIVING_OFFSET))
00064
00065 #define BMS_MGROUP 4
00066 #define BMS_OFFSET 0
00067 #define BMS_ADDR NULL
00068
00069 #define IMD_MGROUP 4
00070 #define IMD_OFFSET 128
00071 #define IMD_ADDR ((void*)(START_OF_USER_FLASH + IMD_MGROUP*256 + IMD_OFFSET))
00072
00073 #define CONIFER_MGROUP 5
00074 #define CONIFER_OFFSET 0
00075 #define CONIFER_ADDR ((void*)(START_OF_USER_FLASH + CONIFER_MGROUP*256 + CONIFER_OFFSET))
00076
00077 #define DAQ_HEAD_MGROUP 6
00078 #define DAQ_HEAD_OFFSET 128
00079 #define DAQ_HEAD_ADDR ((daq_loop_args*)(START_OF_USER_FLASH + DAQ_HEAD_MGROUP*256 + DAQ_HEAD_OFFSET))
00080

```

```

00081 #define DAQ_PARAMS1_MGROUP 7
00082 #define DAQ_PARAMS1_OFFSET 0
00083 #define DAQ_PARAMS1_ADDR ((void*) (START_OF_USER_FLASH + DAQ_PARAMS1_MGROUP*256 + DAQ_PARAMS1_OFFSET))
00084
00085 #define DAQ_PARAMS2_MGROUP 8
00086 #define DAQ_PARAMS2_OFFSET 0
00087 #define DAQ_PARAMS2_ADDR
00088
00089 #define DAQ_PARAMS3_MGROUP 9
00090 #define DAQ_PARAMS3_OFFSET 0
00091 #define DAQ_PARAMS3_ADDR
00092
00093 #define PERSISTENT_DATA_MGROUP 13
00094 #define CRC_MGROUP1 14
00095 #define CRC_MGROUP2 15
00096
00097 //This below macro lowkey confusing ngl, It seems as though there is in fact 4kB reserved, so why th
00098 #define SETTING_BRANCH_SIZE (256*10)
00099
00100 #define CRC_POLY 0x04C11DB7
00101 //CRC-32 (POSIX Checksum)
00102
00103
00104 //Positions of different settings in SBLOCK
00105
00106 //Because this is little endian, it looks like DEADBEEF in the memory viewer
00107 #define MAGIC_NUMBER 0xEFBEADDE
00108
00109
00110 //AAA
00111 typedef struct veh_gen_info{
00112     uint32_t wheel_size;
00113     uint32_t drive_ratio;
00114     uint16_t test1;
00115     uint16_t test2;
00116     uint16_t test3;
00117     uint8_t test4;
00118     uint8_t test5;
00119
00120     uint32_t test6;
00121
00122 }veh_gen_info;
00123
00124 typedef enum uv_status_t uv_status;
00125
00126 typedef enum{
00127
00128     LAPTOP_HANDSHAKE = 0x01,
00129     ENTER_PROGRAMMING_MODE = 0x02,
00130     REQUEST_VCU_STATUS = 0x03,
00131     CLEAR_FAULTS = 0x04,
00132     REQUEST_VCU_FIRMWARE_VERSION = 0x05,
00133     GENERIC_ACK = 0x10,
00134     CANNOT_PERFORM_REQUEST = 0x11,
00135     SET_SPECIFIC_PARAM = 0x20,
00136     END_OF_SPECIFIC_PARAMS = 0x3F,
00137     REQUEST_ALL_SETTINGS = 0x40,
00138     REQUEST_ALL_JOURNAL_ENTRIES = 0x41,
00139     REQUEST_JOURNAL_ENTRIES_BY_TIME = 0x42,
00140     REQUEST_SPECIFIC_SETTING = 0x43,
00141     SAVE_AND_APPLY_NEW_SETTINGS = 0x80,
00142     DISCARD_NEW_SETTINGS = 0x82,
00143     DISCARD_NEW_SETTINGS_AND_EXIT = 0x83,
00144     FORCE_RESTORE_FACTORY_DEFAULT = 0x84,
00145     REQUEST_PEDAL_CALIBRATION = 0x90,
00146     ADVANCE_CALIBRATION_SEQ = 0x91,
00147     ABORT_CALIBRATION_SEQ = 0x92,
00148     FINISH_CALIBRATION_SEQ = 0x93
00149
00150 }laptop_CMD;
00151
00152 typedef struct uv_vehicle_settings{
00153     struct veh_gen_info* veh_info;
00154
00155     struct uv_os_settings* os_settings;
00156     struct motor_controller_settings* mc_settings;
00157
00158     driving_loop_args* driving_loop_settings;
00159
00160     struct uv_imd_settings* imd_settings;
00161     bms_settings_t* bms_settings;
00162
00163     daq_loop_args* daq_settings;
00164
00165     daq_msg* daq_param_list;
00166
00167

```

```

00168     struct conifer_settings* conifer_settings;
00169     //struct motor_controller_settings motor_controller_settings;
00170
00171     uint16_t flags;
00174 }uv_vehicle_settings;
00175
00176 //typedef struct motor_controller_settings motor_controller_settings;
00177
00178 typedef struct motor_controller_settings{
00179     //firmware version
00180     uint32_t can_id_tx;
00181     uint32_t can_id_rx;
00182     uint32_t mc_CAN_timeout;
00183     uint8_t proportional_gain;
00184
00185     uint32_t integral_time_constant;
00186     uint8_t integral_memory_max;
00187     // extra
00188     uint16_t max_speed;    // e.g., RPM in register units (0x34)
00189     uint16_t max_current; // e.g., 0x4D
00190     uint16_t cont_current; // e.g., 0x4E
00191     uint16_t max_torque;   // if using 0x90 or similar torque command
00192     uint16_t max_motor_temp; //max motor temp
00193     uint16_t warning_motor_temp; //trigger point for motor temp
00194
00195     uint8_t mc_bus;
00196 }motor_controller_settings;
00199
00200
00201
00202 //typedef struct motor_controller_settings motor_controller_settings;
00203
00204
00205
00206
00207 uv_status uvConfigSettingTask(void* args);
00208
00209 void nukeSettings(uv_vehicle_settings** settings_to_delete);
00210 uv_status uvValidateSettingsFromFlash();
00211
00212 enum uv_status_t uvSettingsInit();
00213 uv_status setupDefaultSettings();
00214 uv_status uvSaveSettingsToFlash();
00215 uv_status uvComputeMemRegionChecksum(void* sblock, uint32_t* csums, char mregion);
00216 uv_status uvComputeSettingsCheckSums(void* sblock);
00217 uv_status uvValidateChecksums(void* sblock);
00218
00219
00220 #ifndef SRC_UVFR_SETTINGS_C_
00221 //extern includes
00222
00223 extern uv_vehicle_settings* current_vehicle_settings;
00224
00225 #endif
00226
00227
00228 #endif /* INC_UVFR_SETTINGS_H_ */

```

7.49 Core/lnc/uvfr_state_engine.h File Reference

```

#include "uvfr_utils.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "cmsis_os.h"
#include "message_buffer.h"

```

Include dependency graph for uvfr_state_engine.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [uv_scd_response](#)

- struct `task_management_info`
Struct to contain data about a parent task.
- struct `task_status_block`
Information about the task.
- struct `uv_os_settings`
Settings that dictate state engine behavior.
- struct `uv_task_info`
This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine.

Macros

- `#define _UV_DEFAULT_TASK_INSTANCES` 128
- `#define _UV_DEFAULT_TASK_STACK_SIZE` 128
- `#define _UV_DEFAULT_TASK_PERIOD` 100
- `#define _UV_MIN_TASK_PERIOD` 5
- `#define _LONGEST_SC_TIME` 300
- `#define _SC_DAEMON_PERIOD` 10
- `#define SVC_TASK_MAX_CHECKIN_PERIOD` 500
- `#define UV_TASK_VEHICLE_APPLICATION` `0x0001U<<(0)`
- `#define UV_TASK_PERIODIC_SVC` `0x0001U<<(1)`
- `#define UV_TASK_DORMANT_SVC` `0b0000000000000000000000011`
- `#define UV_TASK_GENERIC_SVC` `0x0001U<<(2)`
- `#define UV_TASK_MANAGER_MASK` `0b0000000000000000000000011`
- `#define UV_TASK_LOG_START_STOP_TIME` `0x0001U<<(2)`
- `#define UV_TASK_LOG_MEM_USAGE` `0x0001U<<(3)`
- `#define UV_TASK_SCD_IGNORE` `0x0001U<<(4)`
- `#define UV_TASK_IS_PARENT` `0x0001U<<(5)`
- `#define UV_TASK_IS_CHILD` `0x0001U<<(6)`
- `#define UV_TASK_IS_ORPHAN` `0x0001U<<(7)`
- `#define UV_TASK_ERR_IN_CHILD` `0x0001U<<(8)`
- `#define UV_TASK_AWAITING_DELETION` `0x0001U<<(9)`
- `#define UV_TASK_DEFER_DELETION` `0x0001U<<(10)`
- `#define UV_TASK_DEADLINE_NOT_ENFORCED` `0x00`
- `#define UV_TASK_PRIO_INCREMENTATION` `0x0001U<<(11)`
- `#define UV_TASK_DEADLINE_FIRM` `0x0001U<<(12)`
- `#define UV_TASK_DEADLINE_HARD` `(0x0001U<<(11)|0x0001U<<(12))`
- `#define UV_TASK_DEADLINE_MASK` `(0x0001U<<(11)|0x0001U<<(12))`
- `#define UV_TASK_MISSION_CRITICAL` `0x0001U<<(13)`
- `#define UV_TASK_DELAYING` `0x0001U<<(14)`
- `#define uvTaskSetDeletionBit(t)`
- `#define uvTaskResetDeletionBit(t)`
- `#define uvTaskSetDelayBit(t)`
- `#define uvTaskResetDelayBit(t)`
- `#define uvTaskIsDelaying(t)`
- `#define uvTaskDelay(x, t)`
State engine aware vTaskDelay wrapper.
- `#define uvTaskDelayUntil(x, lasttim, per)`
State engine aware vTaskDelayUntil wrapper.

Typedefs

- `typedef uint8_t uv_task_id`
- `typedef uint32_t uv_timespan_ms`
- `typedef enum uv_vehicle_state_t uv_vehicle_state`

Type representing the overall state and operating mode of the vehicle.
- `typedef enum uv_task_cmd_e uv_task_cmd`

Special commands used to start and shutdown tasks.
- `typedef struct uv_scd_response uv_scd_response`
- `typedef enum uv_task_state_t uv_task_status`

Enum representing the state of a managed task.
- `typedef enum task_priority task_priority`

Priority of a managed task. Maps directly to OS priority.
- `typedef struct task_management_info task_management_info`

Struct to contain data about a parent task.
- `typedef struct task_status_block task_status_block`

Information about the task.
- `typedef struct uv_os_settings uv_os_settings`

Settings that dictate state engine behavior.
- `typedef struct uv_task_info uv_task_info`

This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine.

Enumerations

- `enum uv_vehicle_state_t {`
 - `UV_INIT = 0x0001 , UV_READY = 0x0002 , PROGRAMMING = 0x0004 , UV_DRIVING = 0x0008 ,`
 - `UV_SUSPENDED = 0x0010 , UV_LAUNCH_CONTROL = 0x0020 , UV_ERROR_STATE = 0x0040 ,`
 - `UV_BOOT = 0x0080 ,`
 - `UV_HALT = 0x0100 }`

Type representing the overall state and operating mode of the vehicle.
- `enum uv_task_cmd_e { UV_NO_CMD , UV_KILL_CMD , UV_SUSPEND_CMD , UV_TASK_START_CMD }`

Special commands used to start and shutdown tasks.
- `enum uv_scd_response_e {`
 - `UV_SUCCESSFUL_DELETION , UV_SUCCESSFUL_SUSPENSION , UV_COULDNT_DELETE ,`
 - `UV_COULDNT_SUSPEND ,`
 - `UV_UNSAFE_STATE }`

Response from a task confirming it has been either deleted or suspended.
- `enum uv_task_state_t { UV_TASK_NOT_STARTED , UV_TASK_DELETED , UV_TASK_RUNNING , UV_TASK_SUSPENDED }`

Enum representing the state of a managed task.
- `enum task_priority {`
 - `IDLE_TASK_PRIORITY , LOW_PRIORITY , BELOW_NORMAL , MEDIUM_PRIORITY ,`
 - `ABOVE_NORMAL , HIGH_PRIORITY , REALTIME_PRIORITY }`

Priority of a managed task. Maps directly to OS priority.
- `enum os_flag { UV_OS_LOG_MEM = 0x01 , UV_OS_LOG_TASK_END_TIME = 0x02 , UV_OS_ATTEMPT_RESTART_NC_TASK = 0x04 , UV_OS_ENABLE_NONCRIT_TASK_THROTTLE = 0x08 }`

Functions

- void [uvTaskPeriodEnd \(uv_task_info *t\)](#)
Function called at the end of the task period.
- struct [uv_task_info * uvCreateTask \(\)](#)
This function gets called when you want to create a task, and register it with the task register. Theres some gnarlyness here, but not unacceptable levels. Pray this thing doesn't hang itself.
- struct [uv_task_info * uvCreateServiceTask \(\)](#)
Create a new service task, because fuck you, thats why.
- struct [uv_task_info * uvGetTaskByld \(uint8_t id\)](#)
- [uv_status _uvValidateSpecificTask \(uint8_t id\)](#)
make sure the parameters of a task_info struct is valid
- [uv_status uvValidateManagedTasks \(\)](#)
ensure that all the tasks people have created actually make sense, and are valid
- [uv_status uvStartTask \(uint32_t *tracker, struct uv_task_info *t\)](#)
: This is a function that starts tasks which are already registered in the system
- [uv_status uvRegisterTask \(\)](#)
- [uv_status uvInitStateEngine \(\)](#)
Function that prepares the state engine to do its thing.
- [uv_status uvStartStateMachine \(\)](#)
Actually starts up the state engine to do state engine things.
- [uv_status uvDeleteTask \(uint32_t *tracker, struct uv_task_info *t\)](#)
deletes a managed task via the system
- [uv_status uvSuspendTask \(uint32_t *tracker, struct uv_task_info *t\)](#)
function to suspend one of the managed tasks.
- [uv_status uvDelInitStateEngine \(\)](#)
Stops and frees all resources used by uvfr_state_engine.
- [uv_status updateRunningTasks \(\)](#)
- [uv_status changeVehicleState \(uint16_t state\)](#)
Function for changing the state of the vehicle, as well as the list of active + inactive tasks.
- void [__uvPanic \(char *msg, uint8_t msg_len, const char *file, const int line, const char *func\)](#)
- void [killSelf \(struct uv_task_info *t\)](#)
This function is called by a task to nuke itself. Is a wrapper function that is used to do all the different things.
- void [suspendSelf \(struct uv_task_info *t\)](#)
Called by a task that needs to suspend itself, once the task has determined it is safe to do so.
- [uv_task_id getSVCTaskID \(char *tsk_name\)](#)

Variables

- enum [uv_vehicle_state_t vehicle_state](#)

7.49.1 Macro Definition Documentation

7.49.1.1 _LONGEST_SC_TIME

```
#define _LONGEST_SC_TIME 300
```

Definition at line 70 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#).

7.49.1.2 _SC_DAEMON_PERIOD

```
#define _SC_DAEMON_PERIOD 10
```

Definition at line 71 of file [uvfr_state_engine.h](#).

Referenced by [_stateChangeDaemon\(\)](#).

7.49.1.3 _UV_DEFAULT_TASK_INSTANCES

```
#define _UV_DEFAULT_TASK_INSTANCES 128
```

Definition at line 63 of file [uvfr_state_engine.h](#).

7.49.1.4 _UV_DEFAULT_TASK_PERIOD

```
#define _UV_DEFAULT_TASK_PERIOD 100
```

Definition at line 67 of file [uvfr_state_engine.h](#).

7.49.1.5 _UV_DEFAULT_TASK_STACK_SIZE

```
#define _UV_DEFAULT_TASK_STACK_SIZE 128
```

Definition at line 65 of file [uvfr_state_engine.h](#).

Referenced by [initOdometer\(\)](#), [initTempMonitor\(\)](#), [uvCreateServiceTask\(\)](#), and [uvCreateTask\(\)](#).

7.49.1.6 _UV_MIN_TASK_PERIOD

```
#define _UV_MIN_TASK_PERIOD 5
```

Definition at line 68 of file [uvfr_state_engine.h](#).

7.49.1.7 SVC_TASK_MAX_CHECKIN_PERIOD

```
#define SVC_TASK_MAX_CHECKIN_PERIOD 500
```

Definition at line 73 of file [uvfr_state_engine.h](#).

7.49.2 Typedef Documentation

7.49.2.1 uv_task_id

```
typedef uint8_t uv_task_id
```

Definition at line 59 of file [uvfr_state_engine.h](#).

7.49.2.2 uv_timespan_ms

```
typedef uint32_t uv_timespan_ms
```

Definition at line 77 of file [uvfr_state_engine.h](#).

7.49.3 Function Documentation

7.49.3.1 __uvPanic()

```
void __uvPanic (
    char * msg,
    uint8_t msg_len,
    const char * file,
    const int line,
    const char * func)
```

7.49.3.2 getSVCTaskID()

```
uv_task_id getSVCTaskID (
    char * tsk_name)
```

7.49.3.3 updateRunningTasks()

```
uv_status updateRunningTasks ()
```

7.49.3.4 uvGetTaskById()

```
struct uv_task_info * uvGetTaskById (
    uint8_t id)
```

7.49.3.5 uvRegisterTask()

```
uv_status uvRegisterTask ()
```

7.50 uvfr_state_engine.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * uvfr_state_engine.h
00003  *
00004  * Created on: Oct 15, 2024
00005  * Author: byo10
00006 */
00007
00043 #ifndef INC_UVFR_STATE_ENGINE_H_
00044 #define INC_UVFR_STATE_ENGINE_H_
00045
00046
00047 #include "uvfr_utils.h"
00048 #include "FreeRTOS.h"
00049 #include "task.h"
00050 #include "queue.h"
00051 #include "semphr.h"
00052 #include "cmsis_os.h"
00053 #include "message_buffer.h"
00054
00055
00056
00057 //#include "uvfr_utils.h"
00058 typedef enum uv_status_t uv_status;
00059 typedef uint8_t uv_task_id; //WHY DO I NEED TO DO THIS STUPID REDEFINITION HERE
00060
00061
00062
00063 #define _UV_DEFAULT_TASK_INSTANCES 128
00064 //STACK size 0, means that it uses operating system defaults
00065 #define _UV_DEFAULT_TASK_STACK_SIZE 128
00066 //period of 100ms, aka 10Hz
00067 #define _UV_DEFAULT_TASK_PERIOD 100
00068 #define _UV_MIN_TASK_PERIOD 5
00069
00070 #define _LONGEST_SC_TIME 300
00071 #define _SC_DAEMON_PERIOD 10
00072
00073 #define SVC_TASK_MAX_CHECKIN_PERIOD 500
00074
00075 //typedef uint8_t uv_task_cmd;
00076
00077 typedef uint32_t uv_timespan_ms;
00078
00088 typedef enum uv_vehicle_state_t{
00089     UV_INIT = 0x0001,
00090     UV_READY = 0x0002,
00091     PROGRAMMING = 0x0004,
00092     UV_DRIVING = 0x0008,
00093     UV_SUSPENDED = 0x0010,
00094     UV_LAUNCH_CONTROL = 0x0020,
00095     UV_ERROR_STATE = 0x0040,
00096     UV_BOOT = 0x0080,
00097     UV_HALT = 0x0100
00098 }uv_vehicle_state;
00099
00103 typedef enum uv_task_cmd_e{
00104     UV_NO_CMD,
00105     UV_KILL_CMD,
00106     UV_SUSPEND_CMD,
00107     UV_TASK_START_CMD
00108 }uv_task_cmd;
00109
00113 enum uv_scd_response_e{
00114     UV_SUCCESSFUL_DELETION,
00115     UV_SUCCESSFUL_SUSPENSION,
00116     UV_COULDNT_DELETE,
00117     UV_COULDNT_SUSPEND,
00118     UV_UNSAFE_STATE
00119 };
00120
00121 typedef struct uv_scd_response{
00122     enum uv_scd_response_e response_val;
00123     uv_task_id meta_id;
00124 }uv_scd_response;
00125
00131 typedef enum uv_task_state_t{
00132     UV_TASK_NOT_STARTED,
00133     UV_TASK_DELETED,
00134     UV_TASK_RUNNING,
00135     UV_TASK_SUSPENDED
00136 } uv_task_status;
00137

```

```

00138
00142 typedef enum task_priority{
00143     IDLE_TASK_PRIORITY,
00144     LOW_PRIORITY,
00145     BELOW_NORMAL,
00146     MEDIUM_PRIORITY,
00147     ABOVE_NORMAL,
00148     HIGH_PRIORITY,
00149     REALTIME_PRIORITY
00150 }task_priority;
00151
00152
00153
00161 typedef struct task_management_info{
00162     TaskHandle_t task_handle;
00163     QueueHandle_t parent_msg_queue;
00164 }task_management_info;
00165
00169 typedef struct task_status_block{
00170     uint32_t task_high_water_mark;
00171     TickType_t task_report_time;
00172 }task_status_block;
00173
00174 enum os_flag{
00175     UV_OS_LOG_MEM = 0x01,
00176     UV_OS_LOG_TASK_END_TIME = 0x02,
00177     UV_OS_ATTEMPT_RESTART_NC_TASK = 0x04,
00178     UV_OS_ENABLE_NONCRIT_TASK_THROTTLE = 0x08
00179 };
00180
00184 typedef struct uv_os_settings{
00185     TickType_t svc_task_manager_period;
00186     TickType_t task_manager_period;
00187     TickType_t max_svc_task_period;
00188     TickType_t max_task_period; //fuckin lethal man
00189     TickType_t min_task_period;
00190     float task_overshoot_margin_noncrit;
00191     float task_overshoot_margin_crit;
00192     float task_throttle_increment;
00193
00194     uint16_t os_flags;
00195
00196 }uv_os_settings;
00197
00198
00199 #define UV_TASK_VEHICLE_APPLICATION 0x0001U<(0)
00200 #define UV_TASK_PERIODIC_SVC 0x0001U<(1)
00201 #define UV_TASK_DORMANT_SVC 0b0000000000000011 //Bruh this syntax is wacko
00202 #define UV_TASK_GENERIC_SVC 0x0001U<(2)
00203 #define UV_TASK_MANAGER_MASK 0b0000000000000001
00204 #define UV_TASK_LOG_START_STOP_TIME 0x0001U<(2)
00205 #define UV_TASK_LOG_MEM_USAGE 0x0001U<(3)
00206 #define UV_TASK_SCD_IGNORE 0x0001U<(4)
00207 #define UV_TASK_IS_PARENT 0x0001U<(5)
00208 #define UV_TASK_IS_CHILD 0x0001U<(6)
00209 #define UV_TASK_IS_ORPHAN 0x0001U<(7)
00210 #define UV_TASK_ERR_IN_CHILD 0x0001U<(8)
00211 #define UV_TASK_AWAITING_DELETION 0x0001U<(9)
00212 #define UV_TASK_DEFER_DELETION 0x0001U<(10)
00213 #define UV_TASK_DEADLINE_NOT_ENFORCED 0x00 //TODO what the fuck is this piece of shit, empty macro??
HUUH???
00214 #define UV_TASK_PRIO_INCREMENTATION 0x0001U<(11)
00215 #define UV_TASK_DEADLINE_FIRM 0x0001U<(12)
00216 #define UV_TASK_DEADLINE_HARD (0x0001U<(11) | 0x0001U<(12))
00217 #define UV_TASK_DEADLINE_MASK (0x0001U<(11) | 0x0001U<(12))
00218 #define UV_TASK_MISSION_CRITICAL 0x0001U<(13)
00219 #define UV_TASK_DELAYING 0x0001U<(14)
00220
00221
00227 typedef struct uv_task_info{
00228     uv_task_id task_id;
00229     char* task_name;
00230     uv_timespan_ms task_period;
00231     uv_timespan_ms deletion_delay;
00232     TaskFunction_t task_function;
00233     osPriority task_priority;
00234     uint32_t stack_size;
00243     uv_task_status task_state; //tracks the internal state of the task
00244
00245
00246     TaskHandle_t task_handle;
00248     uv_task_cmd cmd_data;
00250     void* task_args;
00252     struct uv_task_info_t* parent;
00254     task_management_info* tmi;
00255     MessageBufferHandle_t task_rx_mailbox;
00256     TickType_t last_execution_time;

```

```

00257
00258     uint16_t active_states; //corresponds to the vehicle states where the task should be active
00259     uint16_t deletion_states; //corresponds to the vehicle states where the task should be suspended
00260     uint16_t suspension_states; //when should the task be suspended? When it should exist, but
00261     shouldnt be active.
00262     uint16_t task_flags;
00278     uint8_t throttle_factor;
00279 }uv_task_info;
00280
00281 #define uvTaskSetDeletionBit(t) (t->task_flags|=UV_TASK_AWAITING_DELETION)
00282 #define uvTaskResetDeletionBit(t) (t->task_flags &=(~UV_TASK_AWAITING_DELETION))
00283
00284 #define uvTaskSetDelayBit(t) (t->task_flags|=UV_TASK_DELAYING)
00285
00286 #define uvTaskResetDelayBit(t) (t->task_flags&=(~UV_TASK_DELAYING))
00287
00288 #define uvTaskIsDelaying(t) ((t->task_flags&UV_TASK_DELAYING)==UV_TASK_DELAYING)
00289
00290 #define uvTaskDelay(x,t) uvTaskSetDelayBit(x);\
00291     vTaskDelay(t);\
00292     uvTaskResetDelayBit(x)
00293
00294 void uvTaskPeriodEnd(uv_task_info* t);
00295
00296 #define uvTaskDelayUntil(x,lasttim,per) uvTaskSetDelayBit(x);\
00297     vTaskDelayUntil(&lasttim,per);\
00298     uvTaskResetDelayBit(x)
00299
00300 struct uv_task_info* uvCreateTask();
00301
00302 struct uv_task_info* uvCreateServiceTask();
00303
00304 struct uv_task_info* uvGetTaskById(uint8_t id);
00305
00306 uv_status _uvValidateSpecificTask(uint8_t id);
00307
00308 uv_status uvValidateManagedTasks();
00309
00310 uv_status uvStartTask(uint32_t* tracker,struct uv_task_info * t);
00311
00312 uv_status uvRegisterTask();
00313
00314 uv_status uvInitStateEngine();
00315
00316 uv_status uvStartStateMachine();
00317
00318 //static enum uv_status_t killEmAll();
00319
00320 uv_status uvDeleteTask(uint32_t* tracker,struct uv_task_info * t);
00321
00322 uv_status uvSuspendTask(uint32_t* tracker,struct uv_task_info * t);
00323
00324 uv_status uvDeInitStateEngine();
00325
00326 uv_status updateRunningTasks();
00327
00328 uv_status changeVehicleState(uint16_t state);
00329
00330 //void uvPanic(char* msg, uint8_t msg_len); //ruh roh scoobs, something has gone a little bit fucky
00331 wucky
00332 void __uvPanic(char* msg, uint8_t msg_len, const char* file, const int line, const char* func);
00333
00334 #ifndef uvPanic
00335 #define uvPanic(msg, errnum) __uvPanic(msg, errnum, __FILE__, __LINE__, __FUNCTION__)
00336 #endif
00337
00338 void killSelf(struct uv_task_info * t);
00339
00340 void suspendSelf(struct uv_task_info * t);
00341
00342 #ifndef UVFR_STATE_MACHINE_IMPLEMENTATION
00343
00344 //EXTERNAL VARIABLES
00345 extern enum uv_vehicle_state_t vehicle_state; //This is the one that ya'll are permitted to know
00346 #else
00347 //These shouold only be visible in the implemtation file
00348
00349 void _stateChangeDaemon(void * args);
00350
00351 void uvSVCTaskManager(void* args);
00352
00353 #endif
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386

```

```
00387 uv_task_id getSVCTaskID(char* tsk_name);
00388 #endif /* INC_UVFR_STATE_ENGINE_H_ */
00389
00390
```

7.51 Core/Inc/uvfr_utils.h File Reference

```
#include "uvfr_global_config.h"
#include "main.h"
#include "adc.h"
#include "can.h"
#include "dma.h"
#include "tim.h"
#include "gpio.h"
#include "uvfr_settings.h"
#include "uvfr_state_engine.h"
#include "uvfr_diagnostics.h"
#include "rb_tree.h"
#include "uvfr_vehicle_commands.h"
#include "bms.h"
#include "motor_controller.h"
#include "dash.h"
#include "imd.h"
#include "pdu.h"
#include "daq.h"
#include "uvfr_conifer.h"
#include "uvfr_vehicle_logger.h"
#include "driving_loop.h"
#include "temp_monitoring.h"
#include "odometer.h"
#include "FreeRTOSConfig.h"
#include "stdint.h"
#include <stdlib.h>
#include "cmsis_os.h"
#include "FreeRTOS.h"
#include "message_buffer.h"
#include "task.h"
```

Include dependency graph for uvfr_utils.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [uv_mutex_info](#)
- struct [uv_binary_semaphore_info](#)
- struct [uv_semaphore_info](#)
- union [access_control_info](#)
- struct [uv_CAN_msg](#)

Representative of a CAN message.

- struct [uv_init_struct](#)
- struct [uv_task_msg_t](#)

Struct containing a message between two tasks.

- struct [p_status](#)
- struct [uv_init_task_args](#)

Struct designed to act like the [uv_task_info](#) struct, but for the initialisation tasks. As a result it takes fewer arguments.

- struct [uv_internal_params](#)
Data used by the uvfr_utils library to do what it needs to do :)
- struct [uv_init_task_response](#)
Struct representing the response of one of the initialization tasks.

Macros

- `#define _BV(x)`
- `#define _BV_8(x)`
- `#define _BV_16(x)`
- `#define _BV_32(x)`
- `#define endianSwap(x)`
- `#define endianSwap8(x)`
- `#define endianSwap16(x)`
- `#define endianSwap32(x)`
- `#define deserializeSmallE16(x, i)`
- `#define deserializeSmallE32(x, i)`
- `#define deserializeBigE16(x, i)`
- `#define deserializeBigE32(x, i)`
- `#define serializeSmallE16(x, d, i)`
- `#define serializeSmallE32(x, d, i)`
- `#define serializeBigE16(x, d, i)`
- `#define serializeBigE32(x, d, i)`
- `#define setBits(x, msk, data)`
macro to set bits of an int without touching the ones we dont want to edit
- `#define isPowerOfTwo(x)`
Returns a truthy value if "x" is a power of two.
- `#define safePtrRead(x)`
lil treat to help us avoid the dreaded null pointer dereference
- `#define safePtrWrite(p, x)`
- `#define false 0`
- `#define true !false`
- `#define TEXTIFY(A)`
- `#define MAX_INIT_TIME 2500`
- `#define INIT_CHECK_PERIOD 100`
- `#define UV_CAN1`
- `#define UV_CAN2`
- `#define USE_OLED_DEBUG 1`
- `#define UV_CAN_EXTENDED_ID 0b00000001`
- `#define CAN_BUS_1 0b00000010`
- `#define CAN_BUS_2 0b00000100`
- `#define UV_CAN_CHANNEL_MASK 0b00000110`
- `#define UV_CAN_DYNAMIC_MEM 0b00001000`

TypeDefs

- `typedef uint8_t bool`
- `typedef uint8_t uv_ext_device_id`
- `typedef enum access_control_t access_control_type`
- `typedef enum uv_msg_type_t uv_msg_type`

Enum dictating the meaning of a generic message.
- `typedef union access_control_info access_control_info`
- `typedef struct uv_init_struct uv_init_struct`
- `typedef struct uv_task_msg_t uv_task_msg`

Struct containing a message between two tasks.
- `typedef struct p_status p_status`
- `typedef struct uv_init_task_args uv_init_task_args`

Struct designed to act like the `uv_task_info` struct, but for the initialisation tasks. As a result it takes fewer arguments.
- `typedef struct uv_internal_params uv_internal_params`

Data used by the uvfr_utils library to do what it needs to do :)
- `typedef struct uv_init_task_response uv_init_task_response`

Struct representing the response of one of the initialization tasks.

Enumerations

- `enum uv_status_t { UV_OK, UV_WARNING, UV_ERROR, UV_ABORTED }`

This is meant to be a return type from functions that indicates what is actually going on.
- `enum data_type {`
`UV_UINT8 = 0, UV_INT8 = 1, UV_UINT16 = 2, UV_INT16 = 3,`
`UV_UINT32 = 4, UV_INT32 = 5, UV_FLOAT = 6, UV_DOUBLE = 7,`
`UV_INT64 = 8, UV_UINT64 = 9, UV_STRING = 10 }`

Represents the data type of some variable.
- `enum uv_driving_mode_t { normal, accel, econ, limp }`
- `enum uv_external_device { MOTOR_CONTROLLER = 0, BMS = 1, IMD = 2, PDU = 3 }`

ID for external devices, which allows us to know what's good with them.
- `enum access_control_t {`
`UV_NONE, UV_DUMB_FLAG, UV_MUTEX, UV_BINARY_SEMAPHORE,`
`UV_SEMAPHORE }`
- `enum uv_msg_type_t {`
`UV_TASK_START_COMMAND, UV_TASK_DELETE_COMMAND, UV_TASK_SUSPEND_COMMAND,`
`UV_COMMAND_ACKNOWLEDGEMENT,`
`UV_TASK_STATUS_REPORT, UV_ERROR_REPORT, UV_WAKEUP, UV_PARAM_REQUEST,`
`UV_PARAM_READY, UV_RAW_DATA_TRANSFER, UV_SC_COMMAND, UV_INVALID_MSG,`
`UV_ASSIGN_TASK }`

Enum dictating the meaning of a generic message.

Functions

- `void uvInit (void *arguments)`

: Function that initializes all of the car's stuff.
- `void __uvInitPanic ()`

Low Level Panic, that does not require the full UVFR utils functionality to be operational.
- `uv_status uvIsPTRValid (void *ptr)`

function that checks to make sure a pointer points to a place it is allowed to point to

Variables

- `uv_internal_params global_context`
- `const uint8_t data_size []`

7.51.1 Detailed Description

Author

Byron Oser

Definition in file [uvfr_utils.h](#).

7.51.2 Macro Definition Documentation

7.51.2.1 CAN_BUS_1

```
#define CAN_BUS_1 0b00000010
```

Definition at line [271](#) of file [uvfr_utils.h](#).

Referenced by [BMS_Init\(\)](#), [CANbusTxSvcDaemon\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [insertCANMessageHandler\(\)](#), [nuke_hash_table\(\)](#), [tempMonitorTask\(\)](#), [testfunc3\(\)](#), and [uvSettingsInit\(\)](#).

7.51.2.2 CAN_BUS_2

```
#define CAN_BUS_2 0b00000100
```

Definition at line [272](#) of file [uvfr_utils.h](#).

Referenced by [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [nuke_hash_table\(\)](#), [tempMonitorTask\(\)](#), and [uvSettingsInit\(\)](#).

7.51.2.3 INIT_CHECK_PERIOD

```
#define INIT_CHECK_PERIOD 100
```

Definition at line [153](#) of file [uvfr_utils.h](#).

Referenced by [uvInit\(\)](#).

7.51.2.4 MAX_INIT_TIME

```
#define MAX_INIT_TIME 2500
```

Definition at line [152](#) of file [uvfr_utils.h](#).

Referenced by [uvInit\(\)](#).

7.51.2.5 USE_OLED_DEBUG

```
#define USE_OLED_DEBUG 1
```

Definition at line 164 of file [uvfr_utils.h](#).

7.51.2.6 UV_CAN1

```
#define UV_CAN1
```

Definition at line 160 of file [uvfr_utils.h](#).

7.51.2.7 UV_CAN2

```
#define UV_CAN2
```

Definition at line 161 of file [uvfr_utils.h](#).

7.51.2.8 UV_CAN_CHANNEL_MASK

```
#define UV_CAN_CHANNEL_MASK 0b00000110
```

Definition at line 275 of file [uvfr_utils.h](#).

7.51.2.9 UV_CAN_DYNAMIC_MEM

```
#define UV_CAN_DYNAMIC_MEM 0b00001000
```

Definition at line 276 of file [uvfr_utils.h](#).

7.51.2.10 UV_CAN_EXTENDED_ID

```
#define UV_CAN_EXTENDED_ID 0b00000001
```

Definition at line 269 of file [uvfr_utils.h](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), and [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#).

7.51.3 Typedef Documentation

7.51.3.1 access_control_info

```
typedef union access_control_info access_control_info
```

7.51.3.2 `access_control_type`

```
typedef enum access_control_t access_control_type
```

7.51.3.3 `bool`

```
typedef uint8_t bool
```

Definition at line 139 of file [uvfr_utils.h](#).

7.51.3.4 `p_status`

```
typedef struct p_status p_status
```

7.51.3.5 `uv_ext_device_id`

```
typedef uint8_t uv_ext_device_id
```

Definition at line 143 of file [uvfr_utils.h](#).

7.51.3.6 `uv_init_struct`

```
typedef struct uv_init_struct uv_init_struct
```

contains info relevant to initializing the vehicle

7.51.3.7 `uv_init_task_args`

```
typedef struct uv_init_task_args uv_init_task_args
```

Struct designed to act like the `uv_task_info` struct, but for the initialisation tasks. As a result it takes fewer arguments.

7.51.3.8 `uv_init_task_response`

```
typedef struct uv_init_task_response uv_init_task_response
```

Struct representing the response of one of the initialization tasks.

Is returned in the initialization queue, and is read by `uvInit()` to determine whether the initialization of the internal device has failed or succeeded.

7.51.3.9 uv_internal_params

```
typedef struct uv_internal_params uv_internal_params
```

Data used by the uvfr_utils library to do what it needs to do :)

This is a global variable that is initialized at some point at launch

7.51.3.10 uv_msg_type

```
typedef enum uv_msg_type_t uv_msg_type
```

Enum dictating the meaning of a generic message.

7.51.3.11 uv_task_msg

```
typedef struct uv_task_msg_t uv_task_msg
```

Struct containing a message between two tasks.

This is a generic type that is best used in situations where the message could mean a variety of different things. For niche applications or where efficiency is paramount, we recommend creating a bespoke protocol.

7.51.4 Enumeration Type Documentation

7.51.4.1 access_control_t

```
enum access_control_t
```

Enumerator

UV_NONE	
UV_DUMB_FLAG	
UV_MUTEX	
UV_BINARY_SEMAPHORE	
UV_SEMAPHORE	

Definition at line 218 of file [uvfr_utils.h](#).

7.51.4.2 data_type

```
enum data_type
```

Represents the data type of some variable.

Enumerator

UV_UINT8	
UV_INT8	
UV_UINT16	
UV_INT16	
UV_UINT32	
UV_INT32	
UV_FLOAT	
UV_DOUBLE	
UV_INT64	
UV_UINT64	
UV_STRING	

Definition at line 184 of file [uvfr_utils.h](#).

7.51.4.3 uv_driving_mode_t

```
enum uv_driving_mode_t
```

Enumerator

normal	
accel	
econ	
limp	

Definition at line 201 of file [uvfr_utils.h](#).

7.51.4.4 uv_external_device

```
enum uv_external_device
```

ID for external devices, which allows us to know what's good with them.

Enumerator

MOTOR_CONTROLLER	
BMS	
IMD	
PDU	

Definition at line 211 of file [uvfr_utils.h](#).

7.51.4.5 uv_msg_type_t

```
enum uv_msg_type_t
```

Enum dictating the meaning of a generic message.

Enumerator

UV_TASK_START_COMMAND
UV_TASK_DELETE_COMMAND
UV_TASK_SUSPEND_COMMAND
UV_COMMAND_ACKNOWLEDGEMENT
UV_TASK_STATUS_REPORT
UV_ERROR_REPORT
UV_WAKEUP
UV_PARAM_REQUEST
UV_PARAM_READY
UV_RAW_DATA_TRANSFER
UV_SC_COMMAND
UV_INVALID_MSG
UV_ASSIGN_TASK

Definition at line 229 of file [uvfr_utils.h](#).

7.51.4.6 uv_status_t

```
enum uv_status_t
```

This is meant to be a return type from functions that indicates what is actually going on.

Use this as a return value for functions you want to know the success of. In general, any function you write must return something, as well as account for any possible errors that may have occurred.

Enumerator

UV_OK
UV_WARNING
UV_ERROR
UV_ABORTED

Definition at line 173 of file [uvfr_utils.h](#).

7.51.5 Function Documentation

7.51.5.1 __uvInitPanic()

```
void __uvInitPanic ()
```

Low Level Panic, that does not require the full UVFR utils functionality to be operational.

Attention

Calling `_uvInitPanic()` is irreversible and will cause the vehicle to hang itself. This is only to be used as a last resort to stop the vehicle from entering an invalid state.

Definition at line 357 of file [uvfr_utils.c](#).

Referenced by [uvInit\(\)](#), [uvInitStateEngine\(\)](#), [uvSettingsInit\(\)](#), and [uvSVCTaskManager\(\)](#).

7.51.5.2 uvInit()

```
void uvInit (
    void * arguments)
```

: Function that initializes all of the car's stuff.

This is an RTOS task, and it serves to setup all of the car's different functions. at this point in our execution, we have already initialized all of our favorite hardware peripherals using HAL. Now we get to configure our convoluted system of OS-level settings and state machines.

It executes the following functions, in order:

- Load Vehicle Settings
 - Initialize and Start State Machine
 - Start Service Tasks, such as CAN, ADC, etc...
 - Initialize External Devices such as BMS, IMD, Motor Controller
 - Validate that these devices have actually booted up
 - Set vehicle state to UV_READY
- Pretty important shit if you ask me.

First on the block is our settings. The uv_settings are a bit strange, in the following way. We will check if we have saved custom settings, or if these settings are the default or not. It will then perform a checksum on the settings, and validate them to ensure they are safe If it fails to validate the settings, it will attempt to return to factory default.

If it is unable to return even to factory default settings, then we are in HUGE trouble, and some catastrophic bug has occurred. If it fails to even start this, it will not be safe to drive We must therefore panic.

Once the settings are initialized, we will initialize the system diagnostics. This is done early, so that future errors will result in events being properly tracked and logged

The second thing initialized is internal diagnostics and telemetry.

Next up we will attempt to initialize the state engine. If this fails, then we are in another case where we are genuinely unsafe to drive. This will create the prototypes for a bajillion tasks that will be started and stopped. Which tasks are currently running, depends on the whims of the state engine. Since the state engine is critical to our ability to handle errors and implausibilities, we cannot proceed without a fully operational state engine.

Once the state machine is initialized we get to actually start the thing.

Once the state machine is initialized we get to actually start the thing.

Once we have initialized the state engine, what we want to do is create the prototypes of all the tasks that will be running.

Next we setup conifer and the associated drivers. This is needed, because we need to enable power to devices such as the BMS, and motor controller so that we can talk to them over CANbus

Now we are going to create a bunch of tasks that will initialize our car's external devices. The reason that these are RTOS tasks, is that it takes a buncha time to verify the existance of some devices. As a direct result, we can sorta just wait around and check that each task sends a message confirming that it has successfully executed. :) However, first we need to actually create a Queue for these tasks to use

```
/  
QueueHandle_t init_validation_queue = xQueueCreate(8,sizeof(uv_init_task_response));  
if(init_validation_queue == NULL){
```

```

    __uvInitPanic();
}

The next big thing on our plate is checking the status of all external devices we need, and initializing them with appropriate parameters. These are split into tasks because it takes a bit of time, especially for devices that need to be configured via CANBus such as the motor controller. That is why it is split the way it is, to allow these to run somewhat concurrently
*/
BaseType_t retval;
uint16_t ext_devices_status = 0x0000; //Tracks which devices are currently setup
//osThreadDef_t MC_init_thread = {"MC_init",MC_Startup,osPriorityNormal,128,0};
uv_init_task_args* MC_init_args = uvMalloc(sizeof(uv_init_task_args));
MC_init_args->init_info_queue = init_validation_queue;
MC_init_args->specific_args = &(current_vehicle_settings->mc_settings);

retval =
    xTaskCreate(MC_Startup,"MC_init",256,MC_init_args,osPriorityAboveNormal,&(MC_init_args->meta_task_handle));
if(retval != pdPASS){
    //FUCK
    error_msg = "bruh";
} else{
    ext_devices_status |= 0x01U << MOTOR_CONTROLLER;
}

This thread is for initializing the BMS
*/
//osThreadDef_t BMS_init_thread = {"BMS_init",BMS_Init,osPriorityNormal,128,0};
uv_init_task_args* BMS_init_args = uvMalloc(sizeof(uv_init_task_args));
BMS_init_args->init_info_queue = init_validation_queue;
BMS_init_args->specific_args = &(current_vehicle_settings->bms_settings);
//BMS_init_args->meta_task_handle = osThreadCreate(&BMS_init_thread,BMS_init_args);
retval =
    xTaskCreate(BMS_Init,"BMS_init",256,BMS_init_args,osPriorityAboveNormal,&(BMS_init_args->meta_task_handle));
if(retval != pdPASS){
    //FUCK
    error_msg = "bruh";
} else{
    ext_devices_status |= 0x01U << BMS;
}

This variable is a tracker that tracks which devices have successfully initialized
/*
uv_init_task_args* IMD_init_args = uvMalloc(sizeof(uv_init_task_args));
IMD_init_args->init_info_queue = init_validation_queue;
IMD_init_args->specific_args = &(current_vehicle_settings->imd_settings);
retval =
    xTaskCreate(initIMD,"IMD_init",128,IMD_init_args,osPriorityAboveNormal,&(IMD_init_args->meta_task_handle));
if(retval != pdPASS){
    //FUCK
    error_msg = "bruh";
} else{
    ext_devices_status |= 0x01U << IMD;
}

// uv_init_task_args* PDU_init_args = uvMalloc(sizeof(uv_init_task_args));
// PDU_init_args->init_info_queue = init_validation_queue;
// PDU_init_args->specific_args = &(current_vehicle_settings->imd_settings);
// retval =
//     xTaskCreate(initPDU,"PDU_init",128,PDU_init_args,osPriorityAboveNormal,&(PDU_init_args->meta_task_handle));
//     //pass in the right settings, dumdum
// if(retval != pdPASS){
//     //FUCK
//     error_msg = "bruh";
// }

initADCTask(); //START THE ADCs

```

Wait for all the spawned in tasks to do their thing. This should not take that long, but we wanna be sure that everything is chill If we are say, missing a BMS, then it will not allow you to proceed past the initialisation step This is handled by a message buffer, that takes inputs from all of the tasks

We allocate space for a response from the initialization.

Clean up, clean up, everybody clean up, clean up, clean up, everybody do your share! The following code cleans up all the threads that were running, and free up used memory

Definition at line 50 of file [uvfr_utils.c](#).

References [__uvInitPanic\(\)](#), [uv_task_info::active_states](#), [BAMO_RFE](#), [BAMO_RUN](#), [BeepBeepMotherFucker\(\)](#), [BMS](#), [BMS_Init\(\)](#), [uv_vehicle_settings::bms_settings](#), [CAN_RX_DAEMON_NAME](#), [CAN_TX_DAEMON_NAME](#), [CANbusRxSvcDaemon\(\)](#), [CANbusTxSvcDaemon\(\)](#), [changeVehicleState\(\)](#), [coniferEnChannel\(\)](#), [coniferInit\(\)](#), [current_vehicle_settings](#), [uv_init_task_response::device](#), [uv_init_task_response::errmsg](#), [HVIL_PWR](#), [IMD](#), [uv_vehicle_settings::imd_settings](#), [INIT_CHECK_PERIOD](#), [uv_init_task_args::init_info_queue](#), [init_task_handle](#), [initADCTask\(\)](#), [initIMD\(\)](#), [MAX_INIT_TIME](#), [uv_vehicle_settings::mc_settings](#), [MC_Startup\(\)](#), [uv_init_task_args::meta_task_handle](#), [MOTOR_CONTROLLER](#), [uv_init_task_response::nchar](#), [SDC_BOARD_PWR](#), [uv_init_task_args::specific_args](#), [uv_init_task_response::status](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [UV_OK](#), [UV_READY](#), [uvCreateServiceTask\(\)](#), [uvInitDiagnostics\(\)](#), [uvInitStateEngine\(\)](#), [uvSettingsInit\(\)](#), [uvStartStateMachine\(\)](#), and [uvStartTask\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.51.5.3 uvIsPTRValid()

```
uv_status uvIsPTRValid (
    void * ptr)
```

function that checks to make sure a pointer points to a place it is allowed to point to

The primary motivation for this is to avoid trying to dereference a pointer that doesn't exist, and triggering the [HardFaultHandler\(\)](#). That is never a fun time. This allows us to exit gracefully instead of getting stuck in an IRQ handler

Exiting gracefully can be pretty neat sometimes.

Definition at line 487 of file [uvfr_utils.c](#).

References [UV_ERROR](#), [UV_OK](#), and [UV_WARNING](#).

Referenced by [__uvFreeCriticalSection\(\)](#), [__uvFreeOS\(\)](#), [__uvMallocOS\(\)](#), [associateDaqParamWithVar\(\)](#), and [uvCreateTmpSettingsCopy\(\)](#).

7.51.6 Variable Documentation

7.51.6.1 data_size

```
const uint8_t data_size[] [extern]
```

Definition at line 21 of file [uvfr_utils.c](#).

Referenced by [insertParamToRegister\(\)](#), and [uvSettingsProgrammerTask\(\)](#).

7.51.6.2 global_context

```
uv_internal_params global_context [extern]
```

7.52 uvfr_utils.h

[Go to the documentation of this file.](#)

```

00001
00021 #ifndef INC_UVFR_UTILS_H_
00022 #define INC_UVFR_UTILS_H_
00023
00024 #include "uvfr_global_config.h"
00025
00026 #include "main.h"
00027 #include "adc.h"
00028 #include "can.h"
00029 #include "dma.h"
00030 #include "tim.h"
00031 #include "gpio.h"
00032
00033
00034 #include "uvfr_settings.h"
00035 #include "uvfr_state_engine.h"
00036 #include "uvfr_diagnostics.h"
00037 #include "rb_tree.h"
00038 #include "uvfr_vehicle_commands.h"
00039
00040 #include "bms.h"
00041 #include "motor_controller.h"
00042 #include "dash.h"
00043 #include "imd.h"
00044 #include "pdu.h"
00045 #include "daq.h"
00046
00047
00048
00049 #include "uvfr_conifer.h"
00050 #include "uvfr_vehicle_logger.h"
00051 //mainstay meat and potatoes tasks
00052 #include "driving_loop.h"
00053 #include "temp_monitoring.h"
00054 #include "odometer.h"
00055
00056 #include "FreeRTOSConfig.h"
00057
00058 //#include "stdlib.h"
00059 #include "stdint.h"
00060 #include <stdlib.h>
00061 #include "cmsis_os.h"
00062 #include "FreeRTOS.h"
00063 #include "message_buffer.h"
00064 #include "task.h"
00065
00066
00071 #define _BV(x) _BV_16(x)
00072 #define _BV_8(x) ((uint8_t)(0x01U >> x))
00073 #define _BV_16(x) ((uint16_t)(0x01U >> x))
00074 #define _BV_32(x) ((uint32_t)(0x01U >> x))
00075
00076 #define endianSwap(x) endianSwap16(x)
00077 #define endianSwap8(x) x //if someone calls this, why? its 1 byte... but here ya go I guess
00078 #define endianSwap16(x) (((x & 0x00FF)«8) | ((x & 0xFF00)«8))
00079 #define endianSwap32(x) (((x & 0x000000FF)«16) | ((x & 0x0000FF00)«8) | ((x & 0x00FF0000)«8) | ((x & 0xFF000000)«16))
00080
00081 #define deserializeSmallE16(x,i) ((x[i]) | (x[i+1]«8))
00082 #define deserializeSmallE32(x,i) ((x[i]) | (x[i+1]«8) | (x[i+2]«16) | (x[i+3]«24))
00083 #define deserializeBigE16(x,i) ((x[i]«8) | (x[i+1]))
00084 #define deserializeBigE32(x,i) ((x[i]«24) | (x[i+1]«16) | (x[i+2]«8) | (x[i+3]))
00085
00086 #define serializeSmallE16(x,d,i) x[i]=d&0x00FF; x[i+1]=(d&0xFF00)«8
00087 #define serializeSmallE32(x,d,i) x[i]=d&0x000000FF; x[i+1]=(d&0x0000FF00)«8; x[i+2]=(d&0x00FF0000)«16;
00088 #define serializeBigE16(x,d,i) x[i+1]=d&0x00FF; x[i]=(d&0xFF00)«8
00089 #define serializeBigE32(x,d,i) x[i+3]=d&0x000000FF; x[i+2]=(d&0x0000FF00)«8; x[i+1]=(d&0x00FF0000)«16;
00090 x[i]=(d&0xFF000000)«24
00114 #define setBits(x,msk,data) x=(x&(~msk))|data)
00115
00119 #define isPowerOfTwo(x) (x&&(! (x&(x-1))))
00120
00124 #define safePtrRead(x) (*((x)?x:uvPanic("nullptr_deref",0)))
00125 #define safePtrWrite(p,x) (*((p)?p:&x))
00126
00127
00129 #define false 0
00130 #define true !false
00131
00133 #define TEXTIFY(A) #A

```

```

00134
00137 //Typedefs used throughout vehicle
00138
00139 typedef uint8_t bool;
00140 typedef uint8_t uv_task_id;
00141 typedef enum uv_task_cmd_e uv_task_cmd;
00142 //typedef enum
00143 typedef uint8_t uv_ext_device_id;
00144 typedef uint32_t uv_timespan_ms;
00145
00146 //typedef enum CONIFER_OUTPUT conifer_output_channel;
00147
00148
00149
00150
00151 //Time limits for the initialization. Car has 2.5 seconds to initialize all peripherals, before it
     decides that something has gone horrifically wrong
00152 #define MAX_INIT_TIME 2500//if the car takes more than 2.5 seconds to boot, something seems a little
     fishy
00153 #define INIT_CHECK_PERIOD 100
00154
00155 //Memory management macros
00156
00157
00158
00159 //Some fun CAN macros
00160 #define UV_CAN1
00161 #define UV_CAN2
00162
00163 //Feature flags WHY IS THIS NOT IN GLOBALCONFIG?????
00164 #define USE_OLED_DEBUG 1
00165
00173 typedef enum uv_status_t{
00174     UV_OK,
00175     UV_WARNING,
00176     UV_ERROR,
00177     UV_ABORTED //nothing wrong per say,
00178 }uv_status;
00179
00180
00184 typedef enum {
00185     UV_UINT8 = 0,
00186     UV_INT8 = 1,
00187     UV_UINT16 = 2,
00188     UV_INT16 = 3,
00189     UV_UINT32 = 4,
00190     UV_INT32 = 5,
00191     UV_FLOAT = 6,
00192     UV_DOUBLE = 7,
00193     UV_INT64 = 8,
00194     UV_UINT64 = 9,
00195     UV_STRING = 10
00196 }
00197 }data_type;
00198
00199
00200 //not really sure how I want this implemented yet. Variable number of driving modes?
00201 enum uv_driving_mode_t{
00202     normal,
00203     accel,
00204     econ,
00205     limp
00206 };
00207
00211 enum uv_external_device{
00212     MOTOR_CONTROLLER = 0,
00213     BMS = 1,
00214     IMD = 2,
00215     PDU = 3
00216 };
00217
00218 typedef enum access_control_t{
00219     UV_NONE,
00220     UV_DUMB_FLAG,
00221     UV_MUTEX,
00222     UV_BINARY_SEMAPHORE,
00223     UV_SEMAPHORE
00224 }access_control_type;
00225
00229 typedef enum uv_msg_type_t{
00230     UV_TASK_START_COMMAND,
00231     UV_TASK_DELETE_COMMAND,
00232     UV_TASK_SUSPEND_COMMAND,
00233     UV_COMMAND_ACKNOWLEDGEMENT,
00234     UV_TASK_STATUS_REPORT,
00235     UV_ERROR_REPORT,
00236     UV_WAKEUP,

```

```
00237     UV_PARAM_REQUEST,
00238     UV_PARAM_READY,
00239     UV_RAW_DATA_TRANSFER,
00240     UV_SC_COMMAND,
00241     UV_INVALID_MSG,
00242     UV_ASSIGN_TASK
00243
00244 }uv_msg_type;
00245
00246 struct uv_mutex_info{
00247     SemaphoreHandle_t handle;
00248
00249 };
00250
00251 struct uv_binary_semaphore_info{
00252     SemaphoreHandle_t handle;
00253
00254 };
00255
00256 struct uv_semaphore_info{
00257     SemaphoreHandle_t handle;
00258
00259 };
00260
00261 typedef union access_control_info{
00262     struct uv_mutex_info mutex;
00263     struct uv_binary_semaphore_info bin_semaphore;
00264     struct uv_semaphore_info semaphore;
00265
00266 }access_control_info;
00267
00268
00269 #define UV_CAN_EXTENDED_ID 0b00000001
00270
00271 #define CAN_BUS_1 0b00000010
00272 #define CAN_BUS_2 0b00000100
00273
00274
00275 #define UV_CAN_CHANNEL_MASK 0b000000110
00276 #define UV_CAN_DYNAMIC_MEM 0b00001000
00277
00278
00279 typedef struct uv_CAN_msg{
00280     uint8_t flags;
00281     uint8_t dlc;
00282     uint32_t msg_id;
00283     uint8_t data[8];
00284
00285 }uv_CAN_msg;
00286
00287
00288 typedef struct uv_init_struct{
00289     bool use_default_settings;//Flag for using default settings
00290
00291 }uv_init_struct;
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
```

```

00344     TaskHandle_t meta_task_handle; //Handle to itself, which it can use to delete itself
00345 }uv_init_task_args;
00346
00347
00353 typedef struct uv_internal_params{
00354     uv_init_struct* init_params;
00355     uv_vehicle_settings* vehicle_settings;
00356     p_status peripheral_status[8];
00357     uint16_t e_code[8];
00358 }uv_internal_params;
00359
00360
00367 typedef struct uv_init_task_response{
00368     uv_status status; //Did it succeed? This gets to be a UV_OK if success
00369     uv_ext_device_id device; //Which device was it
00370     uint8_t nchar;
00371     char* errmsg; //if we didn't succeed, then what went wrong?
00372 }uv_init_task_response;
00373
00374
00375
00376 #ifndef UV_UTILS_SRC_IMPLEMENTATION
00377     extern uv_internal_params global_context;
00378
00379     extern const uint8_t data_size[];
00380
00381 #endif
00382
00383 void uvInit(void * arguments);
00384
00385 void __uvInitPanic();
00386
00387 #ifndef uvMalloc
00388 #if USE_OS_MEM_MGMT
00389     void* __uvMallocOS(size_t memrequest);
00390     #define uvMalloc(x) __uvMallocOS(x)
00391 #else //default to STLib
00392     void * __uvMallocCriticalSection(size_t memrequest);
00393     #define uvMalloc(x) __uvMallocCriticalSection(x)
00394
00395 #endif //OS mem mgmt
00396 #endif //allow macro override
00397
00398 #ifndef uvFree
00399 #if USE_OS_MEM_MGMT
00400     uv_status __uvFreeOS(void* ptr);
00401     #define uvFree(x) __uvFreeOS(x)
00402
00403 #else
00404     uv_status __uvFreeCriticalSection(void* ptr);
00405     #define uvFree(x) __uvFreeCriticalSection(x)
00406
00407 #endif //OS mem mgmt
00408 #endif //Allows macro overriding
00409
00410 uv_status uvIsPTRValid(void* ptr);
00411
00412
00413
00414
00415
00416 #endif /* INC_UVFR_UTILS_H_ */

```

7.53 Core/lnc/uvfr_vehicle_commands.h File Reference

```
#include "uvfr_global_config.h"
#include "uvfr_utils.h"
```

Include dependency graph for uvfr_vehicle_commands.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [output_channel](#)
- struct [output_channel_settings](#)

Typedefs

- `typedef struct output_channel output_channel`
- `typedef struct output_channel_settings output_channel_settings`

Functions

- `void uvActivateHorn ()`
- `void uvSilenceHorn ()`
- `void BeepBeepMotherFucker ()`
- `void uvSecureVehicle ()`

Function to put vehicle into safe state.

7.53.1 Typedef Documentation

7.53.1.1 `output_channel`

```
typedef struct output_channel output_channel
```

7.53.1.2 `output_channel_settings`

```
typedef struct output_channel_settings output_channel_settings
```

Definition at line 17 of file [uvfr_settings.c](#).

7.53.2 Function Documentation

7.53.2.1 `BeepBeepMotherFucker()`

```
void BeepBeepMotherFucker ()
```

Definition at line 5 of file [uvfr_vehicle_commands.c](#).

References [coniferDisChannel\(\)](#), [coniferEnChannel\(\)](#), [coniferToggleChannel\(\)](#), and [HORN](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.53.2.2 `uvActivateHorn()`

```
void uvActivateHorn ()
```

7.53.2.3 uvSecureVehicle()

```
void uvSecureVehicle ()
```

Function to put vehicle into safe state.

Should perform the following functions in order:

- Prevent new MC torque or speed requests
- Open shutdown cct

Definition at line 23 of file [uvfr_vehicle_commands.c](#).

References [BAMO_RFE](#), [coniferDisChannel\(\)](#), and [MC_Shutdown\(\)](#).

Here is the call graph for this function:

7.53.2.4 uvSilenceHorn()

```
void uvSilenceHorn ()
```

7.54 uvfr_vehicle_commands.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * uvfr_vehicle_commands.h
00003  *
00004  * Created on: Nov 27, 2024
00005  * Author: byo10
00006 */
00007
00016 #ifndef INC_UVFR_VEHICLE_COMMANDS_H_
00017 #define INC_UVFR_VEHICLE_COMMANDS_H_
00018
00019 #include "uvfr_global_config.h"
00020 #include "uvfr_utils.h"
00021
00022 #if (UV19_PDU + ECUMASTER_PMU) > 1
00023 #error "Invalid PDU configuration"
00024 #endif
00025
00026 /*
00027  *
00028 */
00029 typedef struct output_channel{
00030     uint16_t I_max;
00031     uint8_t flags;
00032     uint8_t chID;
00033 }output_channel;
00034
00035 /*
00036  *
00037 */
00038 typedef struct output_channel_settings{
00039     uint16_t var;
00040 }output_channel_settings;
00041
00042 #ifndef uvOpenSDC
00043
00044
00045     #define uvOpenSDC() coniferDisChannel(HVIL_PWR)
00046 #endif
00047
00048 #ifndef uvCloseSDC
00049
00050     #define uvCloseSDC() coniferEnChannel(HVIL_PWR)
```

```

00051
00052 #endif //Close SDC
00053
00054 #ifndef uvStartFans
00055
00056 #define uvStartFans() do{\
00057     coniferEnChannel(RAD_FANS1);\
00058     coniferEnChannel(RAD_FANS2);\
00059 }while(0);
00060
00061 #endif //start fans
00062
00063 #ifndef uvStopFans
00064
00065 #define uvStopFans(x) do{\
00066     coniferDisChannel(RAD_FANS1);\
00067     coniferDisChannel(RAD_FANS2);\
00068 }while(0);
00069
00070 #endif //start fans
00071
00072 #ifndef uvStartCoolantPump
00073
00074 #define uvStartCoolantPump() confiferEnChannel(COOLANT_PUMP1)
00075
00076 #endif
00077
00078 //Coolant pump
00079 #ifndef uvStopCoolantPump
00080
00081 #define uvStopCoolantPump() confiferDisChannel(COOLANT_PUMP1)
00082
00083 #endif
00084
00085 //Turn on the horn
00086 void uvActivateHorn();
00087
00088 //Turn off the horn
00089 void uvSilenceHorn();
00090
00091 //Beep Beep
00092 void BeepBeepMotherFucker();
00093
00094 //Put vehicle into a safe state
00095 void uvSecureVehicle();
00096
00097
00098 #endif /* INC_UVFR_VEHICLE_COMMANDS_H_ */

```

7.55 Core/Inc/uvfr_vehicle_logger.h File Reference

```
#include <stdint.h>
#include "uvfr_utils.h"
#include "uvfr_state_engine.h"
```

Include dependency graph for uvfr_vehicle_logger.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [vehicle_log_entry](#)

Enumerations

- enum [fault_event_type_e](#) {
 FAULT_PANIC = 0 , FAULT_TASK_FAIL , FAULT_TASK_LATE , FAULT_STATE_ILLEGAL ,
 FAULT_SCD_TIMEOUT , FAULT_SENSOR_ERROR , FAULT_STATE_TRANSITION , FAULT_MANUAL ,
 FAULT_FLASH_ERROR , FAULT_SETTINGS_CORRUPT , FAULT_CAN_MSG_FAIL }

Functions

- void `logVehicleFault` (`fault_event_type_e` type, `uv_task_info` *task, const char *msg, const char *file, int line, const char *func, `bool` is_panic)
- `uint8_t getLogCount` (void)
- const `vehicle_log_entry` * `getLogEntry` (`uint8_t` index)
- void `dumpLogsToUART` (void)
- `uint32_t getSystemFaultFlags` (void)
- void `clearSystemFaultFlags` (`uint32_t` flags_to_clear)
- void `flushLogsToCAN` (void)

7.55.1 Enumeration Type Documentation

7.55.1.1 `fault_event_type_e`

enum `fault_event_type_e`

Enumerator

FAULT_PANIC	
FAULT_TASK_FAIL	
FAULT_TASK_LATE	
FAULT_STATE_ILLEGAL	
FAULT_SCD_TIMEOUT	
FAULT_SENSOR_ERROR	
FAULT_STATE_TRANSITION	
FAULT_MANUAL	
FAULT_FLASH_ERROR	
FAULT_SETTINGS_CORRUPT	
FAULT_CAN_MSG_FAIL	

Definition at line 24 of file `uvfr_vehicle_logger.h`.

7.55.2 Function Documentation

7.55.2.1 `clearSystemFaultFlags()`

```
void clearSystemFaultFlags (
    uint32_t flags_to_clear)
```

clears specific fault flags (set bits) from the bitfield

Definition at line 46 of file `uvfr_vehicle_logger.c`.

7.55.2.2 dumpLogsToUART()

```
void dumpLogsToUART (
    void )
```

prints all stored logs to UART (e.x, for debugging)

Definition at line 68 of file [uvfr_vehicle_logger.c](#).

References [vehicle_log_entry::file](#), [vehicle_log_entry::func](#), [getLogCount\(\)](#), [getLogEntry\(\)](#), [vehicle_log_entry::is_panic](#), [vehicle_log_entry::line](#), [vehicle_log_entry::msg](#), [vehicle_log_entry::task_id](#), [vehicle_log_entry::task_name](#), [vehicle_log_entry::task_state](#), [vehicle_log_entry::timestamp](#), [vehicle_log_entry::type](#), and [vehicle_log_entry::vehicle_state](#).

Here is the call graph for this function:

7.55.2.3 flushLogsToCAN()

```
void flushLogsToCAN (
    void )
```

Flush all current log entries out over CAN for diagnostic laptop tools. Each entry is sent as a single compact CAN frame.

Frame layout: ID = LOG_DUMP_CAN_ID DLC = 8 Byte 0 : fault type Byte 1 : timestamp LSB Byte 2 : timestamp MSB Byte 3 : task ID Byte 4 : is_panic flag Byte 5 : vehicle_state Byte 6 : reserved (0) Byte 7 : reserved (0)

Definition at line 155 of file [uvfr_vehicle_logger.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [getLogCount\(\)](#), [getLogEntry\(\)](#), [vehicle_log_entry::is_panic](#), [LOG_DUMP_CAN_ID](#), [uv_CAN_msg::msg_id](#), [vehicle_log_entry::task_id](#), [vehicle_log_entry::timestamp](#), [vehicle_log_entry::type](#), [uvSendCanMSG\(\)](#), and [vehicle_log_entry::vehicle_state](#).

Referenced by [handleIncomingLaptopMsg\(\)](#).

Here is the call graph for this function:

7.55.2.4 getLogCount()

```
uint8_t getLogCount (
    void )
```

returns how many logs are stored right now

Definition at line 53 of file [uvfr_vehicle_logger.c](#).

References [MAX_LOG_ENTRIES](#).

Referenced by [dumpLogsToUART\(\)](#), [flushLogsToCAN\(\)](#), and [getLogEntry\(\)](#).

7.55.2.5 getLogEntry()

```
const vehicle\_log\_entry * getLogEntry (
    uint8_t index)
```

returns a specific log entry by index

Definition at line 60 of file [uvfr_vehicle_logger.c](#).

References [getLogCount\(\)](#).

Referenced by [dumpLogsToUART\(\)](#), and [flushLogsToCAN\(\)](#).

Here is the call graph for this function:

7.55.2.6 getSystemFaultFlags()

```
uint32_t getSystemFaultFlags (
    void )
```

returns the current bitfield (1 bit per fault type)

Definition at line 39 of file [uvfr_vehicle_logger.c](#).

7.55.2.7 logVehicleFault()

```
void logVehicleFault (
    fault\_event\_type\_e type,
    uv\_task\_info * task,
    const char * msg,
    const char * file,
    int line,
    const char * func,
    bool is_panic)
```

adds a new log entry for a fault or error.

Definition at line 97 of file [uvfr_vehicle_logger.c](#).

References [vehicle_log_entry::file](#), [vehicle_log_entry::func](#), [vehicle_log_entry::is_panic](#), [vehicle_log_entry::line](#), [MAX_LOG_ENTRIES](#), [vehicle_log_entry::msg](#), [uv_task_info::task_id](#), [vehicle_log_entry::task_id](#), [uv_task_info::task_name](#), [vehicle_log_entry::task_name](#), [uv_task_info::task_state](#), [vehicle_log_entry::task_state](#), [vehicle_log_entry::timestamp](#), [vehicle_log_entry::type](#), [UV_TASK_NOT_STARTED](#), [vehicle_log_entry::vehicle_state](#), and [vehicle_state](#).

Referenced by [__uvPanic\(\)](#).

7.56 uvfr_vehicle_logger.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * uvfr_vehicle_logger.h
00003 *
00004 * Created on: Jul 13, 2025
00005 * Author: rachangrewal
00006 *
00007 * Header for the vehicle logger system.
00008 * Tracks and stores faults in a small log buffer,
00009 * and uses a bitfield to flag which types of faults have occurred.
00010 */
00011
00012 #ifndef INC_UVFR_VEHICLE_LOGGER_H_
00013 #define INC_UVFR_VEHICLE_LOGGER_H_
00014
00015 #include <stdint.h>
00016 #include "uvfr_utils.h"
00017 #include "uvfr_state_engine.h" // for uv_task_info, uv_task_status, etc.
00018
00019
00020 typedef uint8_t bool;
00021
00022 // types of faults we can log, this might be over kill
00023
00024 typedef enum {
00025     FAULT_PANIC = 0,
00026     FAULT_TASK_FAIL,
00027     FAULT_TASK_LATE,
00028     FAULT_STATE_ILLEGAL,
00029     FAULT_SCD_TIMEOUT,
00030     FAULT_SENSOR_ERROR,
00031     FAULT_STATE_TRANSITION,
00032     FAULT_MANUAL,
00033     FAULT_FLASH_ERROR,
00034     FAULT_SETTINGS_CORRUPT,
00035     FAULT_CAN_MSG_FAIL,
00036     // Add more if needed?
00037 } fault_event_type_e;
00038
00039
00040 // log entry format
00041
00042 typedef struct {
00043     fault_event_type_e type;      // what type of error (enum)
00044     const char* task_name;        // name of the task (if any)
00045     const char* msg;              // error message
00046     uint32_t timestamp;           // time of event (xTaskGetTickCount)
00047     const char* file;             // source file
00048     int line;                   // line number
00049     const char* func;
00050     uint8_t is_panic;            // was this triggered by a panic?
00051     uv_task_id task_id;          // task ID (or 0xFF if none)
00052     uv_vehicle_state vehicle_state; // vehicle state at time of error
00053     uv_task_status task_state;   // what state the task was in (if known)
00054 } vehicle_log_entry;
00055
00056
00057
00058 // public logger functions
00059
00060 // logs a fault or panic
00061 void logVehicleFault(fault_event_type_e type,
00062                         uv_task_info* task,
00063                         const char* msg,
00064                         const char* file,
00065                         int line,
00066                         const char* func,
00067                         bool is_panic);
00068
00069 // returns number of log entries stored
00070 uint8_t getLogCount(void);
00071
00072 // returns a pointer to a log entry by index
00073 const vehicle_log_entry* getLogEntry(uint8_t index);
00074
00075 // prints all logs to UART
00076 void dumpLogsToUART(void);
00077
00078 // returns current fault bitfield
00079 uint32_t getSystemFaultFlags(void);
00080
00081 // clears fault bits (bitmask of flags to clear)
00082 void clearSystemFaultFlags(uint32_t flags_to_clear);

```

```

00083
00084 // Sends all stored log entries as formatted CAN messages.
00085 // Intended for post-panic diagnostics.
00086 void flushLogsToCAN(void);
00087
00088
00089 #endif // UVFR_VEHICLE_LOGGER_H

```

7.57 Core/Src/adc.c File Reference

This file provides code for the configuration of the ADC instances.

```

#include "adc.h"
#include "uvfr_utils.h"
#include "../FreeRTOS/Source/CMSIS_RTOS/cmsis_os.h"
Include dependency graph for adc.c:

```

7.58 adc.c

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Includes -----*/
00021 #include "adc.h"
00022
00023 /* USER CODE BEGIN 0 */
00024 #include "uvfr_utils.h"
00025
00026 extern volatile uint32_t adc_buf1[5]; // ADC1 - high priority readings
00027
00028 extern uint16_t adc1_APPS1; //These are the locations for the sensor inputs for APPS and BPS
00029 extern uint16_t adc1_APPS2;
00030 extern uint16_t adc1_BPS1;
00031 extern uint16_t adc1_BPS2; // Allows access to buffer from main.c
00032 extern uint16_t adc1_pack_curr; //Measurement Via BSPD
00033
00034 extern volatile uint32_t adc_buf2[7]; // ADC2 - lower priority readings
00035 extern uint16_t adc2_steering_pos;
00036 extern uint16_t adc2_CoolantTemp1;
00037 extern uint16_t adc2_CoolantTemp2;
00038 extern uint16_t adc2_CoolantFlow;
00039
00040
00041 extern volatile uint32_t adc_buf3[4];
00042 extern uint16_t adc3_damper_FL;
00043 extern uint16_t adc3_damper_FR;
00044 extern uint16_t adc3_damper_RL;
00045 extern uint16_t adc3_damper_RR;
00046
00047 /* USER CODE END 0 */
00048
00049 ADC_HandleTypeDef hadc1;
00050 ADC_HandleTypeDef hadc2;
00051 ADC_HandleTypeDef hadc3;
00052 DMA_HandleTypeDef hdma_adc1;
00053 DMA_HandleTypeDef hdma_adc2;
00054 DMA_HandleTypeDef hdma_adc3;
00055
00056 /* ADC1 init function */
00057 void MX_ADC1_Init(void)
00058 {
00059
00060     /* USER CODE BEGIN ADC1_Init_0 */
00061     //code most be here or it will be deleted
00062     /* USER CODE END ADC1_Init_0 */
00063
00064     ADC_ChannelConfTypeDef sConfig = {0};
00065     ADC_InjectionConfTypeDef sConfigInjected = {0};
00066
00067     /* USER CODE BEGIN ADC1_Init_1 */
00068
00069     /* USER CODE END ADC1_Init_1 */
00070

```

```
00073     hadc1.Instance = ADC1;
00074     hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV8;
00075     hadc1.Init.Resolution = ADC_RESOLUTION_12B;
00076     hadc1.Init.ScanConvMode = ENABLE;
00077     hadc1.Init.ContinuousConvMode = DISABLE;
00078     hadc1.Init.DiscontinuousConvMode = DISABLE;
00079     hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
00080     hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
00081     hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00082     hadc1.Init.NbrOfConversion = 5;
00083     hadc1.Init.DMAContinuousRequests = DISABLE;
00084     hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
00085     if (HAL_ADC_Init(&hadc1) != HAL_OK)
00086     {
00087         Error_Handler();
00088     }
00089
00090     sConfig.Channel = ADC_CHANNEL_0;
00091     sConfig.Rank = 1;
00092     sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;
00093     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
00094     {
00095         Error_Handler();
00096     }
00097
00098     sConfig.Channel = ADC_CHANNEL_1;
00099     sConfig.Rank = 2;
00100     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
00101     {
00102         Error_Handler();
00103     }
00104
00105     sConfig.Channel = ADC_CHANNEL_2;
00106     sConfig.Rank = 3;
00107     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
00108     {
00109         Error_Handler();
00110     }
00111
00112     sConfig.Channel = ADC_CHANNEL_3;
00113     sConfig.Rank = 4;
00114     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
00115     {
00116         Error_Handler();
00117     }
00118
00119     sConfigInjected.InjectecdChannel = ADC_CHANNEL_0;
00120     sConfigInjected.InjectecdRank = 1;
00121     sConfigInjected.InjectecdNbrOfConversion = 4;
00122     sConfigInjected.InjectecdSamplingTime = ADC_SAMPLETIME_3CYCLES;
00123     sConfigInjected.ExternalTrigInjecConvEdge = ADC_EXTERNALTRIGINJECCONVEDGE_NONE;
00124     sConfigInjected.ExternalTrigInjecConv = ADC_INJECTED_SOFTWARE_START;
00125     sConfigInjected.AutoInjectedConv = DISABLE;
00126     sConfigInjected.InjectecdDiscontinuousConvMode = DISABLE;
00127     sConfigInjected.InjectecdOffset = 0;
00128     if (HAL_ADCEEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
00129     {
00130         Error_Handler();
00131     }
00132
00133     sConfigInjected.InjectecdRank = 2;
00134     if (HAL_ADCEEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
00135     {
00136         Error_Handler();
00137     }
00138
00139     sConfigInjected.InjectecdRank = 3;
00140     if (HAL_ADCEEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
00141     {
00142         Error_Handler();
00143     }
00144
00145     sConfigInjected.InjectecdRank = 4;
00146     if (HAL_ADCEEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
00147     {
00148         Error_Handler();
00149     }
00150
00151
00152     /* USER CODE BEGIN ADC1_Init_2 */
00153
00154     /* USER CODE END ADC1_Init_2 */
```

```
00178 }
00179 /* ADC2 init function */
00180 void MX_ADC2_Init(void)
00181 {
00182
00183     /* USER CODE BEGIN ADC2_Init_0 */
00184
00185     /* USER CODE END ADC2_Init_0 */
00186
00187     ADC_ChannelConfTypeDef sConfig = {0};
00188
00189     /* USER CODE BEGIN ADC2_Init_1 */
00190
00191     /* USER CODE END ADC2_Init_1 */
00192
00193     hadc2.Instance = ADC2;
00194     hadc2.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV8;
00195     hadc2.Init.Resolution = ADC_RESOLUTION_12B;
00196     hadc2.Init.ScanConvMode = ENABLE;
00197     hadc2.Init.ContinuousConvMode = DISABLE;
00198     hadc2.Init.DiscontinuousConvMode = DISABLE;
00199     hadc2.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
00200     hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
00201     hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00202     hadc2.Init.NbrOfConversion = 7;
00203     hadc2.Init.DMAContinuousRequests = DISABLE;
00204     hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
00205     if (HAL_ADC_Init(&hadc2) != HAL_OK)
00206     {
00207         Error_Handler();
00208     }
00209
00210     sConfig.Channel = ADC_CHANNEL_5;
00211     sConfig.Rank = 1;
00212     sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;
00213     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00214     {
00215         Error_Handler();
00216     }
00217
00218     sConfig.Channel = ADC_CHANNEL_6;
00219     sConfig.Rank = 2;
00220     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00221     {
00222         Error_Handler();
00223     }
00224
00225     sConfig.Channel = ADC_CHANNEL_7;
00226     sConfig.Rank = 3;
00227     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00228     {
00229         Error_Handler();
00230     }
00231
00232     sConfig.Channel = ADC_CHANNEL_8;
00233     sConfig.Rank = 4;
00234     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00235     {
00236         Error_Handler();
00237     }
00238
00239     sConfig.Channel = ADC_CHANNEL_9;
00240     sConfig.Rank = 5;
00241     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00242     {
00243         Error_Handler();
00244     }
00245
00246     sConfig.Channel = ADC_CHANNEL_14;
00247     sConfig.Rank = 6;
00248     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00249     {
00250         Error_Handler();
00251     }
00252
00253     sConfig.Channel = ADC_CHANNEL_15;
00254     sConfig.Rank = 7;
00255     if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
00256     {
00257         Error_Handler();
00258     }
00259
00260     /* USER CODE BEGIN ADC2_Init_2 */
00261
00262     /* USER CODE END ADC2_Init_2 */
00263 }
```

```
00281 /* ADC3 init function */
00282 void MX_ADC3_Init(void)
00283 {
00284
00285     /* USER CODE BEGIN ADC3_Init_0 */
00286
00287     /* USER CODE END ADC3_Init_0 */
00288
00289     ADC_ChannelConfTypeDef sConfig = {0};
00290
00291     /* USER CODE BEGIN ADC3_Init_1 */
00292
00293     /* USER CODE END ADC3_Init_1 */
00294
00295     hadc3.Instance = ADC3;
00296     hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV8;
00297     hadc3.Init.Resolution = ADC_RESOLUTION_12B;
00298     hadc3.Init.ScanConvMode = ENABLE;
00299     hadc3.Init.ContinuousConvMode = DISABLE;
00300     hadc3.Init.DiscontinuousConvMode = DISABLE;
00301     hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
00302     hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
00303     hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00304     hadc3.Init.NbrOfConversion = 4;
00305     hadc3.Init.DMAContinuousRequests = DISABLE;
00306     hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
00307     if (HAL_ADC_Init(&hadc3) != HAL_OK)
00308     {
00309         Error_Handler();
00310     }
00311
00312     sConfig.Channel = ADC_CHANNEL_10;
00313     sConfig.Rank = 1;
00314     sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;
00315     if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
00316     {
00317         Error_Handler();
00318     }
00319
00320     sConfig.Channel = ADC_CHANNEL_11;
00321     sConfig.Rank = 2;
00322     if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
00323     {
00324         Error_Handler();
00325     }
00326
00327     sConfig.Channel = ADC_CHANNEL_12;
00328     sConfig.Rank = 3;
00329     if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
00330     {
00331         Error_Handler();
00332     }
00333
00334     sConfig.Channel = ADC_CHANNEL_13;
00335     sConfig.Rank = 4;
00336     if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
00337     {
00338         Error_Handler();
00339     }
00340
00341
00342     /* USER CODE BEGIN ADC3_Init_2 */
00343
00344     /* USER CODE END ADC3_Init_2 */
00345
00346 }
00347
00348
00349
00350
00351
00352
00353
00354 }
00355
00356 void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
00357 {
00358
00359     GPIO_InitTypeDef GPIO_InitStruct = {0};
00360     if (adcHandle->Instance==ADC1)
00361     {
00362         /* USER CODE BEGIN ADC1_MspInit_0 */
00363
00364         /* USER CODE END ADC1_MspInit_0 */
00365         /* ADC1 clock enable */
00366         __HAL_RCC_ADC1_CLK_ENABLE();
00367
00368         __HAL_RCC_GPIOA_CLK_ENABLE();
00369         GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
00370                         |GPIO_PIN_4;
00371         GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00372         GPIO_InitStruct.Pull = GPIO_NOPULL;
00373         HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00374
00375         /* ADC1 DMA Init */
00376         /* ADC1 Init */
00377         hdma_adcl.Instance = DMA2_Stream0;
```

```

00385     hdma_adc1.Init.Channel = DMA_CHANNEL_0;
00386     hdma_adc1.Init.Direction = DMA_PERIPH_TO_MEMORY;
00387     hdma_adc1.InitPeriphInc = DMA_PINC_DISABLE;
00388     hdma_adc1.InitMemInc = DMA_MINC_ENABLE;
00389     hdma_adc1.InitPeriphDataAlignment = DMA_PDATAALIGN_WORD;
00390     hdma_adc1.InitMemDataAlignment = DMA_MDATAALIGN_WORD;
00391     hdma_adc1.Init.Mode = DMA_CIRCULAR;
00392     hdma_adc1.Init.Priority = DMA_PRIORITY_HIGH;
00393     hdma_adc1.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
00394     if (HAL_DMA_Init(&hdma_adc1) != HAL_OK)
00395     {
00396         Error_Handler();
00397     }
00398     __HAL_LINKDMA(adCHandle,DMA_Handle,hdma_adc1);
00400
00401     /* ADC1 interrupt Init */
00402     HAL_NVIC_SetPriority(ADC_IRQn, 5, 0);
00403     HAL_NVIC_EnableIRQ(ADC_IRQn);
00404 /* USER CODE BEGIN ADC1_MspInit 1 */
00405
00406 /* USER CODE END ADC1_MspInit 1 */
00407 }
00408 else if(adchandle->Instance==ADC2)
00409 {
00410 /* USER CODE BEGIN ADC2_MspInit 0 */
00411
00412 /* USER CODE END ADC2_MspInit 0 */
00413 /* ADC2 clock enable */
00414 __HAL_RCC_ADC2_CLK_ENABLE();
00415
00416 __HAL_RCC_GPIOA_CLK_ENABLE();
00417 __HAL_RCC_GPIOC_CLK_ENABLE();
00418 __HAL_RCC_GPIOB_CLK_ENABLE();
00419 GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
00420 GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00421 GPIO_InitStruct.Pull = GPIO_NOPULL;
00422 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00423
00424 GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5;
00425 GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00426 GPIO_InitStruct.Pull = GPIO_NOPULL;
00427 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
00428
00429 /* ADC2 DMA Init */
00430 /* ADC2 Init */
00431 hdma_adc2.Instance = DMA2_Stream2;
00432 hdma_adc2.Init.Channel = DMA_CHANNEL_1;
00433 hdma_adc2.Init.Direction = DMA_PERIPH_TO_MEMORY;
00434 hdma_adc2.InitPeriphInc = DMA_PINC_DISABLE;
00435 hdma_adc2.InitMemInc = DMA_MINC_ENABLE;
00436 hdma_adc2.InitPeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
00437 hdma_adc2.InitMemDataAlignment = DMA_MDATAALIGN_HALFWORD;
00438 hdma_adc2.Init.Mode = DMA_NORMAL;
00439 hdma_adc2.Init.Priority = DMA_PRIORITY_HIGH;
00440 hdma_adc2.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
00441 if (HAL_DMA_Init(&hdma_adc2) != HAL_OK)
00442 {
00443     Error_Handler();
00444 }
00445     __HAL_LINKDMA(adchandle,DMA_Handle,hdma_adc2);
00446
00447 /* ADC2 interrupt Init */
00448 HAL_NVIC_SetPriority(ADC_IRQn, 5, 0);
00449 HAL_NVIC_EnableIRQ(ADC_IRQn);
00450 /* USER CODE BEGIN ADC2_MspInit 1 */
00451
00452 /* USER CODE END ADC2_MspInit 1 */
00453 }
00454 else if(adchandle->Instance==ADC3)
00455 {
00456 /* USER CODE BEGIN ADC3_MspInit 0 */
00457
00458 /* USER CODE END ADC3_MspInit 0 */
00459 /* ADC3 clock enable */
00460 __HAL_RCC_ADC3_CLK_ENABLE();
00461
00462 __HAL_RCC_GPIOC_CLK_ENABLE();
00463 GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3;
00464 GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00465 GPIO_InitStruct.Pull = GPIO_NOPULL;

```

```
00487     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
00488
00489     /* ADC3 DMA Init */
00490     /* ADC3 Init */
00491     hdma_adc3.Instance = DMA2_Stream1;
00492     hdma_adc3.Init.Channel = DMA_CHANNEL_2;
00493     hdma_adc3.Init.Direction = DMA_PERIPH_TO_MEMORY;
00494     hdma_adc3.InitPeriphInc = DMA_PINC_DISABLE;
00495     hdma_adc3.InitMemInc = DMA_MINC_ENABLE;
00496     hdma_adc3.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
00497     hdma_adc3.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
00498     hdma_adc3.Init.Mode = DMA_NORMAL;
00499     hdma_adc3.Init.Priority = DMA_PRIORITY_HIGH;
00500     hdma_adc3.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
00501     if (HAL_DMA_Init(&hdma_adc3) != HAL_OK)
00502     {
00503         Error_Handler();
00504     }
00505
00506     __HAL_LINKDMA(adCHandle, DMA_Handle, hdma_adc3);
00507
00508     /* ADC3 interrupt Init */
00509     HAL_NVIC_SetPriority(ADC_IRQn, 5, 0);
00510     HAL_NVIC_EnableIRQ(ADC_IRQn);
00511 /* USER CODE BEGIN ADC3_MspInit 1 */
00512
00513 /* USER CODE END ADC3_MspInit 1 */
00514 }
00515 }
00516
00517 void HAL_ADC_MspDeInit(ADC_HandleTypeDef* adcHandle)
00518 {
00519
00520     if(adcHandle->Instance==ADC1)
00521     {
00522         /* USER CODE BEGIN ADC1_MspDeInit 0 */
00523
00524         /* USER CODE END ADC1_MspDeInit 0 */
00525         /* Peripheral clock disable */
00526         __HAL_RCC_ADC1_CLK_DISABLE();
00527
00528         HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
00529                         |GPIO_PIN_4);
00530
00531         /* ADC1 DMA DeInit */
00532         HAL_DMA_DeInit(adcHandle->DMA_Handle);
00533
00534         /* ADC1 interrupt Deinit */
00535         /* USER CODE BEGIN ADC1:ADC_IRQn disable */
00536         /* HAL_NVIC_DisableIRQ(ADC_IRQn); */
00537         /* USER CODE END ADC1:ADC_IRQn disable */
00538
00539         /* USER CODE BEGIN ADC1_MspDeInit 1 */
00540
00541         /* USER CODE END ADC1_MspDeInit 1 */
00542     }
00543
00544     else if(adcHandle->Instance==ADC2)
00545     {
00546         /* USER CODE BEGIN ADC2_MspDeInit 0 */
00547
00548         /* USER CODE END ADC2_MspDeInit 0 */
00549         /* Peripheral clock disable */
00550         __HAL_RCC_ADC2_CLK_DISABLE();
00551
00552         HAL_GPIO_DeInit(GPIOA, GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);
00553
00554         HAL_GPIO_DeInit(GPIOC, GPIO_PIN_4|GPIO_PIN_5);
00555
00556         HAL_GPIO_DeInit(GPIOB, GPIO_PIN_0|GPIO_PIN_1);
00557
00558         /* ADC2 DMA DeInit */
00559         HAL_DMA_DeInit(adcHandle->DMA_Handle);
00560
00561         /* ADC2 interrupt Deinit */
00562         /* USER CODE BEGIN ADC2:ADC_IRQn disable */
00563         /* HAL_NVIC_DisableIRQ(ADC_IRQn); */
00564         /* USER CODE END ADC2:ADC_IRQn disable */
00565
00566         /* USER CODE BEGIN ADC2_MspDeInit 1 */
00567
00568         /* USER CODE END ADC2_MspDeInit 1 */
00569     }
00570
00571     else if(adcHandle->Instance==ADC3)
00572     {
00573         /* USER CODE BEGIN ADC3_MspDeInit 0 */
00574
00575         /* USER CODE END ADC3_MspDeInit 0 */
00576 }
```

```

00598     /* Peripheral clock disable */
00599     __HAL_RCC_ADC3_CLK_DISABLE();
00600
00607     HAL_GPIO_DeInit(GPIOC, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
00608
00609     /* ADC3 DMA DeInit */
00610     HAL_DMA_DeInit(adcHandle->DMA_Handle);
00611
00612     /* ADC3 interrupt Deinit */
00613     /* USER CODE BEGIN ADC3:ADC IRQn disable */
00618     /* HAL_NVIC_DisableIRQ(ADC IRQn); */
00619     /* USER CODE END ADC3:ADC IRQn disable */
00620
00621     /* USER CODE BEGIN ADC3_MspDeInit 1 */
00622
00623     /* USER CODE END ADC3_MspDeInit 1 */
00624 }
00625 }
00626
00627 /* USER CODE BEGIN 1 */
00628 #include "../FreeRTOS/Source\CMSIS_RTOS/cmsis_os.h" // Add at the top of adc.c if not already
00629
00630 #define DEBUG_TOGGLE_LED() HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15)
00631
00632 void StartADCTask(void *argument) {
00633     TickType_t tick_rate = pdMS_TO_TICKS(100); // 100ms cycle
00634     TickType_t last_wake_time = xTaskGetTickCount(); // capture the current tick count as the starting
00635     point
00636     for (;;) {
00637         DEBUG_TOGGLE_LED(); // Optional - shows task is alive
00638         if(HAL_ADC_Start_DMA(&hadc1, (uint32_t *)adc_buf1, 4) != HAL_OK){
00639             int a = 0;
00640         }
00641
00642         if(HAL_ADC_Start_DMA(&hadc2, (uint32_t *)adc_buf2, 7)!= HAL_OK){
00643
00644         }
00645
00646         if(HAL_ADC_Start_DMA(&hadc3, (uint32_t *)adc_buf3, 7)!= HAL_OK){
00647
00648         }
00649
00650         vTaskDelayUntil(&last_wake_time, tick_rate);
00651     }
00652 }
00653
00654 void processADCBuffer(uint8_t adc) {
00655
00656     //adc_buf1[4];
00657
00658     switch(adc) {
00659     case 1:
00660         adcl_APPS1 = adc_buf1[0]; //These are the locations for the sensor inputs for APPS and BPS
00661         adcl_APPSS2 = adc_buf1[1];
00662         adcl_BPS1 = adc_buf1[2];
00663         adcl_BPS2 = adc_buf1[3];
00664         adcl_pack_curr = adc_buf1[4];
00665         break;
00666     case 2:
00667         adc2_steering_pos = adc_buf2[0];
00668         //PA5 is currently spare
00669         adc2_CoolantFlow = adc_buf2[2];
00670         adc2_CoolantTemp1 = adc_buf2[3];
00671         adc2_CoolantTemp2 = adc_buf2[4];
00672         //adc_buf2[5] and adc_buf2[6] are spares for now;
00673         break;
00674     case 3:
00675         adc3_damper_FL = adc_buf3[0]; //Damper pots
00676         adc3_damper_FR = adc_buf3[1];
00677         adc3_damper_RL = adc_buf3[2];
00678         adc3_damper_RR = adc_buf3[3];
00679         break;
00680     default:
00681         break;
00682     }
00683
00684 }
00685
00686
00687 void initADCTask(void) {
00688     BaseType_t result = xTaskCreate(
00689         StartADCTask,           // Task function
00690         "ADC Task",            // Name for debugging
00691         128,                  // Stack size (adjust if needed)
00692         NULL,                 // No args
00693         osPriorityAboveNormal, // Priority

```

```

00694     NULL
00695     );
00696
00697     if (result != pdPASS) {
00698         Error_Handler();
00699     }
00700 }
00701
00702
00703 /* USER CODE END 1 */

```

7.59 Core/Src/bms.c File Reference

```

#include "main.h"
#include "bms.h"
#include "constants.h"
#include "pdu.h"
#include "can.h"
#include "tim.h"
#include "dash.h"
#include "uvfr_utils.h"
Include dependency graph for bms.c:

```

Macros

- `#define bms_settings current_vehicle_settings->bms_settings`

Functions

- `void BMS_msg1 (uv_CAN_msg *msg)`
- `void BMS_msg2 (uv_CAN_msg *msg)`
- `void BMS_Init (void *args)`

Variables

- `bms_settings_t default_bms_settings`
- `TickType_t bms_last_msg_time = 0`
- `uint8_t is_bms_connected = 0`
- `uint16_t packCurrent = 0`
- `uint16_t packVoltage = 0`
- `uint16_t stateOfCharge = 0`
- `uint16_t relayState = 0`
- `uint16_t msg1corrupt = 0`
- `uint16_t packDCL = 0`
- `uint16_t lowTemp = 0`
- `uint16_t highTemp = 0`
- `uint16_t msg2corrupt = 0`

7.59.1 Macro Definition Documentation

7.59.1.1 bms_settings

```
#define bms_settings current_vehicle_settings->bms_settings
```

Definition at line 12 of file [bms.c](#).

Referenced by [BMS_Init\(\)](#).

7.59.2 Function Documentation

7.59.2.1 BMS_Init()

```
void BMS_Init (
    void * args)
```

Definition at line 100 of file [bms.c](#).

References [BMS](#), [BMS_msg1\(\)](#), [BMS_msg2\(\)](#), [bms_settings](#), [CAN_BUS_1](#), [uv_init_task_args::init_info_queue](#), [insertCANMessageHandler\(\)](#), [is_bms_connected](#), [uv_init_task_args::meta_task_handle](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.59.2.2 BMS_msg1()

```
void BMS_msg1 (
    uv_CAN_msg * msg)
```

Definition at line 47 of file [bms.c](#).

References [bms_last_msg_time](#), [uv_CAN_msg::data](#), [is_bms_connected](#), [msg1corrupt](#), [packCurrent](#), [packVoltage](#), [relayState](#), and [stateOfCharge](#).

Referenced by [BMS_Init\(\)](#).

7.59.2.3 BMS_msg2()

```
void BMS_msg2 (
    uv_CAN_msg * msg)
```

Definition at line 75 of file [bms.c](#).

References [bms_last_msg_time](#), [uv_CAN_msg::data](#), [highTemp](#), [is_bms_connected](#), [lowTemp](#), [msg2corrupt](#), and [packDCL](#).

Referenced by [BMS_Init\(\)](#).

7.59.3 Variable Documentation

7.59.3.1 bms_last_msg_time

```
TickType_t bms_last_msg_time = 0
```

Definition at line 31 of file [bms.c](#).

Referenced by [BMS_msg1\(\)](#), and [BMS_msg2\(\)](#).

7.59.3.2 default_bms_settings

```
bms_settings_t default_bms_settings
```

Initial value:

```
= {  
    .BMS_CAN_timeout = 200,  
    .max_cell_temp = 500,  
    .min_cell_temp = -200,  
    .min_soc = 10,  
    .min_cell_voltage = 2.5,  
    .max_cell_voltage = 4.2,  
    .max_pack_voltage = 5000,  
    .max_variance_between_cells = 1  
}
```

Definition at line 14 of file [bms.c](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.59.3.3 highTemp

```
uint16_t highTemp = 0
```

Definition at line 42 of file [bms.c](#).

Referenced by [BMS_msg2\(\)](#).

7.59.3.4 is_bms_connected

```
uint8_t is_bms_connected = 0
```

Definition at line 32 of file [bms.c](#).

Referenced by [BMS_Init\(\)](#), [BMS_msg1\(\)](#), and [BMS_msg2\(\)](#).

7.59.3.5 lowTemp

```
uint16_t lowTemp = 0
```

Definition at line 41 of file [bms.c](#).

Referenced by [BMS_msg2\(\)](#).

7.59.3.6 msg1corrupt

```
uint16_t msg1corrupt = 0
```

Definition at line 38 of file [bms.c](#).

Referenced by [BMS_msg1\(\)](#).

7.59.3.7 msg2corrupt

```
uint16_t msg2corrupt = 0
```

Definition at line 43 of file [bms.c](#).

Referenced by [BMS_msg2\(\)](#).

7.59.3.8 packCurrent

```
uint16_t packCurrent = 0
```

Definition at line 34 of file [bms.c](#).

Referenced by [BMS_msg1\(\)](#).

7.59.3.9 packDCL

```
uint16_t packDCL = 0
```

Definition at line 40 of file [bms.c](#).

Referenced by [BMS_msg2\(\)](#).

7.59.3.10 packVoltage

```
uint16_t packVoltage = 0
```

Definition at line 35 of file [bms.c](#).

Referenced by [BMS_msg1\(\)](#).

7.59.3.11 relayState

```
uint16_t relayState = 0
```

Definition at line 37 of file [bms.c](#).

Referenced by [BMS_msg1\(\)](#).

7.59.3.12 stateOfCharge

```
uint16_t stateOfCharge = 0
```

Definition at line 36 of file [bms.c](#).

Referenced by [BMS_msg1\(\)](#).

7.60 bms.c

[Go to the documentation of this file.](#)

```

00001 // This where the code to handle BMS errors and such will go
00002
00003 #include "main.h"
00004 #include "bms.h"
00005 #include "constants.h"
00006 #include "pdu.h"
00007 #include "can.h"
00008 #include "tim.h"
00009 #include "dash.h"
00010 #include "uvfr_utils.h"
00011
00012 #define bms_settings current_vehicle_settings->bms_settings
00013
00014 bms_settings_t default_bms_settings = {
00015     .BMS_CAN_timeout = 200, //ms
00016     .max_cell_temp = 500,
00017     .min_cell_temp = -200,
00018     .min_soc = 10,
00019     .min_cell_voltage = 2.5,
00020     .max_cell_voltage = 4.2,
00021     .max_pack_voltage = 5000,
00022     .max_variance_between_cells = 1
00023 };
00024
00025
00026
00027 //two IDs for bms can message
00028 // 0x6B0
00029 // 0x6B1
00030
00031 TickType_t bms_last_msg_time = 0;
00032 uint8_t is_bms_connected = 0;
00033
00034 uint16_t packCurrent = 0; // logged
00035 uint16_t packVoltage = 0; // logged
00036 uint16_t stateOfCharge = 0; // used for toggling regen braking / low charge warning
00037 uint16_t relayState = 0;
00038 uint16_t msg1corrupt = 0;
00039
00040 uint16_t packDCL = 0;
00041 uint16_t lowTemp = 0;
00042 uint16_t highTemp = 0;
00043 uint16_t msg2corrupt = 0;
00044 // extern uint16_t (var name) to call outside this file
00045
00046
00047 void BMS_msg1(uv_CAN_msg* msg){ // msg is raw CAN msg, gets processed in voltageCANdata
00048     // msg1 can handle current, voltage, charge, relay, checksum (corruption)
00049     packCurrent = (msg->data[0]«8 | msg->data[1]); // x 0.1A
00050     packVoltage = (msg->data[2]«8 | msg->data[3]); // x 0.1V
00051
00052     stateOfCharge = (msg->data[4])/2; // x2; this is an int
00053
00054     relayState = (msg->data[5]«8 | msg->data[6]);
00055     msg1corrupt = (msg->data[7]);
00056
00057     bms_last_msg_time = xTaskGetTickCount();
00058     is_bms_connected = 1;
00059
00060     if (stateOfCharge < 20) {
00061         // flash low battery warning to screen
00062     }
00063     if (stateOfCharge >= 95) {
00064         // disable regenerative breaking
00065     }
00066     // in all other cases battery should be functioning as normal
00067
00068
00069
00070     // would make sense for voltageCANdata to turn into float
00071     //log voltage data to where, maybe setting global variable
00072 }
00073
00074
00075 void BMS_msg2(uv_CAN_msg* msg){
00076     // msg2 can handle DCL (Max current output), tempurature, checksum
00077     packDCL = (msg->data[0]«8 | msg->data[3]);
00078     lowTemp = (msg->data[4]); // celsius
00079     highTemp = (msg->data[5]); // celsius
00080     msg2corrupt = (msg->data[7]);
00081
00082     bms_last_msg_time = xTaskGetTickCount();

```

```

00083     is_bms_connected = 1;
00084
00085     if (lowTemp <= -5) {
00086         // warning - temp is too low
00087         // stop operability?
00088
00089     }
00090     else if (lowTemp > -5 && highTemp < 55) {
00091         // normal operating temperature range
00092         // drive as normal
00093         if (highTemp >= 55.0) {
00094             // flash warning - approaching max temp
00095             // maybe enable cooling fan/devices here
00096         }
00097     }
00098 }
00099
00100 void BMS_Init(void* args){
00101     uv_init_task_args* params = (uv_init_task_args*) args;
00102
00103     osDelay(200);
00104
00105     uv_init_task_response response = {UV_OK,BMS,0,NULL};
00106
00107     TickType_t bms_init_time = xTaskGetTickCount();
00108     TickType_t init_deadline = bms_init_time + bms_settings->BMS_CAN_timeout;
00109
00110
00111     insertCANMessageHandler(0x6B0, BMS_msg1, CAN_BUS_1);
00112     insertCANMessageHandler(0x6B1, BMS_msg2, CAN_BUS_1);
00113
00114     for(;;){
00115         if(init_deadline < xTaskGetTickCount()){
00116             //BMS not connected lol
00117             //uvPanic("BMS Not Detected", 0);
00118
00119             //TODO re-enable this if for whatever reason we have this thing again
00120
00121             //response.status = UV_ERROR;
00122             break;
00123         }
00124         if(is_bms_connected == 1){
00125             break;
00126         }
00127     }
00128
00129
00130
00131
00132     //Kill yourself
00133
00134
00135     if(xQueueSendToBack(params->init_info_queue,&response,100) != pdPASS){
00136         //OOPS
00137     }
00138
00139     vTaskSuspend(params->meta_task_handle);
00140 }
00141
00142
00143
00144
00145

```

7.61 Core/Src/can.c File Reference

This file provides code for the configuration of the CAN instances.

```
#include "can.h"
#include "constants.h"
#include "imd.h"
#include "motor_controller.h"
#include "dash.h"
#include "bms.h"
#include "pdu.h"
#include "uvfr_utils.h"
```

```
#include "main.h"
#include "stdlib.h"
#include "string.h"
#include "../FreeRTOS/Source/include/task.h"
Include dependency graph for can.c:
```

Data Structures

- struct [CAN_Callback](#)

Macros

- #define [HAL_CAN_ERROR_INVALID_CALLBACK](#) (0x00400000U)
- #define [table_size](#) 128
- #define [table_size_1](#) [table_size](#)
- #define [table_size_2](#) [table_size](#)

Typedefs

- typedef struct CAN_Callback [CAN_Callback](#)

Functions

- void [handleCANbusError](#) (const CAN_HandleTypeDef *hcan, const uint32_t err_to_ignore)
- void [MX_CAN1_Init](#) (void)
- void [MX_CAN2_Init](#) (void)
- void [HAL_CAN_MspInit](#) (CAN_HandleTypeDef *canHandle)
- void [HAL_CAN_MspDelInit](#) (CAN_HandleTypeDef *canHandle)
- void [HAL_CAN_RxFifo0MsgPendingCallback](#) (CAN_HandleTypeDef *hcan)
- void [HAL_CAN_RxFifo1MsgPendingCallback](#) (CAN_HandleTypeDef *hcan)
- unsigned int [generateHash](#) (uint32_t Incoming_CAN_id)
- void [insertCANMessageHandler](#) (uint32_t id, void *handlerfunc, int can_num)

Function to insert an id and function into the lookup table of callback functions.
- void [nuke_hash_table](#) (int can_num)
- [uv_status __uvCANtxCriticalSection](#) ([uv_CAN_msg](#) *tx_msg)
- [uv_status uvSendCanMSG](#) ([uv_CAN_msg](#) *tx_msg)

Function to send CAN message.
- void [CANbusTxSvcDaemon](#) (void *args)

Background task that handles any CAN messages that are being sent.
- void [CANbusRxSvcDaemon](#) (void *args)

Background task that executes the CAN message callback functions.

Variables

- [CAN_Callback CAN_callback_table_1](#) [[table_size_1](#)] = {0}
- [CAN_Callback CAN_callback_table_2](#) [[table_size_2](#)] = {0}
- [SemaphoreHandle_t callback_table_1_mutex](#) = NULL
- [SemaphoreHandle_t callback_table_2_mutex](#) = NULL
- [uint8_t is_can_ok](#) = 1
- [CAN_TxHeaderTypeDef TxHeader2](#)
- [CAN_HandleTypeDef hcan1](#)
- [CAN_HandleTypeDef hcan2](#)

7.61.1 Detailed Description

This file provides code for the configuration of the CAN instances.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [can.c](#).

7.61.2 Macro Definition Documentation

7.61.2.1 HAL_CAN_ERROR_INVALID_CALLBACK

```
#define HAL_CAN_ERROR_INVALID_CALLBACK (0x00400000U)
```

Definition at line [44](#) of file [can.c](#).

Referenced by [handleCANbusError\(\)](#).

7.61.2.2 table_size

```
#define table_size 128
```

Definition at line [51](#) of file [can.c](#).

Referenced by [generateHash\(\)](#), and [nuke_hash_table\(\)](#).

7.61.2.3 table_size_1

```
#define table_size_1 table_size
```

Definition at line [52](#) of file [can.c](#).

7.61.2.4 table_size_2

```
#define table_size_2 table_size
```

Definition at line [53](#) of file [can.c](#).

7.61.3 Typedef Documentation

7.61.3.1 CAN_Callback

```
typedef struct CAN_Callback CAN_Callback
```

7.61.4 Function Documentation

7.61.4.1 __uvCANtxCriticalSection()

```
uv_status __uvCANtxCriticalSection (
    uv_CAN_msg * tx_msg)
```

Definition at line 826 of file [can.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [hcan2](#), [is_can_ok](#), [uv_CAN_msg::msg_id](#), [TxHeader](#), [TxMailbox](#), [UV_CAN_EXTENDED_ID](#), [UV_ERROR](#), and [UV_OK](#).

Referenced by [uvSendCanMSG\(\)](#).

7.61.4.2 CANbusRxSvcDaemon()

```
void CANbusRxSvcDaemon (
    void * args)
```

Background task that executes the CAN message callback functions.

dequeue can message to call function

Definition at line 1045 of file [can.c](#).

References [callback_table_1_mutex](#), [callback_table_2_mutex](#), [uv_task_info::cmd_data](#), [killSelf\(\)](#), [suspendSelf\(\)](#), [uv_task_info::task_handle](#), [UV_KILL_CMD](#), [UV_OK](#), and [UV_SUSPEND_CMD](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.61.4.3 CANbusTxSvcDaemon()

```
void CANbusTxSvcDaemon (
    void * args)
```

Background task that handles any CAN messages that are being sent.

This task sits idle, until the time is right (it receives a notification from the [uvSendCanMSG](#) function) Once this condition has been met, it will actually call the [HAL_CAN_AddTxMessage](#) function. This is a very high priority task, meaning that it will pause whatever other code is going in order to run

dequeue can message to put into can bus

Definition at line 920 of file [can.c](#).

References [CAN_BUS_1](#), [uv_task_info::cmd_data](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [hcan1](#), [hcan2](#), [is_can_ok](#), [killSelf\(\)](#), [uv_CAN_msg::msg_id](#), [suspendSelf\(\)](#), [TxHeader](#), [TxHeader2](#), [TxMailbox](#), [UV_CAN_EXTENDED_ID](#), [UV_KILL_CMD](#), and [UV_SUSPEND_CMD](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.61.4.4 generateHash()

```
unsigned int generateHash (
    uint32_t Incoming_CAN_id)
```

HASH FUNCTION Take a can id and return a "random" hash id The hash id is in range from 0 to table_size The hash id is similar to an array index in its implementation 11 bits in a hash

Definition at line 577 of file [can.c](#).

References [table_size](#).

Referenced by [insertCANMessageHandler\(\)](#).

7.61.4.5 HAL_CAN_MspDeInit()

```
void HAL_CAN_MspDeInit (
    CAN_HandleTypeDef * canHandle)
```

CAN1 GPIO Configuration PB8 -----> CAN1_RX PB9 -----> CAN1_TX

CAN2 GPIO Configuration PB12 -----> CAN2_RX PB13 -----> CAN2_TX

Definition at line 391 of file [can.c](#).

7.61.4.6 HAL_CAN_MspInit()

```
void HAL_CAN_MspInit (
    CAN_HandleTypeDef * canHandle)
```

CAN1 GPIO Configuration PB8 -----> CAN1_RX PB9 -----> CAN1_TX

CAN2 GPIO Configuration PB12 -----> CAN2_RX PB13 -----> CAN2_TX

Definition at line 312 of file [can.c](#).

7.61.4.7 HAL_CAN_RxFifo0MsgPendingCallback()

```
void HAL_CAN_RxFifo0MsgPendingCallback (
    CAN_HandleTypeDef * hcan)
```

Definition at line 456 of file [can.c](#).

References [CAN_BUS_1](#), [CAN_BUS_2](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [Error_Handler\(\)](#), [uv_CAN_msg::flags](#), [hcan1](#), [is_can_ok](#), [uv_CAN_msg::msg_id](#), [RxHeader](#), [RxHeader2](#), and [UV_CAN_EXTENDED_ID](#).

Here is the call graph for this function:

7.61.4.8 HAL_CAN_RxFifo1MsgPendingCallback()

```
void HAL_CAN_RxFifo1MsgPendingCallback (
    CAN_HandleTypeDef * hcan)
```

Definition at line 512 of file [can.c](#).

References [CAN_BUS_1](#), [CAN_BUS_2](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [Error_Handler\(\)](#), [uv_CAN_msg::flags](#), [hcan1](#), [is_can_ok](#), [uv_CAN_msg::msg_id](#), [RxHeader](#), [RxHeader2](#), and [UV_CAN_EXTENDED_ID](#).

Here is the call graph for this function:

7.61.4.9 handleCANbusError()

```
void handleCANbusError (
    const CAN_HandleTypeDef * hcan,
    const uint32_t err_to_ignore)
```

Definition at line 76 of file [can.c](#).

References [HAL_CAN_ERROR_INVALID_CALLBACK](#), and [is_can_ok](#).

Referenced by [main\(\)](#).

7.61.4.10 MX_CAN1_Init()

```
void MX_CAN1_Init (
    void )
```

Definition at line 156 of file [can.c](#).

References [Error_Handler\(\)](#), [hcan1](#), and [TxHeader](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.61.4.11 MX_CAN2_Init()

```
void MX_CAN2_Init (
    void )
```

Definition at line 232 of file [can.c](#).

References [Error_Handler\(\)](#), [hcan2](#), and [TxHeader](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.61.4.12 nuke_hash_table()

```
void nuke_hash_table (
    int can_num)
```

Function to free all malloced memory Index through the hash table and free all the malloced memory at each index

Definition at line 773 of file [can.c](#).

References [CAN_BUS_1](#), [CAN_BUS_2](#), [CAN_callback_table_1](#), [CAN_callback_table_2](#), [CAN_Callback::next](#), and [table_size](#).

7.61.5 Variable Documentation

7.61.5.1 callback_table_1_mutex

```
SemaphoreHandle_t callback_table_1_mutex = NULL
```

Definition at line 70 of file [can.c](#).

Referenced by [CANbusRxSvcDaemon\(\)](#), and [insertCANMessageHandler\(\)](#).

7.61.5.2 callback_table_2_mutex

```
SemaphoreHandle_t callback_table_2_mutex = NULL
```

Definition at line 71 of file [can.c](#).

Referenced by [CANbusRxSvcDaemon\(\)](#), and [insertCANMessageHandler\(\)](#).

7.61.5.3 CAN_callback_table_1

```
CAN_Callback CAN_callback_table_1[table_size_1] = {0}
```

Hash Table To Store CAN Messages Creates a hash table of size table_size and type CAN_Message Initialize all CAN messages in the hash table

Definition at line 66 of file [can.c](#).

Referenced by [insertCANMessageHandler\(\)](#), and [nuke_hash_table\(\)](#).

7.61.5.4 CAN_callback_table_2

```
CAN_Callback CAN_callback_table_2[table_size_2] = {0}
```

Definition at line 68 of file [can.c](#).

Referenced by [insertCANMessageHandler\(\)](#), and [nuke_hash_table\(\)](#).

7.61.5.5 hcan1

CAN_HandleTypeDef hcan1

Definition at line 152 of file [can.c](#).

Referenced by [CAN1_RX0_IRQHandler\(\)](#), [CAN1_RX1_IRQHandler\(\)](#), [CAN1_SCE_IRQHandler\(\)](#), [CAN1_TX_IRQHandler\(\)](#), [CANbusTxSvcDaemon\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), and [MX_CAN1_Init\(\)](#).

7.61.5.6 hcan2

CAN_HandleTypeDef hcan2

Definition at line 153 of file [can.c](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CAN2_RX0_IRQHandler\(\)](#), [CAN2_RX1_IRQHandler\(\)](#), [CAN2_SCE_IRQHandler\(\)](#), [CAN2_TX_IRQHandler\(\)](#), [CANbusTxSvcDaemon\(\)](#), [IMD_Request_Status\(\)](#), [main\(\)](#), [MX_CAN2_Init\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.61.5.7 TxHeader2

CAN_TxHeaderTypeDef TxHeader2 [extern]

Definition at line 5 of file [constants.c](#).

Referenced by [CANbusTxSvcDaemon\(\)](#).

7.62 can.c

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020 /* Includes ----- */
00021 #include "can.h"
00022
00023 /* USER CODE BEGIN 0 */
00024
00030 #include "constants.h"
00031 #include "imd.h"
00032 #include "motor_controller.h"
00033 #include "dash.h"
00034 #include "bms.h"
00035 #include "pdu.h"
00036 #include "uvfr_utils.h"
00037 #include "main.h"
00038 #include "stdlib.h"
00039 #include "string.h"
00040 #include "../FreeRTOS/Source/include/task.h"
00041
00042 //This line holds the entire program together
00043 #ifndef HAL_CAN_ERROR_INVALID_CALLBACK
00044 #define HAL_CAN_ERROR_INVALID_CALLBACK (0x00400000U)
00045 #endif
00046
00047 static QueueHandle_t Tx_msg_queue = NULL;
00048 static QueueHandle_t Rx_msg_queue = NULL;
00049
00050
00051 #define table_size 128
00052 #define table_size_1 table_size //CHANGE THIS TO HOW MANY CAN_MESSAGES YOU NEED TO HANDLE!!!!!
00053 #define table_size_2 table_size

```

```

00054
00055     typedef struct CAN_Callback {
00056         uint32_t CAN_id;
00057         void* function;
00058         struct CAN_Callback* next;
00059     }CAN_Callback;
00060
00061
00062     CAN_Callback CAN_callback_table_1[table_size_1] = {0};
00063
00064     CAN_Callback CAN_callback_table_2[table_size_2] = {0};
00065
00066
00067     SemaphoreHandle_t callback_table_1_mutex = NULL;
00068     SemaphoreHandle_t callback_table_2_mutex = NULL;
00069
00070
00071
00072
00073
00074     uint8_t is_can_ok = 1;
00075
00076     void handleCANbusError(const CAN_HandleTypeDef* hcan, const uint32_t err_to_ignore){
00077         is_can_ok = 0;
00078         if(hcan == NULL){
00079             uvPanic("null can handle",0);
00080             return;
00081         }
00082
00083
00084         uint32_t errcode = HAL_CAN_GetError(hcan);
00085
00086         if(errcode & err_to_ignore){
00087             return;
00088         }
00089
00090         if(errcode == HAL_CAN_ERROR_NONE) {
00091             return;
00092         }else if(errcode & HAL_CAN_ERROR_EWG){ //protocol error
00093             uvPanic("CAN protocol error",0);
00094         }else if(errcode & HAL_CAN_ERROR_EPV){ //passive
00095             uvPanic("CAN passive error",0);
00096         }else if(errcode & HAL_CAN_ERROR_BOF){ //Bus-off error
00097             uvPanic("CAN Bus off error",0);
00098         }else if(errcode & HAL_CAN_ERROR_STF){ //Stuff error
00099             uvPanic("CAN Stuff error",0);
00100         }else if(errcode & HAL_CAN_ERROR_FOR){ //Form error
00101             uvPanic("CAN Form error",0);
00102         }else if(errcode & HAL_CAN_ERROR_ACK){ //Acknowledgement error
00103             uvPanic("CAN Ack error",0);
00104         }else if(errcode & HAL_CAN_ERROR_BR){ //bit recessive
00105             uvPanic("CAN Recessive Bit error",0);
00106         }else if(errcode & HAL_CAN_ERROR_BD){ //bit dominant
00107             uvPanic("CAN Dominant Bit error",0);
00108         }else if(errcode & HAL_CAN_ERROR_CRC){ //cyclic redundancy check
00109             uvPanic("CAN CRC Failed",0);
00110         }else if(errcode & HAL_CAN_ERROR_RX_FVO0){ //overrun rx fifo0
00111             uvPanic("CAN RX FIFO0",0);
00112         }else if(errcode & HAL_CAN_ERROR_RX_FVO1){ //overrun rx fifo1
00113             uvPanic("CAN RX FIFO1",0);
00114         }else if(errcode & HAL_CAN_ERROR_TX_ALST0){ //tx mailbox 0 arbitration lost
00115             uvPanic("CAN0 arbitration",0);
00116         }else if(errcode & HAL_CAN_ERROR_TX_TERR0){ //tx mailbox 0 transmit error
00117             uvPanic("CAN0 transmit err",0);
00118         }else if(errcode & HAL_CAN_ERROR_TX_ALST1){ //tx mailbox 1 arbitration lost
00119             uvPanic("CAN1 arbitration",0);
00120         }else if(errcode & HAL_CAN_ERROR_TX_TERR1){ //tx mailbox 1 transmit error
00121             uvPanic("CAN1 transmit err",0);
00122         }else if(errcode & HAL_CAN_ERROR_TX_ALST2){ //tx mailbox 2 arbitration lost
00123             uvPanic("CAN2 arbitration",0);
00124         }else if(errcode & HAL_CAN_ERROR_TX_TERR2){ //tx mailbox 2 transmit error
00125             uvPanic("CAN2 transmit err",0);
00126         }else if(errcode & HAL_CAN_ERROR_TIMEOUT){ //timeout
00127             uvPanic("CAN timeout",0);
00128         }else if(errcode & HAL_CAN_ERROR_NOT_INITIALIZED){ //is not initialized
00129             uvPanic("CAN not init",0);
00130         }else if(errcode & HAL_CAN_ERROR_NOT_READY){ //Not ready
00131             uvPanic("CAN not ready",0);
00132         }else if(errcode & HAL_CAN_ERROR_NOT_STARTED){ //not started
00133             uvPanic("HAL_CAN_NOT_STARTED",0);
00134         }else if(errcode & HAL_CAN_ERROR_PARAM){ //Param
00135             uvPanic("CAN Param",0);
00136         }else if(errcode & HAL_CAN_ERROR_INVALID_CALLBACK){ //invalid callback
00137             uvPanic("Invalid callback",0);
00138         }else if(errcode & HAL_CAN_ERROR_INTERNAL){ //internal error
00139             uvPanic("HAL_internal",0);
00140         }else{
00141             //no clue how we got here
00142         }
00143     }
00144

```

```
00145
00146 }
00147
00148 extern CAN_TxHeaderTypeDef     TxHeader2;
00149
00150 /* USER CODE END 0 */
00151
00152 CAN_HandleTypeDef hcan1;
00153 CAN_HandleTypeDef hcan2;
00154
00155 /* CAN1 init function */
00156 void MX_CAN1_Init(void)
00157 {
00158
00159     /* USER CODE BEGIN CAN1_Init 0 */
00160
00161     /* USER CODE END CAN1_Init 0 */
00162
00163     /* USER CODE BEGIN CAN1_Init 1 */
00164
00165     /* USER CODE END CAN1_Init 1 */
00166     hcan1.Instance = CAN1;
00167     hcan1.Init.Prescaler = 4;
00168     hcan1.Init.Mode = CAN_MODE_NORMAL;
00169     hcan1.Init.SyncJumpWidth = CAN SJW_1TQ;
00170     hcan1.Init.TimeSeg1 = CAN_BS1_12TQ;
00171     hcan1.Init.TimeSeg2 = CAN_BS2_8TQ;
00172     hcan1.Init.TimeTriggeredMode = DISABLE;
00173     hcan1.Init.AutoBusOff = DISABLE;
00174     hcan1.Init.AutoWakeUp = DISABLE;
00175     hcan1.Init.AutoRetransmission = DISABLE;
00176     hcan1.Init.ReceiveFifoLocked = DISABLE;
00177     hcan1.Init.TransmitFifoPriority = ENABLE;
00178     if (HAL_CAN_Init(&hcan1) != HAL_OK)
00179     {
00180         Error_Handler();
00181     }
00182     /* USER CODE BEGIN CAN1_Init 2 */
00183     // Define the CAN Filter
00184     CAN_FilterTypeDef FilterConfig;
00185
00186
00187     TxHeader.DLC= 1; // Data Length Code
00188     TxHeader.StdId= 0x244; // This is the CAN ID
00189
00190     TxHeader.IDE=CAN_ID_STD; //set identifier to standard
00191     TxHeader.RTR=CAN_RTR_DATA;
00192     TxHeader.ExtId = 0x01;
00193     TxHeader.TransmitGlobalTime = DISABLE;
00194
00195     //filter one (stack light blink)
00196     FilterConfig.FilterFIFOAssignment=CAN_RX_FIFO0; //set fifo assignment
00197     FilterConfig.FilterIdHigh = 0x0000; // filter of zero allows all messages
00198     FilterConfig.FilterIdLow = 0x0000;
00199     FilterConfig.FilterMaskIdHigh = 0x0000;
00200     FilterConfig.FilterMaskIdLow = 0x0000;
00201     FilterConfig.FilterScale=CAN_FILTERSCALE_32BIT; //set filter scale
00202     FilterConfig.FilterActivation=ENABLE;
00203     FilterConfig.FilterBank = 0;
00204     FilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
00205     FilterConfig.SlaveStartFilterBank = 14;
00206     FilterConfig.FilterBank = 0;
00207
00208     // try to configure the filter
00209     if (HAL_CAN_ConfigFilter(&hcan1, &FilterConfig) != HAL_OK)
00210     {
00211         /* Filter configuration Error */
00212         Error_Handler();
00213     }
00214
00215     // Try to set up interrupts for receiving mailbox
00216     if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
00217     {
00218         Error_Handler();
00219     }
00220
00221     // Try to start CAN communication - this is not sending a message, this just initializes it
00222     // If HAL_CAN_Start returns an error, then we want to go into the error handler
00223     if (HAL_CAN_Start(&hcan1) != HAL_OK)
00224     {
00225         /* Start Error */
00226         Error_Handler();
00227     }
00228     /* USER CODE END CAN1_Init 2 */
00229
00230 }
00231 /* CAN2 init function */
```

```

00232 void MX_CAN2_Init(void)
00233 {
00234     /* USER CODE BEGIN CAN2_Init_0 */
00235
00236     /* USER CODE END CAN2_Init_0 */
00237
00238     /* USER CODE BEGIN CAN2_Init_1 */
00239
00240     /* USER CODE END CAN2_Init_1 */
00241     hcan2.Instance = CAN2;
00242     hcan2.Init.Prescaler = 4;
00243     hcan2.Init.Mode = CAN_MODE_NORMAL;
00244     hcan2.Init.SyncJumpWidth = CAN_SJW_1TQ;
00245     hcan2.Init.TimeSeg1 = CAN_BS1_12TQ;
00246     hcan2.Init.TimeSeg2 = CAN_BS2_8TQ;
00247     hcan2.Init.TimeTriggeredMode = DISABLE;
00248     hcan2.Init.AutoBusOff = DISABLE;
00249     hcan2.Init.AutoWakeUp = DISABLE;
00250     hcan2.Init.AutoRetransmission = DISABLE;
00251     hcan2.Init.ReceiveFifoLocked = DISABLE;
00252     hcan2.Init.TransmitFifoPriority = ENABLE;
00253
00254     if (HAL_CAN_Init(&hcan2) != HAL_OK)
00255     {
00256         Error_Handler();
00257     }
00258     /* USER CODE BEGIN CAN2_Init_2 */
00259
00260     // Define the CAN Filter
00261     CAN_FilterTypeDef FilterConfig;
00262
00263     // Set the data length and ID to a temporary value
00264     TxHeader.DLC= 1; // Data Length Code
00265     TxHeader.StdId= 0x244; // This is the CAN ID
00266
00267     TxHeader.IDE=CAN_ID_STD; //set identifier to standard
00268     TxHeader.RTR=CAN_RTR_DATA;
00269     TxHeader.ExtId = 0x01;
00270     TxHeader.TransmitGlobalTime = DISABLE;
00271
00272     //filter one (stack light blink)
00273     FilterConfig.FilterFIFOAssignment=CAN_RX_FIFO0; //set fifo assignment
00274     FilterConfig.FilterIdHigh = 0x0000; // filter of zero allows all messages
00275     FilterConfig.FilterIdLow = 0x0000;
00276     FilterConfig.FilterMaskIdHigh = 0x0000;
00277     FilterConfig.FilterMaskIdLow = 0x0000;
00278     FilterConfig.FilterScale=CAN_FILTERSCALE_32BIT; //set filter scale
00279     FilterConfig.FilterActivation=ENABLE;
00280     FilterConfig.FilterBank = 0;
00281     FilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
00282     FilterConfig.SlaveStartFilterBank = 14;
00283     FilterConfig.FilterBank = 14;
00284
00285     // try to configure the filter
00286     if (HAL_CAN_ConfigFilter(&hcan2, &FilterConfig) != HAL_OK)
00287     {
00288         /* Filter configuration Error */
00289         Error_Handler();
00290     }
00291
00292     // Try to set up interrupts for receiving mailbox
00293     if (HAL_CAN_ActivateNotification(&hcan2, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
00294     {
00295         Error_Handler();
00296     }
00297
00298     // Try to start CAN communication - this is not sending a message, this just initializes it
00299     // If HAL_CAN_Start returns an error, then we want to go into the error handler
00300     if (HAL_CAN_Start(&hcan2) != HAL_OK)
00301     {
00302         /* Start Error */
00303         Error_Handler();
00304     }
00305
00306     /* USER CODE END CAN2_Init_2 */
00307
00308 }
00309
00310 static uint32_t HAL_RCC_CAN1_CLK_ENABLED=0;
00311
00312 void HAL_CAN_MspInit(CAN_HandleTypeDef* canHandle)
00313 {
00314
00315     GPIO_InitTypeDef GPIO_InitStruct = {0};
00316     if(canHandle->Instance==CAN1)
00317     {
00318         /* USER CODE BEGIN CAN1_MspInit_0 */

```

```

00319 /* USER CODE END CAN1_MspInit 0 */
00320 /* CAN1 clock enable */
00321 HAL_RCC_CAN1_CLK_ENABLED++;
00322 if(HAL_RCC_CAN1_CLK_ENABLED==1) {
00323     __HAL_RCC_CAN1_CLK_ENABLE();
00324 }
00325
00326 __HAL_RCC_GPIOB_CLK_ENABLE();
00327 GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9;
00328 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00329 GPIO_InitStruct.Pull = GPIO_NOPULL;
00330 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
00331 GPIO_InitStruct.Alternate = GPIO_AF9_CAN1;
00332 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00333
00334 /* CAN1 interrupt Init */
00335 HAL_NVIC_SetPriority(CAN1_TX_IRQn, 6, 0);
00336 HAL_NVIC_EnableIRQ(CAN1_TX_IRQn);
00337 HAL_NVIC_SetPriority(CAN1_RX0_IRQn, 6, 0);
00338 HAL_NVIC_EnableIRQ(CAN1_RX0_IRQn);
00339 HAL_NVIC_SetPriority(CAN1_RX1_IRQn, 6, 0);
00340 HAL_NVIC_EnableIRQ(CAN1_RX1_IRQn);
00341 HAL_NVIC_SetPriority(CAN1_SCE_IRQn, 6, 0);
00342 HAL_NVIC_EnableIRQ(CAN1_SCE_IRQn);
00343
00344 /* USER CODE BEGIN CAN1_MspInit 1 */
00345
00346 /* USER CODE END CAN1_MspInit 1 */
00347 }
00348 else if(canHandle->Instance==CAN2)
00349 {
00350 /* USER CODE BEGIN CAN2_MspInit 0 */
00351
00352 /* CAN2 clock enable */
00353 HAL_RCC_CAN2_CLK_ENABLE();
00354 HAL_RCC_CAN1_CLK_ENABLED++;
00355 if(HAL_RCC_CAN1_CLK_ENABLED==1) {
00356     __HAL_RCC_CAN1_CLK_ENABLE();
00357 }
00358
00359 __HAL_RCC_GPIOB_CLK_ENABLE();
00360 GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_12|GPIO_PIN_13;
00361 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00362 GPIO_InitStruct.Pull = GPIO_NOPULL;
00363 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
00364 GPIO_InitStruct.Alternate = GPIO_AF9_CAN2;
00365 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00366
00367 /* CAN2 interrupt Init */
00368 HAL_NVIC_SetPriority(CAN2_TX_IRQn, 6, 0);
00369 HAL_NVIC_EnableIRQ(CAN2_TX_IRQn);
00370 HAL_NVIC_SetPriority(CAN2_RX0_IRQn, 6, 0);
00371 HAL_NVIC_EnableIRQ(CAN2_RX0_IRQn);
00372 HAL_NVIC_SetPriority(CAN2_RX1_IRQn, 6, 0);
00373 HAL_NVIC_EnableIRQ(CAN2_RX1_IRQn);
00374 HAL_NVIC_SetPriority(CAN2_SCE_IRQn, 6, 0);
00375 HAL_NVIC_EnableIRQ(CAN2_SCE_IRQn);
00376
00377 /* USER CODE BEGIN CAN2_MspInit 1 */
00378
00379 /* USER CODE END CAN2_MspInit 1 */
00380 }
00381
00382 void HAL_CAN_MspDeInit(CAN_HandleTypeDef* canHandle)
00383 {
00384 if(canHandle->Instance==CAN1)
00385 {
00386 /* USER CODE BEGIN CAN1_MspDeInit 0 */
00387
00388 /* CAN1 clock disable */
00389 HAL_RCC_CAN1_CLK_ENABLED--;
00390 if(HAL_RCC_CAN1_CLK_ENABLED==0) {
00391     __HAL_RCC_CAN1_CLK_DISABLE();
00392 }
00393
00394 HAL_GPIO_DeInit(GPIOB, GPIO_PIN_8|GPIO_PIN_9);
00395
00396 /* CAN1 interrupt Deinit */
00397 HAL_NVIC_DisableIRQ(CAN1_TX_IRQn);
00398 HAL_NVIC_DisableIRQ(CAN1_RX0_IRQn);
00399 HAL_NVIC_DisableIRQ(CAN1_RX1_IRQn);
00400 HAL_NVIC_DisableIRQ(CAN1_SCE_IRQn);
00401
00402 /* USER CODE BEGIN CAN1_MspDeInit 1 */
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
010010
010011
010012
010013
010014
010015
010016
010017
010018
010019
010020
010021
010022
010023
010024
010025
010026
010027
010028
010029
010030
010031
010032
010033
010034
010035
010036
010037
010038
010039
010040
010041
010042
010043
010044
010045
010046
010047
010048
010049
010050
010051
010052
010053
010054
010055
010056
010057
010058
010059
010060
010061
010062
010063
010064
010065
010066
010067
010068
010069
010070
010071
010072
010073
010074
010075
010076
010077
010078
010079
010080
010081
010082
010083
010084
010085
010086
010087
010088
010089
010090
010091
010092
010093
010094
010095
010096
010097
010098
010099
0100100
0100101
0100102
0100103
0100104
0100105
0100106
0100107
0100108
0100109
0100110
0100111
0100112
0100113
0100114
0100115
0100116
0100117
0100118
0100119
0100120
0100121
0100122
0100123
0100124
0100125
0100126
0100127
0100128
0100129
0100130
0100131
0100132
0100133
0100134
0100135
0100136
0100137
0100138
0100139
0100140
0100141
0100142
0100143
0100144
0100145
0100146
0100147
0100148
0100149
0100150
0100151
0100152
0100153
0100154
0100155
0100156
0100157
0100158
0100159
0100160
0100161
0100162
0100163
0100164
0100165
0100166
0100167
0100168
0100169
0100170
0100171
0100172
0100173
0100174
0100175
0100176
0100177
0100178
0100179
0100180
0100181
0100182
0100183
0100184
0100185
0100186
0100187
0100188
0100189
0100190
0100191
0100192
0100193
0100194
0100195
0100196
0100197
0100198
0100199
0100200
0100201
0100202
0100203
0100204
0100205
0100206
0100207
0100208
0100209
0100210
0100211
0100212
0100213
0100214
0100215
0100216
0100217
0100218
0100219
0100220
0100221
0100222
0100223
0100224
0100225
0100226
0100227
0100228
0100229
0100230
0100231
0100232
0100233
0100234
0100235
0100236
0100237
0100238
0100239
0100240
0100241
0100242
0100243
0100244
0100245
0100246
0100247
0100248
0100249
0100250
0100251
0100252
0100253
0100254
0100255
0100256
0100257
0100258
0100259
0100260
0100261
0100262
0100263
0100264
0100265
0100266
0100267
0100268
0100269
0100270
0100271
0100272
0100273
0100274
0100275
0100276
0100277
0100278
0100279
0100280
0100281
0100282
0100283
0100284
0100285
0100286
0100287
0100288
0100289
0100290
0100291
0100292
0100293
0100294
0100295
0100296
0100297
0100298
0100299
0100300
0100301
0100302
0100303
0100304
0100305
0100306
0100307
0100308
0100309
0100310
0100311
0100312
0100313
0100314
0100315
0100316
0100317
0100318
0100319
0100320
0100321
0100322
0100323
0100324
0100325
0100326
0100327
0100328
0100329
0100330
0100331
0100332
0100333
0100334
0100335
0100336
0100337
0100338
0100339
0100340
0100341
0100342
0100343
0100344
0100345
0100346
0100347
0100348
0100349
0100350
0100351
0100352
0100353
0100354
0100355
0100356
0100357
0100358
0100359
0100360
0100361
0100362
0100363
0100364
0100365
0100366
0100367
0100368
0100369
0100370
0100371
0100372
0100373
0100374
0100375
0100376
0100377
0100378
0100379
0100380
0100381
0100382
0100383
0100384
0100385
0100386
0100387
0100388
0100389
0100390
0100391
0100392
0100393
0100394
0100395
0100396
0100397
0100398
0100399
0100400
0100401
0100402
0100403
0100404
0100405
0100406
0100407
0100408
0100409
0100410
0100411
0100412
0100413
0100414
0100415
0100416
0100417
0100418
0100419
0100420
0100421
0100422
0100423
0100424
0100425
0100426
0100427
0100428
0100429
0100430
0100431
0100432
0100433
0100434
0100435
0100436
0100437
0100438
0100439
0100440
0100441
0100442
0100443
0100444
0100445
0100446
0100447
0100448
0100449
0100450
0100451
0100452
0100453
0100454
0100455
0100456
0100457
0100458
0100459
0100460
0100461
0100462
0100463
0100464
0100465
0100466
0100467
0100468
0100469
0100470
0100471
0100472
0100473
0100474
0100475
0100476
0100477
0100478
0100479
0100480
0100481
0100482
0100483
0100484
0100485
0100486
0100487
0100488
0100489
0100490
0100491
0100492
0100493
0100494
0100495
0100496
0100497
0100498
0100499
0100500
0100501
0100502
0100503
0100504
0100505
0100506
0100507
0100508
0100509
0100510
0100511
0100512
0100513
0100514
0100515
0100516
0100517
0100518
0100519
0100520
0100521
0100522
0100523
0100524
0100525
0100526
0100527
0100528
0100529
0100530
0100531
0100532
0100533
0100534
0100535
0100536
0100537
0100538
0100539
0100540
0100541
0100542
0100543
0100544
0100545
0100546
0100547
0100548
0100549
0100550
0100551
0100552
0100553
0100554
0100555
0100556
0100557
0100558
0100559
0100560
0100561
0100562
0100563
0100564
0100565
0100566
0100567
0100568
0100569
0100570
0100571
0100572
0100573
0100574
0100575
0100576
0100577
0100578
0100579
0100580
0100581
0100582
0100583
0100584
0100585
0100586
0100587
0100588
0100589
0100590
0100591
0100592
0100593
0100594
0100595
0100596
0100597
0100598
0100599
0100600
0100601
0100602
0100603
0100604
0100605
0100606
0100607
0100608
0100609
0100610
0100611
0100612
0100613
0100614
0100615
0100616
0100617
0100618
0100619
0100620
0100621
0100622
0100623
0100624
0100625
0100626
0100627
0100628
0100629
0100630
0100631
0100632
0100633
0100634
0100635
0100636
0100637
0100638
0100639
0100640
0100641
0100642
0100643
0100644
0100645
0100646
0100647
0100648
0100649
0100650
0100651
0100652
0100653
0100654
0100655
0100656
0100657
0100658
0100659
0100660
0100661
0100662
0100663
0100664
0100665
0100666
0100667
0100668
0100669
0100670
0100671
0100672
0100673
0100674
0100675
0100676
0100677
0100678
0100679
0100680
0100681
0100682
0100683
0100684
0100685
0100686
0100687
0100688
0100689
0100690
0100691
0100692
0100693
0100694
0100695
0100696
0100697
0100698
0100699
0100700
0100701
0100702
0100703
0100704
0100705
0100706
0100707
0100708
0100709
0100710
0100711
0100712
0100713
0100714
0100715
0100716
0100717
0100718
0100719
0100720
0100721
0100722
0100723
0100724
0100725
0100726
0100727
0100728
0100729
0100730
0100731
0100732
0100733
0100734
0100735
0100736
0100737
0100738
0100739
0100740
0100741
0100742
0100743
0100744
0100745
0100746
0100747
0100748
0100749
0100750
0100751
0100752
0100753
0100754
0100755
0100756
0100757
0100758
0100759
0100760
0100761
0100762
0100763
0100764
0100765
0100766
0100767
0100768
0100769
0100770
0100771
0100772
0100773
0100774
0100775
0100776
0100777
0100778
0100779
0100780
0100781
0100782
0100783
0100784
0100785
0100786
0100787
0100788
0100789
0100790
0100791
0100792
0100793
0100794
0100795
0100796
0100797
0100798
0100799
0100800
0100801
0100802
0100803
010
```

```

00418 /* USER CODE END CAN1_MspDeInit 1 */
00419 }
00420 else if(canHandle->Instance==CAN2)
00421 {
00422 /* USER CODE BEGIN CAN2_MspDeInit 0 */
00423
00424 /* USER CODE END CAN2_MspDeInit 0 */
00425 /* Peripheral clock disable */
00426 __HAL_RCC_CAN2_CLK_DISABLE();
00427 HAL_RCC_CAN1_CLK_ENABLED--;
00428 if(HAL_RCC_CAN1_CLK_ENABLED==0) {
00429 __HAL_RCC_CAN1_CLK_DISABLE();
00430 }
00431
00432 HAL_GPIO_DeInit(GPIOB, GPIO_PIN_12|GPIO_PIN_13);
00433
00434 /* CAN2 interrupt Deinit */
00435 HAL_NVIC_DisableIRQ(CAN2_TX_IRQn);
00436 HAL_NVIC_DisableIRQ(CAN2_RX0_IRQn);
00437 HAL_NVIC_DisableIRQ(CAN2_RX1_IRQn);
00438 HAL_NVIC_DisableIRQ(CAN2_SCE_IRQn);
00439 /* USER CODE BEGIN CAN2_MspDeInit 1 */
00440
00441 /* USER CODE END CAN2_MspDeInit 1 */
00442 }
00443 }
00444
00445 /* USER CODE BEGIN 1 */
00446
00447 // When a CAN message comes, the interrupt will call this function
00448 // We need to figure out what device sent it, what the data is, and handle it appropriately
00449 //^this is wrong, we need to only put the message in the queue
00450
00451
00452 void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan){
00453     uv_CAN_msg tmp;
00454     BaseType_t was_higher_priority_task_woken;
00455     CAN_RxHeaderTypeDef* pHeader;
00456
00457     uint8_t bus;
00458
00459     if(hcan == &hcan1){
00460         bus = CAN_BUS_1;
00461         pHeader = &RxHeader;
00462         tmp.flags = bus;
00463     }else{
00464         bus = CAN_BUS_2;
00465         pHeader = &RxHeader2;
00466     }
00467     if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, pHeader, tmp.data) != HAL_OK){
00468         Error_Handler();
00469         is_can_ok = 0;
00470     }
00471
00472     if(Rx_msg_queue == NULL){
00473         is_can_ok = 0;
00474         return; //RxDaemon not active yet
00475     }
00476
00477     tmp.flags = bus;
00478
00479 //uint8_t Data[8] = {0};
00480 //int CAN_ID = 0;
00481 //int DLC = 0;
00482
00483 // Extract the ID
00484 if (RxHeader.IDE == CAN_ID_STD){
00485     tmp.msg_id = pHeader->StdId;
00486 }else if (pHeader->IDE == CAN_ID_EXT){
00487     tmp.msg_id = pHeader->ExtId;
00488     tmp.flags |= UV_CAN_EXTENDED_ID;
00489 }else{
00490     //How did we get here?
00491 }
00492
00493 //
00494 //
00495 // // Extract the data length
00496 tmp.dlc = RxHeader.DLC; // Data Length Code
00497
00498 //TANNER AND FLO CALL YOUR FUNCTION HERE TO DO STUFF
00499 xQueueSendFromISR( Rx_msg_queue, &tmp, &was_higher_priority_task_woken );
00500
00501 if(was_higher_priority_task_woken == pdTRUE){
00502     taskYIELD();
00503 }
00504
00505 }
00506
00507 }
00508

```

```
00509 }
00510
00511 // Here is where the second mailbox ISR would live
00512 void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan){
00513     // do something
00514     uv_CAN_msg tmp;
00515     BaseType_t was_higher_priority_task_woken;
00516     CAN_RxHeaderTypeDef* pHeader;
00517
00518     uint8_t bus;
00519
00520     if(hcan == &hcan1){
00521         bus = CAN_BUS_1;
00522         pHeader = &RxHeader;
00523     }else{
00524         bus = CAN_BUS_2;
00525         pHeader = &RxHeader2;
00526     }
00527     if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, pHeader, tmp.data) != HAL_OK){
00528         Error_Handler();
00529         is_can_ok = 0;
00530     }
00531
00532     if(Rx_msg_queue == NULL){
00533         is_can_ok = 0;
00534         return; //RxDaemon not active yet
00535     }
00536
00537 //uint8_t Data[8] = {0};
00538 //int CAN_ID = 0;
00539 //int DLC = 0;
00540
00541     tmp.flags = bus;
00542
00543     // Extract the ID
00544     if (RxHeader.IDE == CAN_ID_STD){
00545         tmp.msg_id = pHeader->StdId;
00546     }else if (pHeader->IDE == CAN_ID_EXT){
00547         tmp.msg_id = pHeader->ExtId;
00548         tmp.flags |= UV_CAN_EXTENDED_ID;
00549     }else{
00550         //How did we get here?
00551     }
00552
00553 /**
00554 /**
00555 // // Extract the data length
00556 tmp.dlc = RxHeader.DLC; // Data Length Code
00557
00558 //TANNER AND FLO CALL YOUR FUNCTION HERE TO DO STUFF
00559 xQueueSendFromISR( Rx_msg_queue, &tmp, &was_higher_priority_task_woken );
00560
00561     if(was_higher_priority_task_woken == pdTRUE){
00562         taskYIELD();
00563     }
00564
00565 }
00566 // CAN_Messages_Tanner
00567
00568
00569
00570
00571 unsigned int generateHash(uint32_t Incoming_CAN_id) {
00572     unsigned int hash = 0;
00573     uint32_t id = Incoming_CAN_id;
00574
00575     return hash = ((id >> 8) ^ (id >> 4) ^ id) % table_size;
00576 }
00577
00578
00579
00580
00581
00582 }
00583
00584
00585
00586 static inline uv_status callFunctionFromCANid(uv_CAN_msg* msg) {
00587     unsigned int index = generateHash(msg->msg_id);
00588
00589
00590     if((msg->flags) & CAN_BUS_1){
00591
00592
00593         CAN_Callback* current = &CAN_callback_table_1[index]; //getting hash function and checking table
00594         entry
00595
00596         while (current != NULL) {
00597             if (current->CAN_id == msg->msg_id) {//if the ID matches, execute, else keep going
00598                 void (*function_ptr)(uv_CAN_msg* msg) = (void (*) (uv_CAN_msg*))current->function;
00599
00600                 if (function_ptr != NULL) {
00601                     function_ptr(msg);
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263

```

```

00608         return UV_OK;
00609     }else{
00610         is_can_ok = 0;
00611         return UV_ERROR;
00612     }
00613 }
00614 current = current->next;
00615 }
00616
00617 }else{
00618
00619
00620     CAN_Callback* current = &CAN_callback_table_2[index]; //getting hash function and checking table
00621     entry
00622     while (current != NULL) {
00623         if (current->CAN_id == msg->msg_id) { //if the ID matches, execute, else keep going
00624             void (*function_ptr)(uv_CAN_msg* msg) = (void (*) (uv_CAN_msg*))current->function;
00625
00626             if (function_ptr != NULL) {
00627                 function_ptr(msg);
00628                 return UV_OK;
00629             }else{
00630                 is_can_ok = 0;
00631                 return UV_ERROR;
00632             }
00633         }
00634         current = current->next;
00635     }
00636
00637 }
00638
00639
00640     return UV_WARNING;
00641 }
00642
00643
00645 void insertCANMessageHandler(uint32_t id, void* handlerfunc, int can_num) {
00646
00647     unsigned int index = generateHash(id);
00648
00649     if(can_num == CAN_BUS_1){//insert into CAN 1
00650
00651         if(callback_table_1_mutex != NULL){
00652             if(xSemaphoreTake(callback_table_1_mutex,10) == pdTRUE){
00653
00654                 if(callback_table_1[index].CAN_id == 0){ //This means the hash entry is empty and can now
00655                     be used, since 0 is not a real CAN id
00656                     CAN_callback_table_1[index].CAN_id = id;
00657                     CAN_callback_table_1[index].function = handlerfunc;
00658                     CAN_callback_table_1[index].next = NULL;
00659                     if(callback_table_1_mutex != NULL){xSemaphoreGive(callback_table_1_mutex);}
00660                     return;
00661
00662             }else{
00663                 return UV_ERROR;
00664             }
00665         }
00666
00667     }
00668
00669
00670         if(CAN_callback_table_1[index].CAN_id == 0){ //This means the hash entry is empty and can now
00671             be used, since 0 is not a real CAN id
00672             CAN_callback_table_1[index].CAN_id = id;
00673             CAN_callback_table_1[index].function = handlerfunc;
00674             CAN_callback_table_1[index].next = NULL;
00675             if(callback_table_1_mutex != NULL){xSemaphoreGive(callback_table_1_mutex);}
00676             return;
00677
00678         if(CAN_callback_table_1[index].CAN_id == id){ //You are editing a duplicate, overwrite it
00679             CAN_callback_table_1[index].CAN_id = id;
00680             CAN_callback_table_1[index].function = handlerfunc;
00681             if(callback_table_1_mutex != NULL){xSemaphoreGive(callback_table_1_mutex);}
00682             return;
00683
00684         }
00685
00686         CAN_Callback* temp = &CAN_callback_table_1[index]; //if we are here: The table entry is not
00687         empty, but is not the id we are looking for
00688         while(temp->next != NULL){
00689             temp = temp->next;
00690             if(temp->CAN_id == id){
00691                 temp->CAN_id = id;
00692                 temp->function = handlerfunc;
00693                 if(callback_table_1_mutex != NULL){xSemaphoreGive(callback_table_1_mutex);}
00694             }
00695         }
00696
00697         temp->next = uvMalloc(sizeof(CAN_Callback)); //reaching this point means temp->next == NULL
00698         if(temp->next == NULL){
00699             if(callback_table_1_mutex != NULL){xSemaphoreGive(callback_table_1_mutex);}
00700             return;
00701
00702     }
00703 }
```

```
00700     } else{
00701         temp = temp->next;
00702         temp->next = NULL;
00703         temp->CAN_id = id;
00704         temp->function = handlerfunc;
00705     }
00706
00707     if(callback_table_1_mutex != NULL) {xSemaphoreGive(callback_table_1_mutex);}
00708
00709
00710
00711 } else {//insert into CAN 2
00712
00713     if(callback_table_2_mutex != NULL){
00714         if(xSemaphoreTake(callback_table_2_mutex,10) == pdTRUE){
00715
00716             } else{
00717                 return UV_ERROR;
00718             }
00719         }
00720     }
00721
00722
00723     if(CAN_callback_table_2[index].CAN_id == 0){ //This means the hash entry is empty and can now
00724     be used, since 0 is not a real CAN id
00725         CAN_callback_table_2[index].CAN_id = id;
00726         CAN_callback_table_2[index].function = handlerfunc;
00727         CAN_callback_table_2[index].next = NULL;
00728         if(callback_table_2_mutex != NULL) {xSemaphoreGive(callback_table_2_mutex);}
00729         return;
00730     }
00731     if(CAN_callback_table_2[index].CAN_id == id){ //You are editing a duplicate, overwrite it
00732         CAN_callback_table_2[index].CAN_id = id;
00733         CAN_callback_table_2[index].function = handlerfunc;
00734         if(callback_table_2_mutex != NULL) {xSemaphoreGive(callback_table_2_mutex);}
00735         return;
00736     }
00737
00738     CAN_Callback* temp = &CAN_callback_table_2[index]; //if we are here: The table entry is not
00739     empty, but is not the id we are looking for
00740     while(temp->next != NULL){
00741         temp = temp->next;
00742         if(temp->CAN_id == id){
00743             temp->CAN_id = id;
00744             temp->function = handlerfunc;
00745             if(callback_table_2_mutex != NULL) {xSemaphoreGive(callback_table_2_mutex);}
00746             return;
00747         }
00748
00749     temp->next = uvMalloc(sizeof(CAN_Callback)); //reaching this point means temp->next == NULL
00750     if(temp->next == NULL){
00751         if(callback_table_2_mutex != NULL) {xSemaphoreGive(callback_table_2_mutex);}
00752         return;
00753     }
00754     temp = temp->next;
00755     temp->next = NULL;
00756     temp->CAN_id = id;
00757     temp->function = handlerfunc;
00758
00759     if(callback_table_2_mutex != NULL) {xSemaphoreGive(callback_table_2_mutex);}
00760
00761 }
00762
00763
00764
00765
00766
00767 }
00768
00769
00770 void nuke_hash_table(int can_num) {
00771
00772     if(can_num == CAN_BUS_1){//insert into CAN 1
00773
00774     CAN_Callback* temp;
00775
00776     for (int i = 0; i < table_size; i++) {
00777         temp = CAN_callback_table_1 + i*sizeof(CAN_Callback);
00778         if(temp->next != NULL){
00779             temp = temp->next;
00780             CAN_Callback tmp2;
00781             while(temp != NULL){
00782                 tmp2 = temp->next;
00783                 uvFree(temp);
00784             }
00785         }
00786     }
00787 }
```

```

00788             temp = tmp2;
00789         }
00790     }
00791 }
00792
00793 (void)memset(CAN_callback_table_1,0,table_size*sizeof(CAN_Callback)); //set the table to all 0s
00794
00795
00796
00797 }else if (can_num == CAN_BUS_2){//insert into CAN 2
00798
00799
00800
00801     CAN_Callback* temp;
00802
00803     for (int i = 0; i < table_size; i++) {
00804         temp = CAN_callback_table_2 + i*sizeof(CAN_Callback);
00805         if(temp->next != NULL) {
00806             temp = temp->next;
00807             CAN_Callback* tmp2;
00808             while(temp != NULL) {
00809                 tmp2 = temp->next;
00810                 uvFree(temp);
00811                 temp = tmp2;
00812             }
00813         }
00814     }
00815
00816 (void)memset(CAN_callback_table_2,0,table_size*sizeof(CAN_Callback)); //set the table to all 0s
00817
00818
00819
00820 }
00821
00822 }
00823
00824
00825
00826 uv_status __uvCANtxCriticalSection(uv_CAN_msg* tx_msg){
00827     if(tx_msg == NULL){
00828         uvPanic("cannot send null CAN msg",0);
00829     }
00830
00831     if((tx_msg->flags) & UV_CAN_EXTENDED_ID){
00832         TxHeader.IDE = CAN_ID_EXT;
00833         TxHeader.ExtId = tx_msg->msg_id;
00834     }else{
00835         TxHeader.IDE = CAN_ID_STD;
00836         TxHeader.StdId = tx_msg->msg_id;
00837     }
00838
00839     TxHeader.DLC = tx_msg->dlc;
00840
00841
00842     taskENTER_CRITICAL();
00843     if (HAL_CAN_AddTxMessage(&hcan2, &TxHeader, tx_msg->data, &TxMailbox) != HAL_OK){
00844         /* Transmission request Error */
00845         taskEXIT_CRITICAL();
00846         is_can_ok = 0;
00847         uvPanic("Unable to Transmit CAN msg",0);
00848         return UV_ERROR;
00849     }else{
00850         taskEXIT_CRITICAL();
00851     }
00852     return UV_OK;
00853 }
00854
00855
00856 uv_status uvSendCanMSG(uv_CAN_msg* tx_msg){
00857
00858     //static TaskHandle_t can_tx_daemon_handle = NULL;
00859     //static uv_task_id can_tx_daemon_task_id;
00860
00861
00862     if(tx_msg == NULL){
00863         is_can_ok = 0;
00864         return UV_ERROR;
00865     }
00866
00867     uint32_t is_isr = 0;
00868     //Special precautions need to be taken if you do this from an interrupt
00869     //__ASM volatile ("MRS %0, ipsr" : "=r" (is_isr) );
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882

```

```

00883 //BaseType_t higher_priority_task_woken = pdFALSE;
00884 if(Tx_msg_queue != NULL) {
00885     if(!is_isr) {
00886         if(xQueueSendToBack(Tx_msg_queue,tx_msg,0) != pdPASS) {
00887             uvPanic("couldnt enqueue CAN message",0);
00888         }else{
00889             return UV_OK;
00890         }
00891         is_can_ok = 0;
00892         return UV_ERROR;
00893     }else{
00894         if(xQueueSendToBackFromISR(Tx_msg_queue,tx_msg,0) != pdPASS) {
00895             is_can_ok = 0;
00896             uvPanic("couldnt enqueue CAN message",0);
00897         }else{
00898             return UV_OK;
00899         }
00900         is_can_ok = 0;
00901         return UV_ERROR;
00902     }
00903 }else{
00904     if(__uvCANtxCriticalSection(tx_msg) !=UV_OK) {
00905         is_can_ok = 0;
00906         return UV_ERROR;
00907     }
00908 }
00909 return UV_OK;
00910 }
00911
00920 void CANbusTxSvcDaemon(void* args){
00921     uv_task_info* params = (uv_task_info*) args;
00922     //CAN_TxHeaderTypeDef tx_header;
00923
00924     Tx_msg_queue = xQueueCreate(8,sizeof(uv_CAN_msg));
00925
00926
00927     //BaseType_t retval;
00928
00929     uv_CAN_msg* tx_msg = uvMalloc(sizeof(uv_CAN_msg));
00930
00931     //tx_header.TransmitGlobalTime = DISABLE;
00932
00933
00934     BaseType_t result;
00935     //uint32_t notif_val = 0;
00936     for(;;{
00937
00938         result = xQueueReceive(Tx_msg_queue,tx_msg,20);
00939
00940         //tx_msg is uv_can_msg
00941
00942         if(result == pdTRUE) {
00943
00944             if(tx_msg == NULL) {
00945                 uvPanic("cannot send null CAN msg",0);
00946             }
00947
00948
00949
00950
00951
00952
00953     TickType_t attempt_time1 = xTaskGetTickCount();
00954     TickType_t attempt_time2 = attempt_time1;
00955
00956
00957
00958
00959     if((tx_msg->flags) & CAN_BUS_1){
00960
00961         TxHeader.DLC = tx_msg->dlc;
00962
00963         if((tx_msg->flags) & UV_CAN_EXTENDED_ID){ // edited UV_CAN_EXTENDED_ID to both CAN 1 and
2
00964             TxHeader.IDE = CAN_ID_EXT;
00965             TxHeader.ExtId = tx_msg->msg_id;
00966         }else{
00967             TxHeader.IDE = CAN_ID_STD;
00968             TxHeader.StdId = tx_msg->msg_id;
00969         }
00970
00971         while(HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) == 0){
00972             if(xTaskGetTickCount() - attempt_time1 >= 2){
00973                 is_can_ok = 0;
00974                 uvPanic("Unable to Transmit CAN msg",0);
00975
00976                 if (CAN1->ESR & CAN_ESR_BOFF) {

```

```

00977                         // Bus-off condition
00978                         break;
00979                     }
00980
00981                         break;
00982                     }
00983                 }
00984             }
00985
00986             if (HAL_CAN_AddTxMessage(&hcan1, &TxHeader, tx_msg->data, &TxMailbox) != HAL_OK) {
00987             /* Transmission request Error */
00988             is_can_ok = 0;
00989             uvPanic("Unable to Transmit CAN msg",0);
00990         }
00991
00992
00993     }else{
00994
00995         TxHeader2.DLC = tx_msg->dlc;
00996
00997         if((tx_msg->flags)& UV_CAN_EXTENDED_ID){// edited UV_CAN_EXTENDED_ID to both CAN 1 and
2
00999             TxHeader2.IDE = CAN_ID_EXT;
01000             TxHeader2.ExtId = tx_msg->msg_id;
01001         }else{
01002             TxHeader2.IDE = CAN_ID_STD;
01003             TxHeader2.StdId = tx_msg->msg_id;
01004         }
01005
01006         while(HAL_CAN_GetTxMailboxesFreeLevel(&hcan2) == 0){
01007             if(xTaskGetTickCount() - attempt_time2 >= 2){
01008                 is_can_ok = 0;
01009                 uvPanic("Unable to Transmit CAN msg",0);
01010                 break;
01011             }
01012         }
01013
01014         if (HAL_CAN_AddTxMessage(&hcan2, &TxHeader2, tx_msg->data, &TxMailbox) != HAL_OK) {
01015             /* Transmission request Error */
01016             is_can_ok = 0;
01017             uvPanic("Unable to Transmit CAN msg",0);
01018         }
01019
01020     }
01021
01022 }
01023
01024
01025     if(params->cmd_data == UV_KILL_CMD){
01026         QueueHandle_t tmpqueue = Tx_msg_queue;
01027         Tx_msg_queue = NULL;
01028         vQueueDelete(tmpqueue);
01029
01030         killSelf(params);
01031     } else if(params->cmd_data == UV_SUSPEND_CMD){
01032         suspendSelf(params);
01033     }
01034
01035 //main for loop
01036 }
01037
01038
01045 void CANbusRxSvcDaemon(void* args){
01046     uv_task_info* params = (uv_task_info*) args;
01047
01048     uv_CAN_msg tmp;
01049
01050     BaseType_t retval;
01051     uv_status func_status;
01052
01053     Rx_msg_queue = xQueueCreate(8,sizeof(uv_CAN_msg));
01054
01055     callback_table_1_mutex = xSemaphoreCreateMutex();
01056     callback_table_2_mutex = xSemaphoreCreateMutex();
01057
01058
01059     for(;;){
01060         retval = xQueueReceive(Rx_msg_queue,&tmp,10);
01061         if(retval == pdTRUE){
01062             func_status = callFunctionFromCANid(&tmp);
01063             if(func_status != UV_OK){
01064
01065             }
01066         }
01067     if(params->cmd_data == UV_KILL_CMD){

```

```

01069     QueueHandle_t tmpqueue = Rx_msg_queue;
01070     Rx_msg_queue = NULL;
01071     vQueueDelete(tmpqueue);
01072
01073     killSelf(params);
01074 } else if(params->cmd_data == UV_SUSPEND_CMD) {
01075     suspendSelf(params);
01076 }
01077 }
01078
01079 vQueueDelete(Rx_msg_queue);
01080 Rx_msg_queue = NULL;
01081 vTaskDelete(params->task_handle);
01082
01083 }
01084
01085 /* USER CODE END 1 */

```

7.63 Core/Src/constants.c File Reference

```
#include "main.h"
Include dependency graph for constants.c:
```

Variables

- CAN_TxHeaderTypeDef [TxHeader](#)
- CAN_TxHeaderTypeDef [TxHeader2](#)
- CAN_RxHeaderTypeDef [RxHeader](#)
- CAN_RxHeaderTypeDef [RxHeader2](#)
- uint8_t [TxData](#) [8]
- uint32_t [TxMailbox](#)
- uint8_t [RxData](#) [8]

7.63.1 Variable Documentation

7.63.1.1 RxData

```
uint8_t RxData[8]
```

Definition at line 12 of file [constants.c](#).

7.63.1.2 RxHeader

```
CAN_RxHeaderTypeDef RxHeader
```

Definition at line 6 of file [constants.c](#).

Referenced by [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), and [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#).

7.63.1.3 RxHeader2

```
CAN_RxHeaderTypeDef RxHeader2
```

Definition at line 7 of file [constants.c](#).

Referenced by [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), and [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#).

7.63.1.4 TxData

```
uint8_t TxData[8]
```

Definition at line 10 of file [constants.c](#).

Referenced by [IMD_Request_Status\(\)](#), [main\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.63.1.5 TxHeader

```
CAN_TxHeaderTypeDef TxHeader
```

Definition at line 4 of file [constants.c](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [IMD_Request_Status\(\)](#), [main\(\)](#), [MX_CAN1_Init\(\)](#), [MX_CAN2_Init\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.63.1.6 TxHeader2

```
CAN_TxHeaderTypeDef TxHeader2
```

Definition at line 5 of file [constants.c](#).

Referenced by [CANbusTxSvcDaemon\(\)](#).

7.63.1.7 TxMailbox

```
uint32_t TxMailbox
```

Definition at line 11 of file [constants.c](#).

Referenced by [__uvCANtxCriticalSection\(\)](#), [CANbusTxSvcDaemon\(\)](#), [IMD_Request_Status\(\)](#), [main\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.64 constants.c

[Go to the documentation of this file.](#)

```
00001
00002 #include "main.h"
00003
00004 CAN_TxHeaderTypeDef TxHeader;
00005 CAN_TxHeaderTypeDef TxHeader2;
00006 CAN_RxHeaderTypeDef RxHeader;
00007 CAN_RxHeaderTypeDef RxHeader2;
00008
00009
00010 uint8_t TxData[8];
00011 uint32_t TxMailbox;
00012 uint8_t RxData[8];
00013
00014
00015
```

7.65 Core/Src/daq.c File Reference

```
#include "uvfr_utils.h"
#include "daq.h"
#include "stm32f4xx_hal_conf.h"
#include "stm32f407xx.h"
#include "stm32f4xx_hal.h"
#include "stm32f4xx_hal_adc.h"
#include "stm32f4xx_hal_rcc.h"
Include dependency graph for daq.c:
```

Data Structures

- struct `daq_param_list_node`
- struct `daq_child_task`

Macros

- `#define _SRC_UVFR_DAQ`
- `#define INV_DAQ_P 0xFFFF`

Typedefs

- `typedef struct daq_param_list_node daq_param_list_node`
- `typedef struct daq_child_task daq_child_task`

Functions

- `void daqSubTask (void *args)`
Child Task responsible for sending DAQ messages at a set period.
- `uv_status associateDaqParamWithVar (uint16_t paramID, void *var)`
- `uv_status insertParamToRegister (daq_param_list_node *node, daq_msg *datapoint)`
- `uv_status configureDaqSubTasks ()`
This pre-allocates parameters to one of the daq subtasks.
- `uv_status startDaqSubTasks ()`
Function that starts up all the subtasks.
- `uv_status stopDaqSubTasks ()`
Function that shuts down the subtasks.
- `uv_status initDaqTask (void *args)`
initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks
- `void daqMasterTask (void *args)`
Controls the Daq.

Variables

- `uint64_t constant_zero = 0`
- `volatile uint16_t coolant_temp_adc = 0`
- `volatile uint16_t motor_temp_adc = 0`
- `ADC_HandleTypeDef hadc1`
- `daq_loop_args * curr_daq_settings = NULL`
- `daq_loop_args default_daq_settings`
- `daq_msg default_datapoints []`
- `uv_CAN_msg tmp_daq_msg`

7.65.1 Macro Definition Documentation

7.65.1.1 _SRC_UVFR_DAQ

```
#define _SRC_UVFR_DAQ
```

Definition at line 1 of file [daq.c](#).

7.65.1.2 INV_DAQ_P

```
#define INV_DAQ_P 0xFFFF
```

Definition at line 12 of file [daq.c](#).

7.65.2 Typedef Documentation

7.65.2.1 daq_child_task

```
typedef struct daq_child_task daq_child_task
```

7.65.2.2 daq_param_list_node

```
typedef struct daq_param_list_node daq_param_list_node
```

7.65.3 Function Documentation

7.65.3.1 associateDaqParamWithVar()

```
uv_status associateDaqParamWithVar (
    uint16_t paramID,
    void * var)
```

Definition at line 87 of file [daq.c](#).

References [MAX_LOGGABLE_PARAMS](#), [UV_ERROR](#), [UV_OK](#), and [uvIsPTRValid\(\)](#).

Referenced by [initDaqTask\(\)](#), [initDrivingLoop\(\)](#), and [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.65.3.2 configureDaqSubTasks()

```
uv_status configureDaqSubTasks ()
```

This pre-allocates parameters to one of the daq subtasks.

Definition at line 183 of file [daq.c](#).

References [curr_daq_settings](#), [current_vehicle_settings](#), [uv_vehicle_settings::daq_param_list](#), [insertParamToRegister\(\)](#), [daq_loop_args::total_params_logged](#), [UV_ERROR](#), and [UV_OK](#).

Referenced by [initDaqTask\(\)](#).

Here is the call graph for this function:

7.65.3.3 daqMasterTask()

```
void daqMasterTask (
    void * args)
```

Controls the Daq.

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
/*
//TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
//TickType_t last_time = xTaskGetTickCount();           /**
```

Definition at line 305 of file [daq.c](#).

References [uv_task_info::cmd_data](#), [killSelf\(\)](#), [startDaqSubTasks\(\)](#), [uv_task_info::task_period](#), [UV_KILL_CMD](#), [UV_OK](#), [UV_SUSPEND_CMD](#), and [uvTaskDelay](#).

Referenced by [initDaqTask\(\)](#).

Here is the call graph for this function:

7.65.3.4 daqSubTask()

```
void daqSubTask (
    void * args)
```

Child Task responsible for sending DAQ messages at a set period.

One task per period

Definition at line 441 of file [daq.c](#).

References [daq_child_task::param_list](#), and [daq_child_task::period](#).

Referenced by [startDaqSubTasks\(\)](#).

7.65.3.5 initDaqTask()

```
uv_status initDaqTask (
    void * args)
```

initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks

This is a fairly standard function. Here are the things that it does in order:

Step 1: Get Daq settings.

Step 2: Create and configure DAQ task.

Step 3: Read which parameters we want to read.

Step 4: Generate Subtask Metadata

Step 5: Assign params to subtasks

Definition at line 257 of file [daq.c](#).

References [uv_task_info::active_states](#), [associateDaqParamWithVar\(\)](#), [daq_loop_args::can_channel](#), [configureDaqSubTasks\(\)](#), [COOLANT_TEMP_ADC](#), [coolant_temp_adc](#), [curr_daq_settings](#), [current_vehicle_settings](#), [daq_loop_args::daq_child_priority](#), [uv_vehicle_settings::daq_param_list](#), [uv_vehicle_settings::daq_settings](#), [daqMasterTask\(\)](#), [uv_task_info::deletion_states](#), [uv_CAN_msg::flags](#), [MOTOR_TEMP_ADC](#), [motor_temp_adc](#), [PROGRAMMING](#), [uv_task_info::stack_size](#), [uv_task_info::suspension_states](#), [uv_task_info::task_args](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [uv_task_info::task_period](#), [uv_task_info::task_priority](#), [tmp_daq_msg](#), [UV_DRIVING](#), [UV_ERROR](#), [UV_ERROR_STATE](#), [UV_LAUNCH_CONTROL](#), [UV_OK](#), [UV_READY](#), and [uvCreateTask\(\)](#).

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.65.3.6 insertParamToRegister()

```
uv_status insertParamToRegister (
    daq_param_list_node * node,
    daq_msg * datapoint)
```

Definition at line 124 of file [daq.c](#).

References [daq_msg::can_id](#), [daq_param_list_node::can_id](#), [data_size](#), [daq_param_list_node::next](#), [daq_child_task::next_task](#), [daq_msg::param](#), [daq_param_list_node::param](#), [daq_child_task::param_list](#), [daq_child_task::period](#), [daq_msg::period](#), [daq_param_list_node::size](#), [daq_msg::type](#), and [UV_OK](#).

Referenced by [configureDaqSubTasks\(\)](#).

7.65.3.7 startDaqSubTasks()

```
uv_status startDaqSubTasks ()
```

Function that starts up all the subtasks.

Definition at line 210 of file [daq.c](#).

References [curr_daq_settings](#), [daq_loop_args::daq_child_priority](#), [daqSubTask\(\)](#), [daq_child_task::meta_task_handle](#), [daq_child_task::next_task](#), and [UV_OK](#).

Referenced by [daqMasterTask\(\)](#).

Here is the call graph for this function:

7.65.3.8 stopDaqSubTasks()

```
uv_status stopDaqSubTasks ()
```

Function that shuts down the subtasks.

Definition at line 231 of file [daq.c](#).

References [daq_child_task::meta_task_handle](#), and [UV_ERROR](#).

7.65.4 Variable Documentation

7.65.4.1 constant_zero

```
uint64_t constant_zero = 0
```

Definition at line 16 of file [daq.c](#).

7.65.4.2 coolant_temp_adc

```
volatile uint16_t coolant_temp_adc = 0
```

Definition at line 18 of file [daq.c](#).

Referenced by [initDaqTask\(\)](#).

7.65.4.3 curr_daq_settings

```
daq_loop_args* curr_daq_settings = NULL
```

Definition at line 46 of file [daq.c](#).

Referenced by [configureDaqSubTasks\(\)](#), [initDaqTask\(\)](#), and [startDaqSubTasks\(\)](#).

7.65.4.4 default_daq_settings

```
daq_loop_args default_daq_settings
```

Initial value:

```
= {
    .total_params_logged = 8,
    .throttle_daq_to_preserve_performance = 1,
    .minimum_daq_period = 10,
    .can_channel = CAN_BUS_1,
    .daq_child_priority = 1
}
```

Definition at line 48 of file [daq.c](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.65.4.5 default_datapoints

```
daq_msg default_datapoints[ ]
```

Initial value:

```
= {
    {.can_id = 0x530,
     .param = {APPS1_ADC_VAL, APPS2_ADC_VAL, BPS1_ADC_VAL, BPS2_ADC_VAL},
     .period = 50,
     .type = {UV_UINT16, UV_UINT16, UV_UINT16, UV_UINT16}},

    {.can_id = 0x540,
     .param = {VCU_VEHICLE_STATE, INV_DAQ_P, INV_DAQ_P, INV_DAQ_P},
     .period = 100,
     .type = {UV_UINT16, 0, 0, 0}},

}
```

Definition at line 56 of file [daq.c](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.65.4.6 hadc1

```
ADC_HandleTypeDef hadc1
```

Definition at line 22 of file [daq.c](#).

Referenced by [ADC_IRQHandler\(\)](#), [HAL_ADC_ConvCpltCallback\(\)](#), and [HAL_ADC_LevelOutOfWindowCallback\(\)](#).

7.65.4.7 motor_temp_adc

```
volatile uint16_t motor_temp_adc = 0
```

Definition at line 19 of file [daq.c](#).

Referenced by [initDaqTask\(\)](#).

7.65.4.8 tmp_daq_msg

```
uv_CAN_msg tmp_daq_msg
```

Definition at line 240 of file [daq.c](#).

Referenced by [initDaqTask\(\)](#).

7.66 daq.c

[Go to the documentation of this file.](#)

```
00001 #define _SRC_UVFR_DAQ
00002
00003
00004 #include "uvfr_utils.h"
00005 #include "daq.h"
00006 #include "stm32f4xx_hal_conf.h"
00007 #include "stm32f407xx.h"
00008 #include "stm32f4xx_hal.h"
00009 #include "stm32f4xx_hal_adc.h"
00010 #include "stm32f4xx_hal_rcc.h"
00011
00012 #define INV_DAQ_P 0xFFFF
00013
00014 //##define daq_settings current_vehicle_settings->daq_settings
00015
00016 uint64_t constant_zero = 0;
00017
00018 volatile uint16_t coolant_temp_adc = 0;
00019 volatile uint16_t motor_temp_adc = 0;
00020
00021 //configuring ADCs
00022 ADC_HandleTypeDef hadc1;
00023
00024
00025
00026
00027
00028 typedef struct daq_param_list_node{
00029     struct daq_param_list_node* next;
00030
00031     uint32_t can_id;
00032     uint16_t param[4];
00033
00034     uint8_t size[4];
00035 }daq_param_list_node;
00036
00037 typedef struct daq_child_task{
```

```
00038     struct daq_child_task* next_task;
00039     TaskHandle_t meta_task_handle;
00040     uint32_t period;
00041     daq_param_list_node* param_list;
00042
00043
00044 }daq_child_task;
00045
00046 daq_loop_args* curr_daq_settings = NULL;
00047
00048 daq_loop_args default_daq_settings = {
00049     .total_params_logged = 8,
00050     .throttle_daq_to_preserve_performance = 1,
00051     .minimum_daq_period = 10,
00052     .can_channel = CAN_BUS_1,
00053     .daq_child_priority = 1
00054 };
00055
00056 daq_msg default_datapoints[] ={
00057     {.can_id = 0x530,
00058     .param = {APPS1_ADC_VAL,APPS2_ADC_VAL,BPS1_ADC_VAL,BPS2_ADC_VAL},
00059     .period = 50,
00060     .type = {UV_UINT16,UV_UINT16,UV_UINT16,UV_UINT16}},
00061
00062
00063     {.can_id = 0x540,
00064     .param = {VCU_VEHICLE_STATE,INV_DAQ_P,INV_DAQ_P,INV_DAQ_P},
00065     .period = 100,
00066     .type = {UV_UINT16,0,0,0}},
00067
00068
00069
00070
00071
00072 };
00073
00074 static void* param_ptrs[MAX_LOGGABLE_PARAMS];
00075
00076 static daq_child_task* daq_tlist = NULL;
00077
00078 static daq_msg* datapoints = NULL;
00079
00080 static daq_param_list_node* param_bank = NULL;
00081
00082 void daqSubTask(void* args);
00083
00084 uv_status associateDaqParamWithVar(uint16_t paramID, void* var){
00085     if(uvIsPTRValid(var)!=UV_OK){
00086         return UV_ERROR;
00087     }
00088
00089     if(paramID >= MAX_LOGGABLE_PARAMS){
00090         return UV_ERROR;
00091     }
00092     param_ptrs[paramID] = var;
00093
00094     return UV_OK;
00095 }
00096
00097
00098
00099 }
00100
00101 static inline void insertParamToParamList(daq_param_list_node* node, daq_param_list_node** list){
00102     if(*list == NULL){
00103         *list = node;
00104         return;
00105     }
00106
00107     daq_param_list_node* tmp = *list; //Sets a temporary var as the first list entry
00108
00109     while(tmp->next != NULL){
00110         tmp = tmp->next;
00111     }
00112
00113     //Now this is a situation where tmp->next MUST be NULL
00114     tmp->next = node; // Set it's next as a node
00115
00116
00117 }
00118
00119
00120
00121
00122 uv_status insertParamToRegister(daq_param_list_node* node, daq_msg* datapoint){
00123     //Step 1: find which task this will be assigned to. If no tasks have been created yet, then create
00124     them lol
00125     if(daq_tlist == NULL){
00126         daq_tlist = uvMalloc(sizeof(daq_child_task));
00127
00128         if(daq_tlist == NULL){
00129             //Ooopsies
00130         }
00131     }
00132 }
```

```

00133     daq_tlist->period = datapoint->period;
00134     daq_tlist->next_task = NULL;
00135     daq_tlist->param_list = NULL;
00136 }
00137
00138 daq_child_task* list_tmp = daq_tlist;
00139 node->can_id = datapoint->can_id; //Ensure that the daq_node has the needed params
00140 for(int i = 1; i<4 ; i++){
00141     node->param[i] = datapoint->param[i];
00142     node->size[i] = data_size[datapoint->type[i]];
00143 }
00144
00145 node->next = NULL;
00146
00147 while(1){
00148     //Keep going until we find one with a period that matches
00149     if(list_tmp->period == datapoint->period){
00150         //Insert to list
00151         insertParamToParamList(node, &(list_tmp->param_list));
00152         return UV_OK;
00153     }
00154
00155     if(list_tmp->next_task == NULL){
00156         list_tmp->next_task = uvMalloc(sizeof(daq_child_task));
00157
00158         if(list_tmp->next_task == NULL){
00159             //Another big oopsie
00160         }
00161
00162         list_tmp->next_task->period = datapoint->period;
00163         list_tmp->next_task->next_task = NULL;
00164         list_tmp->next_task->param_list = NULL;
00165         //list_tmp->param_list = NULL;
00166         insertParamToParamList(node, &(list_tmp->next_task->param_list));
00167         return UV_OK;
00168     }
00169 }
00170
00171     list_tmp = list_tmp->next_task;
00172 }
00173
00174
00175 return UV_OK;
00176 }
00177
00178 //daq_param_list_node* param_bank;
00179
00180
00181 uv_status configureDaqSubTasks(){
00182     uint16_t n_logged_params = curr_daq_settings->total_params_logged;
00183     //uint16_t n; // Will use this to track number of params used as we iterate
00184
00185     daq_msg* master_param_list = current_vehicle_settings->daq_param_list;
00186
00187     param_bank = uvMalloc(n_logged_params*sizeof(daq_param_list_node));
00188
00189     if(param_bank == NULL){
00190         uvPanic("outofmem",0);
00191         return UV_ERROR;
00192     }
00193
00194     for(int i = 0; i<n_logged_params ; i++){
00195         //We now go through all of the parameters.
00196         if(insertParamToRegister(&(param_bank[i]),&(master_param_list[i])) != UV_OK){
00197             //This is a non_critical system, so I vote we just ignore it.
00198         }
00199     }
00200 }
00201
00202
00203 return UV_OK;
00204
00205 }
00206
00207
00208 uv_status startDaqSubTasks(){
00209     daq_child_task* tmp = daq_tlist;
00210     BaseType_t retval = 0;
00211     while(tmp != NULL){ //Iterate through linked list of DAQ subtasks
00212         retval = xTaskCreate(daqSubTask,
00213             "DagSub", 256,tmp,
00214             curr_daq_settings->daq_child_priority,
00215             &(tmp->meta_task_handle));
00216         tmp = tmp->next_task;
00217
00218         if(retval != pdPASS){
00219             //Do something to handle the error
00220         }
00221
00222     }
00223
00224 }
00225 return UV_OK;

```

```

00226 }
00227
00231 uv_status stopDaqSubTasks(){
00232     daq_child_task* tmp = daq_tlist;
00233     //BaseType_t retval = 0;
00234     while(tmp!=NULL){
00235         vTaskDelete(tmp->meta_task_handle);
00236     }
00237     return UV_ERROR;
00238 }
00239
00240 uv_CAN_msg tmp_daq_msg;
00241
00257 uv_status initDaqTask(void * args){
00258     //MX_ADC1_Init; //calling initialization of ADC1
00259
00260     curr_daq_settings = current_vehicle_settings->daq_settings;
00261     datapoints = current_vehicle_settings->daq_param_list;
00262     tmp_daq_msg.flags = curr_daq_settings->can_channel;
00263
00264     associateDaqParamWithVar(COOLANT_TEMP_ADC, (void*)&coolant_temp_adc); //HOOKING ADC VARS TO DAQ.
00265     associateDaqParamWithVar(MOTOR_TEMP_ADC, (void*)&motor_temp_adc); //HOOKING ADC VARS TO DAQ.
00266
00267     if(configureDaqSubTasks() != UV_OK){
00268         return UV_ERROR;
00269     }
00270 }
00271
00272 uv_task_info* daq_task = uvCreateTask();
00273
00274 if(daq_task == NULL){
00275     //Oh dear lawd
00276     return UV_ERROR;
00277 }
00278
00279
00280     //DO NOT TOUCH ANY OF THE FIELDS WE HAVENT ALREADY MENTIONED HERE. FOR THE LOVE OF
00281     GOD.
00282     daq_task->task_name = "daq";
00283
00284     daq_task->task_function = daqMasterTask;
00285     daq_task->task_priority = curr_daq_settings->daq_child_priority + 1; //Slightly more important
00286     than the children tasks
00287
00288     daq_task->stack_size = 256;
00289
00290     daq_task->active_states = UV_READY | UV_DRIVING | UV_ERROR_STATE | UV_LAUNCH_CONTROL ;
00291     daq_task->suspension_states = 0x00;
00292     daq_task->deletion_states = PROGRAMMING ;
00293
00294     daq_task->task_period = 20; //measured in ms
00295
00296     daq_task->task_args = NULL; //TODO: Add actual settings dipshit
00297
00298
00299     return UV_OK;
00300 }
00301
00305 void daqMasterTask(void* args){
00306     uv_task_info* params = (uv_task_info*) args; //Evil pointer typecast
00307
00308     vTaskDelay(50); //Give it a moment before spawning in a bunch of daq_tasks
00309
00310     //updateReadings(); //update ADC values
00311
00312     if(startDaqSubTasks() !=UV_OK){
00313         //failed to start the daq subtasks
00314     }
00315
00320     //TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS
00321     ticks
00322     //TickType_t last_time = xTaskGetTickCount();           /**@endcode */
00323     for(;){
00324         if(params->cmd_data == UV_KILL_CMD){
00325             //stopDaqSubTasks();
00326
00327             killSelf(params);
00328         }else if(params->cmd_data == UV_SUSPEND_CMD){
00329             //stopDaqSubTasks();
00330
00331             //suspendSelf(params);
00332         }
00333         uvTaskDelay(params,params->task_period);
00334
00335         //HAL_GPIO_TogglePin(GPIOID,GPIO_PIN_15);

```

```

00335 //      if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)){
00336 //          vTaskDelay(25);
00337 //          while(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)){
00338 //
00339 //      }
00340 //
00341 //
00342 //
00343 //          changeVehicleState(UV_DRIVING);
00344 //
00345 //
00346 //
00347 //      }
00348
00349
00350 }
00351 }
00352
00353
00354
00355
00356 static inline void sendDaqMsg(daq_param_list_node* dmsg){
00357 //    if(param->param > 32) {
00358 //        return;
00359 //    }
00360 //
00361 //    if(param_ptrs[param] == NULL){ //We simply ignore nulls instead of say, crashing
00362 //        return;
00363 //    }
00364 //
00365 //    tmp_daq_msg.msg_id = param->can_id;
00366 //    tmp_daq_msg.data[0] = *((uint8_t*)(param_ptrs[param]));
00367 //    tmp_daq_msg.data[1] = *((uint8_t*)(param_ptrs[param]+1));
00368 //    tmp_daq_msg.data[2] = *((uint8_t*)(param_ptrs[param]+2));
00369 //    tmp_daq_msg.data[3] = *((uint8_t*)(param_ptrs[param]+3));
00370 //    tmp_daq_msg.dlc = param->size;
00371
00372 uv_CAN_msg msg = {
00373     .flags = curr_daq_settings->can_channel,
00374     .data = {0,0,0,0,0,0,0,0},
00375     .msg_id = dmsg->can_id
00376 };
00377
00378 uint16_t param = 0xFFFF;
00379 uint8_t dlc = 0;
00380 uint8_t psize = 0;
00381
00382 for(int i = 0; i<4 ; i++){
00383     param = dmsg->param[i];
00384
00385     if(param >= MAX_LOGGABLE_PARAMS){ //Not a real parameter, sorry!
00386         continue;
00387     }
00388
00389     uint8_t* p_ptr = (uint8_t*)param_ptrs[param];
00390
00391     if(p_ptr == NULL){ //cover edge case where the param ID has not been associated with a
00392         //variable - send 0 in this case
00393         p_ptr = (uint8_t*)&constant_zero;
00394     }
00395
00396     psize = dmsg->size[i];
00397
00398     if((dlc + psize) > 8){
00399         //out of room in the message
00400         break;
00401     }
00402
00403     //increment your data length code :
00404     dlc += psize;
00405
00406     for(int j = 0; j<psize ; j++){
00407         //Add each byte of content to the message
00408         msg.data[(dlc - i) - 1] = *(p_ptr + j); //This puts DAQ messages in BIG endian, for
00409         //optimal human readability
00410     }
00411
00412
00413
00414 }
00415
00416 uvSendCanMSG(&msg);
00417
00418 }
00419
00420 static inline void sendAllParamsFromList(daq_param_list_node* list){
00421     if(list == NULL){
00422         return;
00423

```

```

00426     }
00427
00428     //Linked list iteration moment
00429     while(list != NULL){
00430         sendDaqMsg(list);
00431
00432         list = list->next;
00433     }
00434 }
00435
00436 void daqSubTask(void* args){
00437     dag_child_task* params = (dag_child_task*)args;
00438
00439     TickType_t curr_time = xTaskGetTickCount();
00440     for(;;){
00441         sendAllParamsFromList(params->param_list);
00442         vTaskDelayUntil(&curr_time, pdMS_TO_TICKS(params->period));
00443     }
00444 }
00445
00446 }
```

7.67 Core/Src/dash.c File Reference

```
#include "dash.h"
#include "can.h"
#include "main.h"
Include dependency graph for dash.c:
```

Functions

- void [Update_RPM](#) (int16_t value)
- void [Update_Batt_Temp](#) (uint8_t value)
- void [Update_State_Of_Charge](#) (uint8_t value)

7.67.1 Function Documentation

7.67.1.1 Update_Batt_Temp()

```
void Update_Batt_Temp (
    uint8_t value)
```

Definition at line [29](#) of file [dash.c](#).

References [Dash_Battery_Temperature](#), [Error_Handler\(\)](#), [hcan2](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Here is the call graph for this function:

7.67.1.2 Update_RPM()

```
void Update_RPM (
    int16_t value)
```

Definition at line [9](#) of file [dash.c](#).

References [Dash_RPM](#), [Error_Handler\(\)](#), [hcan2](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.67.1.3 Update_State_Of_Charge()

```
void Update_State_Of_Charge (
    uint8_t value)
```

Definition at line 48 of file [dash.c](#).

References [Dash_State_of_Charge](#), [Error_Handler\(\)](#), [hcan2](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Here is the call graph for this function:

7.68 dash.c

[Go to the documentation of this file.](#)

```
00001 // This where the code to handle displaying data, errors and such to the dash will go
00002
00003 #include "dash.h"
00004 #include "can.h"
00005 #include "main.h"
00006
00007
00008 // If we pass it a negative number then a garbage number will show up
00009 void Update_RPM(int16_t value){
00010
00011     // Need ID
00012     TxHeader.StdId = Dash_RPM; // This is the CAN ID
00013     TxHeader.DLC = 2; // Data Length Code
00014
00015
00016     // Need message
00017     TxData[0] = (value >> 8) & 0xFF;
00018     TxData[1] = value & 0xFF;
00019
00020     if (HAL_CAN_AddTxMessage(&hcan2, &TxHeader, TxData, &TxMailbox) != HAL_OK)
00021     {
00022         /* Transmission request Error */
00023         Error_Handler();
00024     }
00025
00026 }
00027
00028 // If we pass it a negative number then a garbage number will show up
00029 void Update_Batt_Temp(uint8_t value){
00030
00031     // Need ID
00032     TxHeader.StdId = Dash_Battery_Temperature; // This is the CAN ID
00033     TxHeader.DLC = 1; // Data Length Code
00034
00035
00036     // Need message
00037     TxData[0] = value;
00038
00039     if (HAL_CAN_AddTxMessage(&hcan2, &TxHeader, TxData, &TxMailbox) != HAL_OK)
00040     {
00041         /* Transmission request Error */
00042         Error_Handler();
00043     }
00044
00045 }
00046
00047
00048 void Update_State_Of_Charge(uint8_t value){
00049
00050     // Need ID
00051     TxHeader.StdId = Dash_State_of_Charge; // This is the CAN ID
00052     TxHeader.DLC = 1; // Data Length Code
00053
00054
00055     // Need message
00056     TxData[0] = value;
00057
00058     if (HAL_CAN_AddTxMessage(&hcan2, &TxHeader, TxData, &TxMailbox) != HAL_OK)
00059     {
00060         /* Transmission request Error */
00061         Error_Handler();
00062     }
```

```
00063  
00064 }  
00065  
00066  
00067  
00068
```

7.69 Core/Src/dma.c File Reference

This file provides code for the configuration of all the requested memory to memory DMA transfers.

```
#include "dma.h"  
Include dependency graph for dma.c:
```

Functions

- void [MX_DMA_Init](#) (void)

7.69.1 Detailed Description

This file provides code for the configuration of all the requested memory to memory DMA transfers.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [dma.c](#).

7.69.2 Function Documentation

7.69.2.1 MX_DMA_Init()

```
void MX_DMA_Init (  
    void )
```

Enable DMA controller clock

Definition at line 39 of file [dma.c](#).

Referenced by [main\(\)](#).

7.70 dma.c

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ----- */
00022 #include "dma.h"
00023
00024 /* USER CODE BEGIN 0 */
00025
00026 /* USER CODE END 0 */
00027
00028 /*-----*/
00029 /* Configure DMA */
00030 /*-----*/
00031
00032 /* USER CODE BEGIN 1 */
00033
00034 /* USER CODE END 1 */
00035
00039 void MX_DMA_Init(void)
00040 {
00041
00042     /* DMA controller clock enable */
00043     __HAL_RCC_DMA2_CLK_ENABLE();
00044
00045     /* DMA interrupt init */
00046     /* DMA2_Stream0_IRQHandler interrupt configuration */
00047     HAL_NVIC_SetPriority(DMA2_Stream0_IRQHandler, 0, 0);
00048     HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQHandler);
00049     /* DMA2_Stream1_IRQHandler interrupt configuration */
00050     HAL_NVIC_SetPriority(DMA2_Stream1_IRQHandler, 0, 0);
00051     HAL_NVIC_EnableIRQ(DMA2_Stream1_IRQHandler);
00052     /* DMA2_Stream2_IRQHandler interrupt configuration */
00053     HAL_NVIC_SetPriority(DMA2_Stream2_IRQHandler, 0, 0);
00054     HAL_NVIC_EnableIRQ(DMA2_Stream2_IRQHandler);
00055
00056 }
00057
00058 /* USER CODE BEGIN 2 */
00059
00060 /* USER CODE END 2 */
00061

```

7.71 Core/Src/driving_loop.c File Reference

File containing the meat and potatoes driving loop thread, and all supporting functions.

```

#include "main.h"
#include "uvfr_utils.h"
#include "can.h"
#include "motor_controller.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "driving_loop.h"
#include "../FreeRTOS/Source/CMSIS_RTOS/cmsis_os.h"
#include "../FreeRTOS/Source/include/FreeRTOS.h"
#include "../FreeRTOS/Source/include/task.h"
Include dependency graph for driving_loop.c:

```

Macros

- #define INPUT_TIMEOUT_MS 500
- #define THROTTLE_CHANGE_THRESHOLD 5
- #define BRAKE_CHANGE_THRESHOLD 5

- #define TORQUE_DECAY_STEP 2.5f
- #define THROTTLE_ZERO_THRESHOLD 0.01f
- #define ACCELERATION 0
- #define AUTOCROSS 1
- #define ENDURANCE 2

Functions

- float calculateThrottlePercentage (uint16_t apps1, uint16_t apps2)
- float calculateBrakePercentage (uint16_t bps1)
- bool performSafetyChecks (driving_loop_args *dl_params, uint16_t apps1_value, uint16_t apps2_value, uint16_t bps1_value, uint16_t bps2_value, enum DL_internal_state *dl_status)

Performs safety checks on APPS (Throttle) and BPS (Brake) sensors.
- enum uv_status_t initDrivingLoop (void *argument)
- void StartDrivingLoop (void *argument)

Sends the filtered torque value to the motor controller.

Variables

- uint16_t adc1_APPS1
- uint16_t adc1_APPS2
- uint16_t adc1_BPS1
- uint16_t adc1_BPS2
- driving_loop_args default_dl_settings
- driving_loop_args * driving_args = NULL
- bool is_accelerating = false
- float T_PREV = 0
- float T_REQ = 0

7.71.1 Detailed Description

File containing the meat and potatoes driving loop thread, and all supporting functions.

Definition in file [driving_loop.c](#).

7.71.2 Macro Definition Documentation

7.71.2.1 ACCELERATION

```
#define ACCELERATION 0
```

Definition at line 156 of file [driving_loop.c](#).

7.71.2.2 AUTOCROSS

```
#define AUTOCROSS 1
```

Definition at line 157 of file [driving_loop.c](#).

7.71.2.3 BRAKE_CHANGE_THRESHOLD

```
#define BRAKE_CHANGE_THRESHOLD 5
```

Definition at line 45 of file [driving_loop.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.71.2.4 ENDURANCE

```
#define ENDURANCE 2
```

Definition at line 158 of file [driving_loop.c](#).

7.71.2.5 INPUT_TIMEOUT_MS

```
#define INPUT_TIMEOUT_MS 500
```

Definition at line 43 of file [driving_loop.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.71.2.6 THROTTLE_CHANGE_THRESHOLD

```
#define THROTTLE_CHANGE_THRESHOLD 5
```

Definition at line 44 of file [driving_loop.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.71.2.7 THROTTLE_ZERO_THRESHOLD

```
#define THROTTLE_ZERO_THRESHOLD 0.01f
```

Definition at line 53 of file [driving_loop.c](#).

7.71.2.8 TORQUE_DECAY_STEP

```
#define TORQUE_DECAY_STEP 2.5f
```

Definition at line 52 of file [driving_loop.c](#).

7.71.3 Function Documentation

7.71.3.1 calculateBrakePercentage()

```
float calculateBrakePercentage (
    uint16_t bps1)
```

Definition at line 203 of file [driving_loop.c](#).

References [driving_args](#), [driving_loop_args::max_BPS_value](#), and [driving_loop_args::min_BPS_value](#).

Referenced by [performSafetyChecks\(\)](#), and [StartDrivingLoop\(\)](#).

7.71.3.2 calculateThrottlePercentage()

```
float calculateThrottlePercentage (
    uint16_t apps1,
    uint16_t apps2)
```

Definition at line 174 of file [driving_loop.c](#).

References [driving_loop_args::apps1_bottom](#), [driving_loop_args::apps1_top](#), and [driving_args](#).

Referenced by [performSafetyChecks\(\)](#), and [StartDrivingLoop\(\)](#).

7.71.3.3 initDrivingLoop()

```
enum uv_status_t initDrivingLoop (
    void * argument)
```

Definition at line 108 of file [driving_loop.c](#).

References [uv_task_info::active_states](#), [adc1_APPS1](#), [adc1_APPS2](#), [adc1_BPS1](#), [adc1_BPS2](#), [APPSS1_ADC_VAL](#), [APPSS2_ADC_VAL](#), [associateDaqParamWithVar\(\)](#), [BPS1_ADC_VAL](#), [BPS2_ADC_VAL](#), [current_vehicle_settings](#), [uv_task_info::deletion_states](#), [driving_args](#), [uv_vehicle_settings::driving_loop_settings](#), [PROGRAMMING](#), [uv_task_info::stack_size](#), [StartDrivingLoop\(\)](#), [uv_task_info::suspension_states](#), [uv_task_info::task_args](#), [uv_task_info::task_function](#), [uv_task_info::task_name](#), [uv_task_info::task_period](#), [uv_task_info::task_priority](#), [UV_DRIVING](#), [UV_ERROR](#), [UV_ERROR_STATE](#), [UV_INIT](#), [UV_LAUNCH_CONTROL](#), [UV_OK](#), [UV_READY](#), [UV_SUSPENDED](#), and [uvCreateTask\(\)](#).

Referenced by [uvInitStateEngine\(\)](#).

Here is the call graph for this function:

7.71.3.4 performSafetyChecks()

```
bool performSafetyChecks (
    driving_loop_args * dl_params,
    uint16_t apps1_value,
    uint16_t apps2_value,
    uint16_t bps1_value,
    uint16_t bps2_value,
    enum DL_internal_state * dl_status)
```

Performs safety checks on APPS (Throttle) and BPS (Brake) sensors.

This function ensures that:

- Throttle position sensors (APPS1 & APPS2) are within 10% of each other.
- Brake pressure sensors (BPS1 & BPS2) are within 5% of each other.
- Sensors are within their expected min/max ranges.
- Brake and throttle are not pressed at the same time.

If a **fatal error** is detected (e.g., sensor out of range), the function:

- **Stops the motor.**
- **Kills the task execution (`killSelf ()`).**

If a **non-fatal error** occurs (e.g., sensor mismatch exceeding the limit):

- **Stops the motor.**
- **Suspends the task temporarily (`suspendSelf ()`).**

If safety conditions return to normal, the function:

- **Restarts the motor.**

Parameters

<i>dl_params</i>	Pointer to the driving loop parameters.
<i>apps1_value</i>	Raw sensor reading from APPS1.
<i>apps2_value</i>	Raw sensor reading from APPS2.
<i>bps1_value</i>	Raw sensor reading from BPS1.
<i>bps2_value</i>	Raw sensor reading from BPS2.
<i>params</i>	Pointer to the current task information.
<i>dl_status</i>	Pointer to the driving loop internal state.

Return values

<i>true</i>	All safety checks passed.
<i>false</i>	One or more safety checks failed.

Definition at line 432 of file [driving_loop.c](#).

References `driving_loop_args::apps1_abs_max_val`, `driving_loop_args::apps1_bottom`, `driving_loop_args::apps1_top`, `driving_loop_args::apps2_abs_max_val`, `driving_loop_args::apps2_bottom`, `driving_loop_args::apps2_top`, `driving_loop_args::apps_plausibility_check_threshold`, `calculateBrakePercentage()`, `calculateThrottlePercentage()`, `driving_loop_args::max_BPS_value`, and `MC_Shutdown()`.

Referenced by [StartDrivingLoop\(\)](#).

Here is the call graph for this function:

7.71.3.5 StartDrivingLoop()

```
void StartDrivingLoop (
    void * argument)
```

Sends the filtered torque value to the motor controller.

Rachan

Parameters

T_{filtered}	Final torque value after filtering.
-----------------------	-------------------------------------

This line extracts the specific driving loop parameters as specified in the vehicle settings

```
/*
driving_loop_args* dl_params = current_vehicle_settings->driving_loop_settings;

//Timeout values
//static TickType_t last_input_change_time = 0;
//static float last_throttle_percent = 0.0f;
//static float last_brake_percent = 0.0f;

TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = xTaskGetTickCount();
last_input_change_time = last_time;

/**
```

Definition at line 254 of file [driving_loop.c](#).

References [adc1_APPS1](#), [adc1_APPS2](#), [adc1_BPS1](#), [adc1_BPS2](#), [BRAKE_CHANGE_THRESHOLD](#), [calculateBrakePercentage\(\)](#), [calculateThrottlePercentage\(\)](#), [uv_task_info::cmd_data](#), [current_vehicle_settings](#), [uv_vehicle_settings::driving_loop_settings](#), [INPUT_TIMEOUT_MS](#), [is_accelerating](#), [killSelf\(\)](#), [last_driver_input_time](#), [performSafetyChecks\(\)](#), [Plausible](#), [sendTorqueToMotorController\(\)](#), [suspendSelf\(\)](#), [T_PREV](#), [T_REQ](#), [uv_task_info::task_period](#), [THROTTLE_CHANGE_THRESHOLD](#), [UV_DRIVING](#), [UV_KILL_CMD](#), [UV_SUSPEND_CMD](#), and [vehicle_state](#).

Referenced by [initDrivingLoop\(\)](#).

Here is the call graph for this function:

7.71.4 Variable Documentation

7.71.4.1 adc1_APPS1

```
uint16_t adc1_APPS1 [extern]
```

Definition at line 67 of file [main.c](#).

Referenced by [HAL_ADC_LevelOutOfWindowCallback\(\)](#), [initDrivingLoop\(\)](#), [main\(\)](#), and [StartDrivingLoop\(\)](#).

7.71.4.2 adc1_APPS2

```
uint16_t adc1_APPS2 [extern]
```

Definition at line 68 of file [main.c](#).

Referenced by [HAL_ADC_LevelOutOfWindowCallback\(\)](#), [initDrivingLoop\(\)](#), [main\(\)](#), and [StartDrivingLoop\(\)](#).

7.71.4.3 adc1_BPS1

```
uint16_t adc1_BPS1 [extern]
```

Definition at line 69 of file [main.c](#).

Referenced by [initDrivingLoop\(\)](#), and [StartDrivingLoop\(\)](#).

7.71.4.4 adc1_BPS2

```
uint16_t adc1_BPS2 [extern]
```

Definition at line 70 of file [main.c](#).

Referenced by [initDrivingLoop\(\)](#), and [StartDrivingLoop\(\)](#).

7.71.4.5 default_dl_settings

```
driving_loop_args default_dl_settings
```

Definition at line 29 of file [driving_loop.c](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.71.4.6 driving_args

```
driving_loop_args* driving_args = NULL
```

Definition at line 31 of file [driving_loop.c](#).

Referenced by [calculateBrakePercentage\(\)](#), [calculateThrottlePercentage\(\)](#), and [initDrivingLoop\(\)](#).

7.71.4.7 is_accelerating

```
bool is_accelerating = false
```

Definition at line 33 of file [driving_loop.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.71.4.8 T_PREV

```
float T_PREV = 0
```

Definition at line 34 of file [driving_loop.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.71.4.9 T_REQ

```
float T_REQ = 0
```

Definition at line 35 of file [driving_loop.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.72 driving_loop.c

[Go to the documentation of this file.](#)

```
00001
00008 #include "main.h"
00009 #include "uvfr_utils.h"
00010 #include "can.h"
00011 #include "motor_controller.h"
00012 #include "main.h"
00013 #include <stdlib.h> // move somewhere else
00014 #include <stdio.h>
00015 #include <math.h>
00016
00017 #include "driving_loop.h"
00018 #include "../FreeRTOS/Source/CMSIS_RTOs/cmsis_os.h"
00019 #include "../FreeRTOS/Source/include/FreeRTOS.h"
00020 #include "../FreeRTOS/Source/include/task.h"
00021
00022 //External Variables:
00023 extern uint16_t adcl_APPS1; //These are the locations for the sensor inputs for APPS and BPS
00024 extern uint16_t adcl_APPS2;
00025 extern uint16_t adcl_BPS1;
00026 extern uint16_t adcl_BPS2; // Brake
00027
00028
00029 driving_loop_args default_dl_settings; //TODO DECIDE WHAT YOU WANT DEFAULT SETTINGS TO BE
00030
00031 driving_loop_args* driving_args = NULL;
00032
00033 bool is_accelerating = false;
00034 float T_PREV = 0;
00035 float T_REQ = 0;
00036
00037 static bool torque_inhibit_active = false; //track state of torque acceptance
00038 float calculateThrottlePercentage(uint16_t appsl, uint16_t apps2);
00039 float calculateBrakePercentage(uint16_t bpsl);
00040 bool performSafetyChecks(driving_loop_args* dl_params, uint16_t appsl_value, uint16_t apps2_value,
    uint16_t bpsl_value, uint16_t bps2_value, enum DL_internal_state* dl_status);
00041
00042 //timeout thresholds
00043 #define INPUT_TIMEOUT_MS 500
00044 #define THROTTLE_CHANGE_THRESHOLD 5 // percentage
00045 #define BRAKE_CHANGE_THRESHOLD 5 // percentage
00046
00047 //driving loop state tracking
00048 static TickType_t last_input_change_time = 0;
00049 static float last_throttle_percent = 0.0f;
00050 static float last_brake_percent = 0.0f;
00051
00052 #define TORQUE_DECAY_STEP 2.5f
00053 #define THROTTLE_ZERO_THRESHOLD 0.01f // // Below this % throttle, we consider "off"
00054 static bool sent_zero_torque = false;
00055
00056 //define default driving loop settings
00057 driving_loop_args default_dl_settings = {
00058     .absolute_max_acc_pwr = 10,           // Set appropriate default max power in watts
00059     .absolute_max_motor_torque = 230,      // Nm
00060     .absolute_max_accum_current = 200,      // Amps
00061     .max_accum_current_5s = 200,          // Amps for 5s burst
00062
00063     .absolute_max_motor_rpm = 6500,        // Max RPM
00064     .regen_rpm_cutoff = 1000,              // Below this, regen is off
00065
00066     .min_apps_offset = 0,
00067     .max_apps_offset = 0,
00068     .min_apps_value = 0,
00070     .appsl_abs_max_val = 0x10C4,
00071     .appsl_abs_min_val = 0x0200,
00072     .appsl_abs_min_val = 0x0202,
```

```

00073     .apps2_abs_max_val = 0x1029,
00074
00075     .min_BPS_value = 0x0106,
00076     .max_BPS_value = 0x0B7E,
00077     .appsl_top = 0x0993,                                //idk
00078     .appsl_bottom = 0x0570,
00079     .apps2_top = 0x0347,
00080     .apps2_bottom = 0x02B0, //idk
00081
00082
00083     .apps_plausibility_check_threshold = 200,    //idk
00084     .bps_plausibility_check_threshold = 500,    //idk
00085     .bps_implausibility_recovery_threshold = 300,    //idk
00086     .apps_implausibility_recovery_threshold = 100,    //idk
00087
00088     .num_driving_modes = 1,                          //idk
00089     .period = 10,                                     // ms, how often loop runs
00090     .accum_regen_soc_threshold = 90,                  // Above this SOC, regen disabled
00091
00092     .dmodes = {0}                                    // Set default driving modes (all 0 for now)
00093 };
00094
00095
00096 //driving_loop_args* driving_args = NULL;
00097
00098 //bool is_accelerating = false;
00099 //float T_PREV = 0;
00100 //float T_REQ = 0;
00101
00102 //allows torque filter to use getKvalue even if it's defined after
00103 static inline float getKValue(int raceMode);
00104
00105 bool performSafetyChecks(driving_loop_args* dl_params, uint16_t appsl_value,uint16_t apps2_value,
00106     uint16_t bpsl_value, uint16_t bps2_value, enum DL_internal_state* dl_status);
00107
00108 //define diff chennels for adcs
00109 enum uv_status_t initDrivingLoop(void *argument){
00110     associateDaqParamWithVar(APPS1_ADC_VAL, &adc1_APPS1);
00111     associateDaqParamWithVar(APPS2_ADC_VAL, &adc1_APPS2);
00112     associateDaqParamWithVar(BPS1_ADC_VAL, &adc1_BPS1);
00113     associateDaqParamWithVar(BPS2_ADC_VAL, &adc1_BPS2);
00114
00115     //allocate memory for the task
00116     uv_task_info* dl_task = uvCreateTask();
00117
00118     if(dl_task == NULL){
00119         //Oh dear lawd if allocation fails return error
00120         return UV_ERROR;
00121     }
00122
00123     //grab those driving loop settings
00124     driving_args = current_vehicle_settings->driving_loop_settings;
00125
00126     //DO NOT TOUCH ANY OF THE FIELDS WE HAVENT ALREADY MENTIONED HERE. FOR THE LOVE OF GOD.
00127
00128     dl_task->task_name = "Driving_Loop"; //assign names
00129     dl_task->task_function = StartDrivingLoop; //defining the function that the task will run
00130     dl_task->task_priority = osPriorityHigh; // assigns a high priority in FreeRTOS
00131     dl_task->stack_size = 256; // memory allocation for task execution
00132     dl_task->active_states = UV_DRIVING; // Specifies when the task should be active
00133     dl_task->suspension_states = 0x00;
00134
00135     dl_task->deletion_states = UV_INIT|UV_READY | PROGRAMMING | UV_SUSPENDED | UV_LAUNCH_CONTROL |
00136     UV_ERROR_STATE;
00137     dl_task->task_period = 100; //runs every 100ms or 0.1 seconds
00138     dl_task->task_args = NULL;
00139
00140     return UV_OK; //
00141 }
00142 // function to map throttle percent to torque value
00143 inline static float mapThrottleToTorque(float throttle_percent) {
00144     static float T_prev = 0.0f; // Stores the last filtered torque value. for filtering to check
00145     if decelerating
00146         //float throttle_percent = calculateThrottlePercentage(appsl, apps2);
00147         if(throttle_percent == 0.0f){
00148             return 0.0f;
00149         }
00150         double T_MAX = driving_args->absolute_max_motor_torque; // got this from driving_loop.h file
00151         float torque_request_current = (throttle_percent / 100.0f) * T_MAX;
00152         float torque_request = T_prev;
00153         T_prev = torque_request_current;
00154         return torque_request;
00155 }
00156 //race modes
00157 #define ACCELERATION 0
00158 #define AUTOCROSS 1

```

```

00158 #define ENDURANCE 2
00159
00160 //function to get K value aka what mode we are in
00161 inline float getKValue(int raceMode) {
00162     float kVal = 0.3; //Default for if racemode Does not exist
00163     if(raceMode == ACCELERATION) {
00164         return kVal = 0.7;
00165     }else if (raceMode == AUTOCROSS) {
00166         return kVal = 0.4;
00167     }else if (raceMode == ENDURANCE) {
00168         return kVal = 0.2;
00169     }
00170     return kVal;
00171 }
00172
00173 //function to calculate throttle percentage
00174 float calculateThrottlePercentage(uint16_t apps1, uint16_t apps2) {
00175     // Ensure both sensor values are within the valid range
00176     //if (apps1 < driving_args->min_apps1_value || apps1 > driving_args->apps1_abs_max_val ||
00177     apps2 < driving_args->apps2_abs_min_val || apps2 > driving_args->apps2_abs_max_val) return 0.0f;
00178     if(apps1 < driving_args->apps1_bottom){
00179         return 0;
00180     }
00181     if(apps1 > driving_args->apps1_top){
00182         return 100.0f;
00183     }
00184     // Compute throttle percentage using linear interpolation
00185     float throttle_percent = ((float)(apps1 - driving_args->apps1_bottom) /
00186     (driving_args->apps1_top - driving_args->apps1_bottom)) * 100.0f;
00187
00188     // SAFETY CHECK: Verify APPS1 and APPS2 values are within 10% of each other
00189     float apps_diff = fabs((float)apps1 - (float)apps2) / (float)apps1;
00190     if (apps_diff > 0.1f) {
00191         // Sensors are out of sync, return 0% to prevent errors
00192         //printf("WARNING: APPS sensors out of sync! Returning 0% throttle.\n");
00193         //uvPanic("idek",0);
00194         //return 0.0f;
00195     }
00196     //else if (apps_diff <= 0.1f){
00197         // Next function call calcThrottlePercentage(param)
00198         /* OR return something
00199         */
00200     return throttle_percent;
00201 }
00202
00203 //function to calculate brake percentage
00204 float calculateBrakePercentage(uint16_t bps1) {
00205     // Ensure both sensor values are within the valid range
00206     if (bps1 < driving_args->min_BPS_value || bps1 > driving_args->max_BPS_value ) return 0.0f;
00207
00208     // Compute brake percentage using linear interpolation
00209     float brake_percent = ((float)(bps1 - driving_args->min_BPS_value) /
00210     (driving_args->max_BPS_value - driving_args->min_BPS_value)) * 100.0f;
00211     return brake_percent;
00212 }
00213
00214 inline static float applyTorqueFilter(float T_req, float T_prev, bool is_accelearting) {
00215     // Filtering formula: T_filtered = T_prev + (T_req - T_prev) * k
00216     //return T_prev + (T_req - T_prev) * FILTER_K;
00217     // except hehehehhe were gonna create a get_k function based on race modes
00218     float FILTER_K = getKValue(0);
00219     if (is_accelerating) {
00220         FILTER_K = 0.4; // smooth acceleration (adjustable for endurance vs sport mode
00221     }
00222     else{
00223         FILTER_K = 1.0; // INSTANT to
00224     }
00225
00226     // Calculate filtered torque
00227     float T_filtered = T_prev + (T_req - T_prev) * FILTER_K;
00228
00229     // Ensures we do not hold residual torque when stopping
00230     if (T_req == 0) {
00231         T_filtered = 0; // If stopping, is requested torque
00232     }
00233     return T_filtered;
00234 }
00235
00236
00237
00238
00239
00240
00241
00242
00243 }
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253 // Start of Driving Loop
00254 void StartDrivingLoop(void * argument) {
00255     //Initialize driving loop now

```

```

00256
00257 //extracting task arguments, and gets parameters like min/max allowed values for the APPS and BPS
00258 uv_task_info* params = (uv_task_info*) argument;
00259
00260 enum DL_internal_state dl_status = Plausible; // no issues are detected
00261
00265 driving_loop_args* dl_params = current_vehicle_settings->driving_loop_settings;
00266
00267 //Timeout values
00268 //static TickType_t last_input_change_time = 0;
00269 //static float last_throttle_percent = 0.0f;
00270 //static float last_brake_percent = 0.0f;
00271
00272 TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS
00273 ticks
00274     TickType_t last_time = xTaskGetTickCount();
00275     last_input_change_time = last_time;
00276
00278 for(); { // enters infinite loop
00279
00280     if(params->cmd_data == UV_KILL_CMD){ // to perform task control (suspend/kill)
00281
00282         killSelf(params);
00283
00284     }else if(params->cmd_data == UV_SUSPEND_CMD){
00285         suspendSelf(params); // if _UV_SUSPEND_CMD received pause the task
00286     }
00287     vTaskDelayUntil( &last_time, tick_period); //Me and the boys on our way to wait for a set
00288 period every 100ms
00289
00290     HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14); //Blink and LED (for debugging)
00291
00292     //Copy the values over into new local variables, in order to avoid messing up the APPS
00293     uint16_t appsl_value = adc1_APPSL; // reading sensor values
00294     uint16_t apps2_value = adc1_APPS2;
00295
00296     uint16_t bpsl_value = adc1_BPSL;
00297     uint16_t bps2_value = adc1_BPS2;
00298     float T_filtered = 0;
00299
00300     bool safe = performSafetyChecks(dl_params, appsl_value, apps2_value, bpsl_value, bps2_value,
00301     &dl_status);
00302
00303     if(!safe) {
00304         // if safety check fails, handle error (stop?)
00305         //continue;
00306         //MC_Shutdown();
00307         T_filtered = 0;
00308         sendTorqueToMotorController(T_filtered);
00309         continue;
00310     }
00311
00312     if(dl_status == Plausible){
00313
00314         //implement motor control logic here
00315
00316         //Compute throttle %
00317         float throttle_percent = calculateThrottlePercentage(appsl_value, apps2_value);
00318         float brake_percent = calculateBrakePercentage(bpsl_value);
00319
00320         // -----
00321         // How much did the throttle or brake change since last time?
00322         float throttle_delta = fabs(throttle_percent - last_throttle_percent);
00323         float brake_delta = fabs(brake_percent - last_brake_percent);
00324
00325         //TickType_t now = xTaskGetTickCount();
00326         TickType_t timeout_ticks = pdMS_TO_TICKS(INPUT_TIMEOUT_MS);
00327
00328         // If either % changed significantly, update the "last time the driver did something"
00329         // Only update the time if there's meaningful input change
00330         if (throttle_delta > THROTTLE_CHANGE_THRESHOLD || brake_delta > BRAKE_CHANGE_THRESHOLD) {
00331             last_driver_input_time = xTaskGetTickCount();
00332             last_throttle_percent = throttle_percent;
00333             last_brake_percent = brake_percent;
00334         }
00335
00336         // Check for input timeout
00337         // If it's been too long since the last meaningful input
00338         //bool input_timeout = (last_driver_input_time - last_input_change_time) > timeout_ticks;
00339
00340         bool input_timeout = (xTaskGetTickCount() - last_input_change_time) > timeout_ticks;
00341
00342         //-----
00343
00344         //APPS and Brake Pedal Plausibility Check

```

```

00344         // 1. Trigger torque inhibit if APPS > 25% and Brake is pressed
00345         if (throttle_percent > 25.0f && brake_percent > 5.0f) {
00346             torque_inhibit_active = true;
00347         }
00348         // 2. Do NOT reset torque inhibit until BOTH are released to < thresholds
00349         if (torque_inhibit_active && throttle_percent < 5.0f && brake_percent < 5.0f) {
00350             torque_inhibit_active = false;
00351         }
00352         // 3. If inhibit is active, override torque request
00353         if (torque_inhibit_active) {
00354             T_filtered = 0.0f;
00355         }
00356         // Enforce torque cutoff if:
00357         // 4. No driver input detected for too long (input_timeout)
00358         // 5. Torque inhibit is active due to APPS >25% and Brake >5% (per FSAE Rule T.4.2.5)
00359         //   → Torque must remain zero until APPS <5% and Brake <5% to clear inhibit
00360         //if (input_timeout || torque_inhibit_active) { //this might cause the inverter to break
00361             bc it'll cut torque after 500ms of no pedal?
00362             if (torque_inhibit_active){
00363                 T_filtered = 0.0f;
00364                 sent_zero_torque = true;
00365             }else{ //normal driving logic
00366                 // 2. Map to torque request
00367                 T_REQ = mapThrottleToTorque(throttle_percent);
00368
00369                 // 3. Determine acceleration status
00370                 is_accelerating = (T_REQ >= T_PREV);
00371
00372                 // 4. Apply filtering to smooth torque
00373                 T_filtered = applyTorqueFilter(T_REQ, T_PREV, is_accelerating);
00374                 // 6. Update previous torque
00375                 //todo: fix T_filter scaling
00376                 //half the requested torque
00377                 T_filtered = T_filtered/2;
00378             }
00379
00380             // 5. Send torque to motor controller via motor_controller.c
00381             if(vehicle_state == UV_DRIVING){
00382                 sendTorqueToMotorController(T_filtered);
00383             }else{
00384                 sendTorqueToMotorController(0.0);
00385             }
00386
00387             T_PREV = T_filtered;
00388
00389         //}
00390     }
00391 }
00392
00393
00394
00395
00396
00397 }
00398
00399
00400
00432 bool performSafetyChecks(driving_loop_args* dl_params,uint16_t apps1_value,uint16_t
    apps2_value,uint16_t bps1_value,uint16_t bps2_value,enum DL_internal_state* dl_status){
00433
00434     //sensor scaling
00435     float apps1_ratio = 0;
00436     float apps2_ratio = 0;
00437
00438     // Normalize APPS1
00439     if (apps1_value < dl_params->apps1_bottom) {
00440         apps1_ratio = 0;
00441     }
00442     else if (apps1_value > dl_params->apps1_top) {
00443         apps1_ratio = 1.0f;
00444     }
00445     else{
00446         apps1_ratio = ((float)(apps1_value - dl_params->apps1_bottom)) / (dl_params->apps1_top -
00447             dl_params->apps1_bottom);
00448     }
00449
00450     // Normalize APPS2
00451     if (apps2_value < dl_params->apps2_bottom) {
00452         apps2_ratio = 0;
00453     }
00454     else if (apps2_value > dl_params->apps2_top) {
00455         apps2_ratio = 1.0f;
00456     }
00457     else {
00458         apps2_ratio = ((float)(apps2_value - dl_params->apps2_bottom)) / (dl_params->apps2_top -
00459             dl_params->apps2_bottom);
00460     }

```

```

00458     }
00459     // Calculate the absolute difference between APPS1 and APPS2 as a percentage
00460     // (FSAE Rule T.4.2.4: Implausibility is deviation >10% for >100ms)
00461     float apps_diff_percent = fabsf(apps1_ratio - apps2_ratio) * 100.0f;
00462
00463
00464     //FATAL RANGE ERRORS
00465     //APPSS1 Range
00466     if (apps1_value < dl_params->apps1_abs_min_val || apps1_value > dl_params->apps1_abs_max_val) {
00467         MC_Shutdown();
00468         return false;
00469     }
00470     //APPSS2 Range
00471     if (apps2_value < dl_params->apps2_abs_min_val || apps2_value > dl_params->apps2_abs_max_val) {
00472         MC_Shutdown();
00473         return false;
00474     }
00475     //BPS1 Range
00476     if (bps1_value < dl_params->min_BPS_value || bps1_value > dl_params->max_BPS_value) {
00477         MC_Shutdown();
00478         return false;
00479     }
00480     //BPS2 Range
00481     if (bps2_value < dl_params->min_BPS_value || bps2_value > dl_params->max_BPS_value) {
00482         MC_Shutdown();
00483         return false;
00484     }
00485
00486     //APPS MISMATCH CHECK. If greater then 10% then shutdown.
00487     //if (apps_diff_percent > 25.0f) {
00488     if (apps_diff_percent > dl_params->apps_plausibility_check_threshold){
00489
00490         /*dl_status = Implausible;
00491         MC_Shutdown();
00492         return false;
00493     */
00494
00495     //THROTTLE + BRAKE Percent
00496     float throttle_percent = calculateThrottlePercentage(apps1_value, apps2_value);
00497     float brake_percent = calculateBrakePercentage(bps1_value);
00498
00499     //RECOVERY CHECK. If implausible on return to original state after both brake and pedal released
00500     //    if (*dl_status == Implausible &&
00501     //        throttle_percent < dl_params->apps_implausibility_recovery_threshold &&
00502     //        brake_percent < dl_params->bps_implausibility_recovery_threshold)
00503     //    {
00504     //        *dl_status = Plausible;
00505     //    }
00506
00507     if (torque_inhibit_active && throttle_percent < dl_params->apps_implausibility_recovery_threshold
00508     && brake_percent < dl_params->bps_implausibility_recovery_threshold){
00509         torque_inhibit_active = false;
00510     }
00511
00512     return true; // All good
00513 }
00514

```

7.73 Core/Src/gpio.c File Reference

This file provides code for the configuration of all used GPIO pins.

```
#include "gpio.h"
Include dependency graph for gpio.c:
```

Functions

- void [MX_GPIO_Init](#) (void)

7.73.1 Detailed Description

This file provides code for the configuration of all used GPIO pins.

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [gpio.c](#).

7.73.2 Function Documentation

7.73.2.1 MX_GPIO_Init()

```
void MX_GPIO_Init (
    void )
```

Configure pins as Analog Input Output EVENT_OUT EXTI

Definition at line [42](#) of file [gpio.c](#).

References [DEBUG_LED_Pin](#).

Referenced by [main\(\)](#).

7.74 gpio.c

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes -----*/
00022 #include "gpio.h"
00023
00024 /* USER CODE BEGIN 0 */
00025
00026 /* USER CODE END 0 */
00027
00028 /*-----*/
00029 /* Configure GPIO */
00030 /*-----*/
00031 /* USER CODE BEGIN 1 */
00032
00033 /* USER CODE END 1 */
00034
00042 void MX_GPIO_Init(void)
00043 {
00044
00045     GPIO_InitTypeDef GPIO_InitStruct = {0};
00046
00047     /* GPIO Ports Clock Enable */
00048     __HAL_RCC_GPIOH_CLK_ENABLE();
00049     __HAL_RCC_GPIOC_CLK_ENABLE();
00050     __HAL_RCC_GPIOA_CLK_ENABLE();
00051     __HAL_RCC_GPIOB_CLK_ENABLE();
00052     __HAL_RCC_GPIOD_CLK_ENABLE();
00053
```

```

00054 /*Configure GPIO pin Output Level */
00055 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10|DEBUG_LED_Pin, GPIO_PIN_RESET);
00056
00057 /*Configure GPIO pins : PD8 PD9 PD10 PD11
00058             PD0 PD1 PD2 PD3
00059             PD4 PD5 PD6 PD7 */
00060 GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
00061             |GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
00062             |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
00063 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00064 GPIO_InitStruct.Pull = GPIO_PULLDOWN;
00065 HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
00066
00067 /*Configure GPIO pins : PD12 PD13 PD14 PD15 */
00068 GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
00069 GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
00070 GPIO_InitStruct.Pull = GPIO_PULLDOWN;
00071 HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
00072
00073 /*Configure GPIO pins : PC10 PCPin */
00074 GPIO_InitStruct.Pin = GPIO_PIN_10|DEBUG_LED_Pin;
00075 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00076 GPIO_InitStruct.Pull = GPIO_NOPULL;
00077 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00078 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
00079
00080 /* EXTI interrupt init*/
00081 HAL_NVIC_SetPriority(EXTI15_10_IRQn, 7, 0);
00082 HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
00083
00084 }
00085
00086 /* USER CODE BEGIN 2 */
00087
00088 /* USER CODE END 2 */

```

7.75 Core/Src/imd.c File Reference

```

#include "imd.h"
#include "can.h"
#include "main.h"
#include "constants.h"
#include "uvfr_utils.h"
#include "pdu.h"

```

Include dependency graph for imd.c:

Functions

- void **IMD_Parse_Message** (int DLC, uint8_t Data[])
- void **IMD_Request_Status** (uint8_t Status)
- void **IMD_Check_Status_Bits** (uint8_t Data)
- void **IMD_Check_Error_Flags** (uint8_t Data[])
- void **IMD_Check_Isolation_State** (uint8_t Data[])
- void **IMD_Check_Isolation_Resistances** (uint8_t Data[])
- void **IMD_Check_Isolation_Capacitances** (uint8_t Data[])
- void **IMD_Check_Voltages_Vp_and_Vn** (uint8_t Data[])
- void **IMD_Check_Battery_Voltage** (uint8_t Data[])
- void **IMD_Check_Temperature** (uint8_t Data[])
- void **IMD_Check_Safety_Touch_Energy** (uint8_t Data[])
- void **IMD_Check_Safety_Touch_Current** (uint8_t Data[])
- void **IMD_Check_Max_Battery_Working_Voltage** (uint8_t Data[])
- void **IMD_Check_Part_Name** (uint8_t Data[])
- void **IMD_Check_Version** (uint8_t Data[])
- void **IMD_Check_Serial_Number** (uint8_t Data[])
- void **IMD_Check_Uptime** (uint8_t Data[])
- void **IMD_Startup** ()
- void **initIMD** (void *args)

Variables

- `uint8_t IMD_status_bits = 0`
- `uint8_t IMD_High_Uncertainty = 0`
- `uint32_t IMD_Read_Part_Name [4]`
- `const uint32_t IMD_Expected_Part_Name [4]`
- `uint8_t IMD_Part_Name_0_Set = 0`
- `uint8_t IMD_Part_Name_1_Set = 0`
- `uint8_t IMD_Part_Name_2_Set = 0`
- `uint8_t IMD_Part_Name_3_Set = 0`
- `uint8_t IMD_Part_Name_Set = 0`
- `uint32_t IMD_Read_Version [3]`
- `const uint32_t IMD_Expected_Version [3]`
- `uint8_t IMD_Version_0_Set = 0`
- `uint8_t IMD_Version_1_Set = 0`
- `uint8_t IMD_Version_2_Set = 0`
- `uint8_t IMD_Version_Set = 0`
- `uint32_t IMD_Read_Serial_Number [4]`
- `const uint32_t IMD_Expected_Serial_Number [4]`
- `uint8_t IMD_Serial_Number_0_Set = 0`
- `uint8_t IMD_Serial_Number_1_Set = 0`
- `uint8_t IMD_Serial_Number_2_Set = 0`
- `uint8_t IMD_Serial_Number_3_Set = 0`
- `uint8_t IMD_Serial_Number_Set = 0`
- `int32_t IMD_Temperature`
- `uint8_t IMD_error_flags_requested = 0`
- `uv_imd_settings default_imd_settings`

7.75.1 Function Documentation

7.75.1.1 IMD_Check_Battery_Voltage()

```
void IMD_Check_Battery_Voltage (
    uint8_t Data[ ])
```

Definition at line 356 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.2 IMD_Check_Error_Flags()

```
void IMD_Check_Error_Flags (
    uint8_t Data[ ])
```

Definition at line 262 of file [imd.c](#).

References [Err_CH](#), [Err_clock](#), [Err_temp](#), [Err_Vexi](#), [Err_Vpwr](#), [Err_Vx1](#), [Err_Vx2](#), [Err_VxR](#), and [Err_Watchdog](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.3 IMD_Check_Isolation_Capacitances()

```
void IMD_Check_Isolation_Capacitances (
    uint8_t Data[ ])
```

Definition at line 342 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.4 IMD_Check_Isolation_Resistances()

```
void IMD_Check_Isolation_Resistances (
    uint8_t Data[ ])
```

Definition at line 317 of file [imd.c](#).

References [IMD_High_Uncertainty](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.5 IMD_Check_Isolation_State()

```
void IMD_Check_Isolation_State (
    uint8_t Data[ ])
```

Definition at line 301 of file [imd.c](#).

References [IMD_High_Uncertainty](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.6 IMD_Check_Max_Battery_Working_Voltage()

```
void IMD_Check_Max_Battery_Working_Voltage (
    uint8_t Data[ ])
```

Definition at line 393 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.7 IMD_Check_Part_Name()

```
void IMD_Check_Part_Name (
    uint8_t Data[ ])
```

Definition at line 406 of file [imd.c](#).

References [IMD_Expected_Part_Name](#), [IMD_Part_Name_0_Set](#), [IMD_Part_Name_1_Set](#), [IMD_Part_Name_2_Set](#), [IMD_Part_Name_3_Set](#), [IMD_Part_Name_Set](#), [IMD_Read_Part_Name](#), [Part_name_0](#), [Part_name_1](#), [Part_name_2](#), and [Part_name_3](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.8 IMD_Check_Safety_Touch_Current()

```
void IMD_Check_Safety_Touch_Current (
    uint8_t Data[])
```

Definition at line 381 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.9 IMD_Check_Safety_Touch_Energy()

```
void IMD_Check_Safety_Touch_Energy (
    uint8_t Data[])
```

Definition at line 374 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.10 IMD_Check_Serial_Number()

```
void IMD_Check_Serial_Number (
    uint8_t Data[])
```

Definition at line 488 of file [imd.c](#).

References [IMD_Expected_Serial_Number](#), [IMD_Read_Serial_Number](#), [IMD_Serial_Number_0_Set](#), [IMD_Serial_Number_1_Set](#), [IMD_Serial_Number_2_Set](#), [IMD_Serial_Number_3_Set](#), [IMD_Serial_Number_Set](#), [Serial_number_0](#), [Serial_number_1](#), [Serial_number_2](#), and [Serial_number_3](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.11 IMD_Check_Status_Bits()

```
void IMD_Check_Status_Bits (
    uint8_t Data)
```

Definition at line 218 of file [imd.c](#).

References [Error_flags](#), [Hardware_Error](#), [High_Battery_Voltage](#), [High_Uncertainty](#), [IMD_error_flags_requested](#), [IMD_High_Uncertainty](#), [IMD_Request_Status\(\)](#), [Isolation_status_bit0](#), [Isolation_status_bit1](#), and [Low_Battery_Voltage](#).

Referenced by [IMD_Parse_Message\(\)](#).

Here is the call graph for this function:

7.75.1.12 IMD_Check_Temperature()

```
void IMD_Check_Temperature (
    uint8_t Data[])
```

Definition at line 363 of file [imd.c](#).

References [IMD_Temperature](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.13 IMD_Check_Uptime()

```
void IMD_Check_Uptime (
    uint8_t Data[ ])
```

Definition at line 529 of file [imd.c](#).

7.75.1.14 IMD_Check_Version()

```
void IMD_Check_Version (
    uint8_t Data[ ])
```

Definition at line 448 of file [imd.c](#).

References [IMD_Expected_Version](#), [IMD_Read_Version](#), [IMD_Version_0_Set](#), [IMD_Version_1_Set](#), [IMD_Version_2_Set](#), [IMD_Version_Set](#), [Version_0](#), [Version_1](#), and [Version_2](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.15 IMD_Check_Voltages_Vp_and_Vn()

```
void IMD_Check_Voltages_Vp_and_Vn (
    uint8_t Data[ ])
```

Definition at line 349 of file [imd.c](#).

Referenced by [IMD_Parse_Message\(\)](#).

7.75.1.16 IMD_Parse_Message()

```
void IMD_Parse_Message (
    int DLC,
    uint8_t Data[ ])
```

Definition at line 73 of file [imd.c](#).

References [battery_voltage](#), [Error_flags](#), [Error_Handler\(\)](#), [IMD_Check_Battery_Voltage\(\)](#), [IMD_Check_Error_Flags\(\)](#), [IMD_Check_Isolation_Capacitances\(\)](#), [IMD_Check_Isolation_Resistances\(\)](#), [IMD_Check_Isolation_State\(\)](#), [IMD_Check_Max_Battery_Working_Voltage\(\)](#), [IMD_Check_Part_Name\(\)](#), [IMD_Check_Safety_Touch_Current\(\)](#), [IMD_Check_Safety_Touch_Energy\(\)](#), [IMD_Check_Serial_Number\(\)](#), [IMD_Check_Status_Bits\(\)](#), [IMD_Check_Temperature\(\)](#), [IMD_Check_Version\(\)](#), [IMD_Check_Voltages_Vp_and_Vn\(\)](#), [isolation_capacitances](#), [isolation_resistances](#), [isolation_state](#), [Max_battery_working_voltage](#), [Part_name_0](#), [Part_name_1](#), [Part_name_2](#), [Part_name_3](#), [safety_touch_current](#), [safety_touch_energy](#), [Serial_number_0](#), [Serial_number_1](#), [Serial_number_2](#), [Serial_number_3](#), [Temperature](#), [Uptime_counter](#), [Vb_hi_res](#), [Version_0](#), [Version_1](#), [Version_2](#), [Vexc_hi_res](#), [Vn_hi_res](#), [voltages_Vp_and_Vn](#), [Vp_hi_res](#), and [Vpwr_hi_res](#).

Here is the call graph for this function:

7.75.1.17 IMD_Request_Status()

```
void IMD_Request_Status (
    uint8_t Status)
```

Definition at line 185 of file [imd.c](#).

References [Error_Handler\(\)](#), [hcan2](#), [IMD_CAN_ID_Tx](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

Referenced by [IMD_Check_Status_Bits\(\)](#), and [IMD_Startup\(\)](#).

Here is the call graph for this function:

7.75.1.18 IMD_Startup()

```
void IMD_Startup ()
```

Definition at line 533 of file [imd.c](#).

References [IMD_Request_Status\(\)](#), [isolation_state](#), [Max_battery_working_voltage](#), [Part_name_0](#), [Part_name_1](#), [Part_name_2](#), [Part_name_3](#), [Serial_number_0](#), [Serial_number_1](#), [Serial_number_2](#), [Serial_number_3](#), [Version_0](#), [Version_1](#), and [Version_2](#).

Here is the call graph for this function:

7.75.1.19 initIMD()

```
void initIMD (
    void * args)
```

Definition at line 559 of file [imd.c](#).

References [IMD](#), [uv_init_task_args::init_info_queue](#), [uv_init_task_args::meta_task_handle](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

7.75.2 Variable Documentation

7.75.2.1 default_imd_settings

```
uv_imd_settings default_imd_settings
```

Initial value:

```
= {
    .min_isolation_resistances = 10000,
    .expected_isolation_capacitances = 100,
    .max_imd_temperature = 60
}
```

Definition at line 65 of file [imd.c](#).

Referenced by [uvResetFlashToDefault\(\)](#).

7.75.2.2 IMD_error_flags_requested

```
uint8_t IMD_error_flags_requested = 0
```

Definition at line [62](#) of file [imd.c](#).

Referenced by [IMD_Check_Status_Bits\(\)](#).

7.75.2.3 IMD_Expected_Part_Name

```
const uint32_t IMD_Expected_Part_Name[4]
```

Definition at line [26](#) of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.4 IMD_Expected_Serial_Number

```
const uint32_t IMD_Expected_Serial_Number[4]
```

Initial value:

```
= {0xB8DD9AF9,  
     0x6094F48B,  
     0xF1C3794,  
     0xFC9A95B}
```

Definition at line [46](#) of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.5 IMD_Expected_Version

```
const uint32_t IMD_Expected_Version[3]
```

Definition at line [36](#) of file [imd.c](#).

Referenced by [IMD_Check_Version\(\)](#).

7.75.2.6 IMD_High_Uncertainty

```
uint8_t IMD_High_Uncertainty = 0
```

Definition at line [20](#) of file [imd.c](#).

Referenced by [IMD_Check_Isolation_Resistances\(\)](#), [IMD_Check_Isolation_State\(\)](#), and [IMD_Check_Status_Bits\(\)](#).

7.75.2.7 IMD_Part_Name_0_Set

```
uint8_t IMD_Part_Name_0_Set = 0
```

Definition at line [28](#) of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.8 IMD_Part_Name_1_Set

```
uint8_t IMD_Part_Name_1_Set = 0
```

Definition at line 29 of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.9 IMD_Part_Name_2_Set

```
uint8_t IMD_Part_Name_2_Set = 0
```

Definition at line 30 of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.10 IMD_Part_Name_3_Set

```
uint8_t IMD_Part_Name_3_Set = 0
```

Definition at line 31 of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.11 IMD_Part_Name_Set

```
uint8_t IMD_Part_Name_Set = 0
```

Definition at line 32 of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.12 IMD_Read_Part_Name

```
uint32_t IMD_Read_Part_Name[4]
```

Definition at line 25 of file [imd.c](#).

Referenced by [IMD_Check_Part_Name\(\)](#).

7.75.2.13 IMD_Read_Serial_Number

```
uint32_t IMD_Read_Serial_Number[4]
```

Definition at line 45 of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.14 IMD_Read_Version

```
uint32_t IMD_Read_Version[3]
```

Definition at line 35 of file [imd.c](#).

Referenced by [IMD_Check_Version\(\)](#).

7.75.2.15 IMD_Serial_Number_0_Set

```
uint8_t IMD_Serial_Number_0_Set = 0
```

Definition at line 50 of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.16 IMD_Serial_Number_1_Set

```
uint8_t IMD_Serial_Number_1_Set = 0
```

Definition at line 51 of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.17 IMD_Serial_Number_2_Set

```
uint8_t IMD_Serial_Number_2_Set = 0
```

Definition at line 52 of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.18 IMD_Serial_Number_3_Set

```
uint8_t IMD_Serial_Number_3_Set = 0
```

Definition at line 53 of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.19 IMD_Serial_Number_Set

```
uint8_t IMD_Serial_Number_Set = 0
```

Definition at line 54 of file [imd.c](#).

Referenced by [IMD_Check_Serial_Number\(\)](#).

7.75.2.20 IMD_status_bits

```
uint8_t IMD_status_bits = 0
```

Definition at line 19 of file [imd.c](#).

7.75.2.21 IMD_Temperature

```
int32_t IMD_Temperature
```

Definition at line 57 of file [imd.c](#).

Referenced by [IMD_Check_Temperature\(\)](#).

7.75.2.22 IMD_Version_0_Set

```
uint8_t IMD_Version_0_Set = 0
```

Definition at line 38 of file [imd.c](#).

Referenced by [IMD_Check_Version\(\)](#).

7.75.2.23 IMD_Version_1_Set

```
uint8_t IMD_Version_1_Set = 0
```

Definition at line 39 of file [imd.c](#).

Referenced by [IMD_Check_Version\(\)](#).

7.75.2.24 IMD_Version_2_Set

```
uint8_t IMD_Version_2_Set = 0
```

Definition at line 40 of file [imd.c](#).

Referenced by [IMD_Check_Version\(\)](#).

7.75.2.25 IMD_Version_Set

```
uint8_t IMD_Version_Set = 0
```

Definition at line 41 of file [imd.c](#).

Referenced by [IMD_Check_Version\(\)](#).

7.76 imd.c

[Go to the documentation of this file.](#)

```

00001 // This where the code to handle IMD errors and such will go
00002
00003
00004 #include "imd.h"
00005
00006 // We need to include can.h because we will send CAN messages through the functions in that file
00007 // When a CAN message comes it will throw an interrupt can.c deals with the incoming message
00008 // the function in can.c gets the ID and sends the data to the functions here
00009 #include "can.h"
00010 #include "main.h"
00011 #include "constants.h"
00012 #include "uvfr_utils.h"
00013
00014 // We need to include pdu.h for the shutdown circuit
00015 #include "pdu.h"
00016
00017
00018
00019 uint8_t IMD_Status_Bits = 0;
00020 uint8_t IMD_High_Uncertainty = 0;
00021
00022
00023
00024 // Declarations for the IMD part name to be checked on startup
00025 uint32_t IMD_Read_Part_Name[4];
00026 const uint32_t IMD_Expected_Part_Name[4];
00027
00028 uint8_t IMD_Part_Name_0_Set = 0;
00029 uint8_t IMD_Part_Name_1_Set = 0;
00030 uint8_t IMD_Part_Name_2_Set = 0;
00031 uint8_t IMD_Part_Name_3_Set = 0;
00032 uint8_t IMD_Part_Name_Set = 0;
00033
00034
00035 uint32_t IMD_Read_Version[3];
00036 const uint32_t IMD_Expected_Version[3];
00037
00038 uint8_t IMD_Version_0_Set = 0;
00039 uint8_t IMD_Version_1_Set = 0;
00040 uint8_t IMD_Version_2_Set = 0;
00041 uint8_t IMD_Version_Set = 0;
00042
00043
00044 // Declarations for the IMD serial number to be checked on startup
00045 uint32_t IMD_Read_Serial_Number[4];
00046 const uint32_t IMD_Expected_Serial_Number[4] = {0xB8DD9AF9,
00047                                         0x6094F48B,
00048                                         0x1F1C3794,
00049                                         0xFPCF9A95B};
00050 uint8_t IMD_Serial_Number_0_Set = 0;
00051 uint8_t IMD_Serial_Number_1_Set = 0;
00052 uint8_t IMD_Serial_Number_2_Set = 0;
00053 uint8_t IMD_Serial_Number_3_Set = 0;
00054 uint8_t IMD_Serial_Number_Set = 0;
00055
00056
00057 int32_t IMD_Temperature;
00058
00059
00060 // If there is a hardware error, that one bit will be a 1 in the status bits -> read error flags
00061 // error flags will return the status bits which will have a 1 in HE bit -> infinite loop
00062 uint8_t IMD_error_flags_requested = 0;
00063
00064
00065 uv_imd_settings default_imd_settings = {
00066     .min_isolation_resistances = 10000,
00067     .expected_isolation_capacitances = 100,
00068     .max_imd_temperature = 60
00069 };
00070
00071
00072 // Need a function to parse the CAN message data received from the IMD
00073 void IMD_Parse_Message(int DLC, uint8_t Data[]){
00074     // The first step is to look at the first byte to figure out what we're looking at
00075
00076     switch (Data[0]){
00077         // important checks
00078         case isolation_state:
00079             IMD_Check_Status_Bits(Data[1]);
00080             IMD_Check_Isolation_State(Data);
00081             break;
00082

```

```
00083     case isolation_resistances:
00084         IMD_Check_Status_Bits(Data[1]);
00085         IMD_Check_Isolation_Resistances(Data);
00086     break;
00087
00088     case isolation_capacitances:
00089         IMD_Check_Status_Bits(Data[1]);
00090         IMD_Check_Isolation_Capacitances(Data);
00091     break;
00092
00093     case voltages_Vp_and_Vn:
00094         IMD_Check_Status_Bits(Data[1]);
00095         IMD_Check_Voltages_Vp_and_Vn(Data);
00096     break;
00097
00098     case battery_voltage:
00099         IMD_Check_Status_Bits(Data[1]);
00100         IMD_Check_Battery_Voltage(Data);
00101     break;
00102
00103     case Error_flags:
00104         IMD_Check_Status_Bits(Data[1]);
00105         IMD_Check_Error_Flags(Data);
00106     break;
00107
00108     case safety_touch_energy:
00109         IMD_Check_Status_Bits(Data[1]);
00110         IMD_Check_Safety_Touch_Energy(Data);
00111     break;
00112
00113     case safety_touch_current:
00114         IMD_Check_Status_Bits(Data[1]);
00115         IMD_Check_Safety_Touch_Current(Data);
00116     break;
00117
00118     // high resolution measurements
00119     case Vn_hi_res:
00120         // do something
00121     break;
00122
00123     case Vp_hi_res:
00124         // do something
00125     break;
00126
00127     case Vexc_hi_res:
00128         // do something
00129     break;
00130
00131     case Vb_hi_res:
00132         // do something
00133     break;
00134
00135     case Vpwr_hi_res:
00136         // do something
00137     break;
00138
00139     case Temperature:
00140         IMD_Check_Temperature(Data);
00141     break;
00142
00143     case Max_battery_working_voltage:
00144         IMD_Check_Max_Battery_Working_Voltage(Data);
00145     break;
00146
00147     // ugly syntax below
00148     case Part_name_0:
00149     case Part_name_1:
00150     case Part_name_2:
00151     case Part_name_3:
00152         IMD_Check_Part_Name(Data);
00153     break;
00154
00155     case Version_0:
00156     case Version_1:
00157     case Version_2:
00158         IMD_Check_Version(Data);
00159     break;
00160
00161     case Serial_number_0:
00162     case Serial_number_1:
00163     case Serial_number_2:
00164     case Serial_number_3:
00165         IMD_Check_Serial_Number(Data);
00166     break;
00167
00168     case Uptime_counter:
00169         // call check uptime counter
```

```

00170         break;
00171
00172     default: // This is a code that is not recognized (bad)
00173         Error_Handler();
00174     break;
00175 }
00176 }
00177 }
00178 }
00179
00180
00181 // -----
00182 // This sends the message to request data. The specific status requested is passed as arg
00183 // The IMD will then send a message with the same code and the data
00184 //
00185 void IMD_Request_Status(uint8_t Status){
00186     TxHeader.IDE = CAN_ID_EXT;
00187     TxHeader.ExtId = IMD_CAN_ID_Tx;
00188     TxHeader.DLC = 1;
00189     TxData[0] = Status;
00190
00191     if (HAL_CAN_AddTxMessage(&hcan2, &TxHeader, TxData, &TxMailbox) != HAL_OK){
00192         /* Transmission request Error */
00193         Error_Handler();
00194     }
00195     TxHeader.IDE = CAN_ID_STD;
00196 }
00197
00198 // -----
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210 // -----
00211 // Functions to check status
00212 // AFAIK the IMD will not send a CAN msg when the status changes - we need to constantly poll it
00213 //
00214
00215 // A lot of the messages will include status bits
00216 // Check for faults
00217 // Then check what the error is to display it for driver
00218 void IMD_Check_Status_Bits(uint8_t Data){
00219     // The touch energy bit will be 1 when connected to batteries
00220     // High uncertainty isn't also something we really care about
00221     // No idea about excitation pulse
00222     uint8_t mask = 0b10001111;
00223
00224     if ((Data & mask) != 0){
00225         // Send message to error handler to shutdown car
00226         //Trigger_Shutdown_Circuit();
00227
00228         if ((Data & Isolation_status_bit0) || (Data & Isolation_status_bit1)){
00229             // Isolation fault BAD
00230             // Want to display fault to dash
00231         }
00232
00233         // This function is only passed the first byte of info so we can't read the error flags
00234         // If we pass the entire data array in then we will read the wrong data
00235         // Need to explicitly request error flags and then read it
00236         // Use a bool to check if we have already requested error flags otherwise it will request
00237         // repeatedly
00238         if (Data & Hardware_Error){
00239             // TODO
00240             // display to dash
00241             if (!IMD_error_flags_requested){
00242                 IMD_Request_Status(Error_flags);
00243                 IMD_error_flags_requested = 1;
00244             }
00245
00246             if (Data & Low_Battery_Voltage){
00247                 // display low voltage on dash
00248                 // If the HV battery ever throws this error it is because of a disconnect
00249             }
00250             if (Data & High_Battery_Voltage){
00251                 // display high voltage on dash
00252                 // If the HV battery ever throws this error it is bad
00253             }
00254         }
00255         // Could check other faults we don't really care about

```

```

00256
00257     IMD_High_Uncertainty = Data & High_Uncertainty;
00258     // If we made it here then there is no error so exit to check rest of message
00259 }
00260
00261 // We need to look at the 2nd and 3rd bytes in the array for the error flags
00262 void IMD_Check_Error_Flags(uint8_t Data[]){
00263     // Need to check the bits to see what caused the hardware error
00264     // Want to display message to dash for safety reasons
00265     uint16_t IMD_Error_Flags = (Data[1] << 8) | Data[2];
00266
00267     // We want to shutdown if any of these are true
00268
00269     if (IMD_Error_Flags & Err_Vx1){
00270         // print to dash I guess
00271     }
00272     if (IMD_Error_Flags & Err_Vx2){
00273         // print to dash I guess
00274     }
00275     if (IMD_Error_Flags & Err_CH){
00276         // print to dash I guess
00277     }
00278     if (IMD_Error_Flags & Err_VxR){
00279         // print to dash I guess
00280     }
00281     if (IMD_Error_Flags & Err_Vexi){
00282         // print to dash I guess
00283     }
00284     if (IMD_Error_Flags & Err_Vpwr){
00285         // print to dash I guess
00286     }
00287     if (IMD_Error_Flags & Err_Watchdog){
00288         // print to dash I guess
00289     }
00290     if (IMD_Error_Flags & Err_clock){
00291         // print to dash I guess
00292     }
00293     if (IMD_Error_Flags & Err_temp){
00294         // print to dash I guess
00295     }
00296 }
00297
00298
00299
00300 // This is the function that will be called when a CAN message is received that has the isolation
00301 state data
00302 void IMD_Check_Isolation_State(uint8_t Data[]){
00303     uint16_t isolation = (Data[2] << 8) | Data[3];
00304
00305     // If the isolation is less than 500 Ohms / volt and the uncertainty is less than 5%
00306     if ( (isolation < 500) && (Data[4] <= 5) ){
00307
00308         // TODO disable shutdown circuit and display error
00309         IMD_High_Uncertainty = 0;
00310     }
00311
00312 }
00313
00314 // Not sure if we necessarily need to check isolation resistances
00315 // check isolation state will be much more important
00316 // We should, however, check this on startup
00317 void IMD_Check_Isolation_Resistances(uint8_t Data[]){
00318
00319     uint16_t Rp_resistance = (Data[2] << 8) | Data[3];
00320
00321     // If the isolation resistance between the positive terminal and the chassis
00322     // is less than 250 kOhms and the uncertainty is less than 5%
00323     if ( (Rp_resistance < 250) && (Data[4] <= 5) ){
00324
00325         // TODO disable shutdown circuit and display error
00326         IMD_High_Uncertainty = 0;
00327     }
00328
00329
00330     uint16_t Rn_resistance = (Data[5] << 8) | Data[6];
00331
00332     // If the isolation resistance between the negative terminal and the chassis
00333     // is less than 250 kOhms and the uncertainty is less than 5%
00334     if ( (Rn_resistance < 250) && (Data[7] <= 5) ){
00335
00336         // TODO disable shutdown circuit and display error
00337         IMD_High_Uncertainty = 0;
00338     }
00339 }
00340
00341

```

```

00342 void IMD_Check_Isolation_Capacitances(uint8_t Data[]){
00343
00344     // I don't know how useful this will be
00345
00346 }
00347
00348
00349 void IMD_Check_Voltages_Vp_and_Vn(uint8_t Data[]){
00350
00351     // This could potentially be useful on startup
00352
00353 }
00354
00355
00356 void IMD_Check_Battery_Voltage(uint8_t Data[]){
00357
00358     // This could be useful to compare with BMS and make sure things are working well
00359     // startup function really
00360
00361 }
00362
00363 void IMD_Check_Temperature(uint8_t Data[]){
00364     // TODO
00365
00366     // byte 1-4 in motorola
00367     IMD_Temperature = (Data[4] << 24) | (Data[3] << 16) | (Data[2] << 8) | Data[1];
00368
00369 }
00370
00371 // -----
00372 // These functions could check to see if stuff is safe to touch
00373
00374 void IMD_Check_Safety_Touch_Energy(uint8_t Data[]){
00375
00376     // I don't really know how to make use of these functions
00377
00378 }
00379
00380
00381 void IMD_Check_Safety_Touch_Current(uint8_t Data[]){
00382     // TODO
00383 }
00384
00385
00386
00387
00388
00389
00390 // -----
00391 // Data that could be checked on startup to make sure everything is good
00392
00393 void IMD_Check_Max_Battery_Working_Voltage(uint8_t Data[]){
00394     uint16_t Max_Battery_Voltage = (Data[1] << 8) | Data[2];
00395
00396     if (Max_Battery_Voltage != 571){
00397         // Max_Battery_Voltage not configured properly
00398     }
00399
00400 }
00401
00402
00403 // This function checks the part name of the IMD matches expected
00404 // The part name is split into 4 messages, each of 4 bytes
00405 // Because it is split over 4 messages, we need to compare only once we have read all messages
00406 void IMD_Check_Part_Name(uint8_t Data[]){
00407     // TODO
00408
00409     // This function will be called from the CAN msg parser
00410     // It will get the array of data bits. We need to check which part name
00411     // We then store the 4 bytes in an array of 32 bit int to compare at the end
00412
00413     switch (Data[0]){
00414         case Part_name_0:
00415             IMD_Read_Part_Name[0] = (Data[4] << 24) | (Data[3] << 16) | (Data[2] << 8) | Data[1];
00416             IMD_Part_Name_0_Set = 1;
00417             break;
00418         case Part_name_1:
00419             IMD_Read_Part_Name[1] = (Data[4] << 24) | (Data[3] << 16) | (Data[2] << 8) | Data[1];
00420             IMD_Part_Name_1_Set = 1;
00421             break;
00422         case Part_name_2:
00423             IMD_Read_Part_Name[2] = (Data[4] << 24) | (Data[3] << 16) | (Data[2] << 8) | Data[1];
00424             IMD_Part_Name_2_Set = 1;
00425             break;
00426         case Part_name_3:
00427             IMD_Read_Part_Name[3] = (Data[4] << 24) | (Data[3] << 16) | (Data[2] << 8) | Data[1];
00428             IMD_Part_Name_3_Set = 1;

```

```

00429         break;
00430     }
00431
00432     if (IMD_Part_Name_0_Set && IMD_Part_Name_1_Set && IMD_Part_Name_2_Set && IMD_Part_Name_3_Set) {
00433         IMD_Part_Name_Set = 1;
00434     }
00435
00436     if (IMD_Part_Name_Set) {
00437         // Check part number matches expected
00438         for (int i = 0; i < 4; ++i){
00439             if (IMD_Read_Part_Name[0] != IMD_Expected_Part_Name[0]){
00440                 //error
00441             }
00442         }
00443     }
00444 }
00445
00446 }
00447
00448 void IMD_Check_Version(uint8_t Data[]){
00449     // TODO
00450
00451     // This function will be called from the CAN msg parser
00452     // It will get the array of data bits. We need to check which firmware version
00453     // We then store the 4 bytes in an array of 32 bit int to compare at the end
00454
00455     switch (Data[0]){
00456         case Version_0:
00457             IMD_Read_Version[0] = (Data[3] << 16) | (Data[2] << 8) | Data[1];
00458             IMD_Version_0_Set = 1;
00459             break;
00460         case Version_1:
00461             IMD_Read_Version[1] = (Data[3] << 16) | (Data[2] << 8) | Data[1];
00462             IMD_Version_1_Set = 1;
00463             break;
00464         case Version_2:
00465             IMD_Read_Version[2] = (Data[3] << 16) | (Data[2] << 8) | Data[1];
00466             IMD_Version_2_Set = 1;
00467             break;
00468     }
00469
00470     if (IMD_Version_0_Set && IMD_Version_1_Set && IMD_Version_2_Set) {
00471         IMD_Version_Set = 1;
00472     }
00473
00474     if (IMD_Version_Set){
00475         // Check part number matches expected
00476         for (int i = 0; i < 3; ++i){
00477             if (IMD_Read_Version[0] != IMD_Expected_Version[0]){
00478                 //error
00479             }
00480         }
00481     }
00482 }
00483 }
00484
00485 // This function checks the serial number of the IMD matches expected
00486 // The part name is split into 4 messages, each of 4 bytes
00487 // Because it is split over 4 messages, we need to compare only once we have read all messages
00488 void IMD_Check_Serial_Number(uint8_t Data[]){
00489
00490     // This function will be called from the CAN msg parser
00491     // It will get the array of data bits. We need to check which serial number
00492     // We then store the 4 bytes in an array of 32 bit int to compare at the end
00493     // The serial number is found by concatenating 3 - 2 - 1 - 0
00494
00495     switch (Data[0]){
00496         case Serial_number_0:
00497             IMD_Read_Serial_Number[0] = (Data[1] << 24) | (Data[2] << 16) | (Data[3] << 8) | Data[4];
00498             IMD_Serial_Number_0_Set = 1;
00499             break;
00500         case Serial_number_1:
00501             IMD_Read_Serial_Number[1] = (Data[1] << 24) | (Data[2] << 16) | (Data[3] << 8) | Data[4];
00502             IMD_Serial_Number_1_Set = 1;
00503             break;
00504         case Serial_number_2:
00505             IMD_Read_Serial_Number[2] = (Data[1] << 24) | (Data[2] << 16) | (Data[3] << 8) | Data[4];
00506             IMD_Serial_Number_2_Set = 1;
00507             break;
00508         case Serial_number_3:
00509             IMD_Read_Serial_Number[3] = (Data[1] << 24) | (Data[2] << 16) | (Data[3] << 8) | Data[4];
00510             IMD_Serial_Number_3_Set = 1;
00511             break;
00512     }
00513
00514     if (IMD_Serial_Number_0_Set && IMD_Serial_Number_1_Set && IMD_Serial_Number_2_Set &&
00515         IMD_Serial_Number_3_Set){

```

```

00515     IMD_Serial_Number_Set = 1;
00516 }
00517
00518 if (IMD_Serial_Number_Set) {
00519     // Check serial number matches expected
00520     for (int i = 0; i < 4; ++i){
00521         if (IMD_Read_Serial_Number[i] != IMD_Expected_Serial_Number[i]){
00522             //error
00523         }
00524     }
00525 }
00526
00527 }
00528
00529 void IMD_Check_Uptime(uint8_t Data[]){
00530     // TODO
00531 }
00532
00533 void IMD_Startup(){
00534     // TODO
00535     // Run check for serial number, max voltage, and such
00536
00537     // The first check is the serial number
00538
00539     IMD_Request_Status(Serial_number_0);
00540     IMD_Request_Status(Serial_number_1);
00541     IMD_Request_Status(Serial_number_2);
00542     IMD_Request_Status(Serial_number_3);
00543
00544     IMD_Request_Status(Version_0);
00545     IMD_Request_Status(Version_1);
00546     IMD_Request_Status(Version_2);
00547
00548     IMD_Request_Status(Part_name_0);
00549     IMD_Request_Status(Part_name_1);
00550     IMD_Request_Status(Part_name_2);
00551     IMD_Request_Status(Part_name_3);
00552
00553     IMD_Request_Status(Max_battery_working_voltage);
00554     IMD_Request_Status(isolation_state);
00555     // Can check further things
00556 }
00557 }
00558
00559 void initIMD(void* args){
00560     uv_init_task_args* params = (uv_init_task_args*) args;
00561     uv_init_task_response response = {UV_OK, IMD, 0, NULL};
00562     vTaskDelay(100); //Pretend to be doing something for now
00563
00564     if(xQueueSendToBack(params->init_info_queue,&response,100) != pdPASS){
00565         //OOPS
00566         uvPanic("Failed to enqueue IMD OK Response",0);
00567     }
00568
00569     vTaskSuspend(params->meta_task_handle);
00570 }
00571 }
00572
00573
00574
00575

```

7.77 Core/Src/main.c File Reference

: Main program body

```

#include "main.h"
#include "adc.h"
#include "can.h"
#include "dma.h"
#include "tim.h"
#include "gpio.h"
#include "constants.h"
#include "bms.h"
#include "dash.h"
#include "imd.h"

```

```
#include "motor_controller.h"
#include "pdu.h"
#include "../FreeRTOS/Source\CMSIS_RTOS/cmsis_os.h"
Include dependency graph for main.c:
```

Macros

- #define __UV_FILENAME__ "main.c"
- #define MAIN_C
- #define DEBUG_CAN_IN_MAIN 0

Functions

- void [SystemClock_Config](#) (void)
System Clock Configuration.
- int [main](#) (void)
The application entry point.
- void [HAL_ADC_ConvCpltCallback](#) (ADC_HandleTypeDef *hadc)
- void [HAL_GPIO_EXTI_Callback](#) (uint16_t GPIO_Pin)
- void [HAL_ADC_LevelOutOfWindowCallback](#) (ADC_HandleTypeDef *hadc)
- void [HAL_TIM_PeriodElapsedCallback](#) (TIM_HandleTypeDef *htim)
Period elapsed callback in non blocking mode.
- void [Error_Handler](#) (void)
This function is executed in case of error occurrence.

Variables

- volatile uint32_t [adc_buf1](#) [5]
- uint16_t [adc1_APPS1](#)
- uint16_t [adc1_APPS2](#)
- uint16_t [adc1_BPS1](#)
- uint16_t [adc1_BPS2](#)
- uint16_t [adc1_pack_curr](#)
- volatile uint32_t [adc_buf2](#) [7]
- uint16_t [adc2_CoolantTemp1](#)
- uint16_t [adc2_CoolantTemp2](#)
- uint16_t [adc2_CoolantFlow](#)
- uint16_t [adc2_steering_pos](#)
- volatile uint32_t [adc_buf3](#) [4]
- uint16_t [adc3_damper_FL](#)
- uint16_t [adc3_damper_FR](#)
- uint16_t [adc3_damper_RL](#)
- uint16_t [adc3_damper_RR](#)
- TaskHandle_t [init_task_handle](#)
- [uv_init_struct init_settings](#)

7.77.1 Detailed Description

: Main program body

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [main.c](#).

7.77.2 Macro Definition Documentation

7.77.2.1 __UV_FILENAME__

```
#define __UV_FILENAME__ "main.c"
```

Definition at line [19](#) of file [main.c](#).

7.77.2.2 DEBUG_CAN_IN_MAIN

```
#define DEBUG_CAN_IN_MAIN 0
```

Definition at line [54](#) of file [main.c](#).

7.77.2.3 MAIN_C

```
#define MAIN_C
```

Definition at line [20](#) of file [main.c](#).

7.77.3 Function Documentation

7.77.3.1 Error_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

None	<input type="text"/>
----------------------	----------------------

Definition at line [365](#) of file [main.c](#).

Referenced by [HAL_ADC_MspInit\(\)](#), [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#), [HAL_CAN_RxFifo1MsgPendingCallback\(\)](#), [IMD_Parse_Message\(\)](#), [IMD_Request_Status\(\)](#), [initADCTask\(\)](#), [MX_ADC1_Init\(\)](#), [MX_ADC2_Init\(\)](#), [MX_ADC3_Init\(\)](#), [MX_CAN1_Init\(\)](#), [MX_CAN2_Init\(\)](#), [MX_TIM11_Init\(\)](#), [MX_TIM3_Init\(\)](#), [SystemClock_Config\(\)](#), [Update_Batt_Temp\(\)](#), [Update_RPM\(\)](#), and [Update_State_Of_Charge\(\)](#).

7.77.3.2 HAL_ADC_ConvCpltCallback()

```
void HAL_ADC_ConvCpltCallback (
    ADC_HandleTypeDef * hadc)
```

Definition at line 295 of file [main.c](#).

References [hadc1](#), [hadc2](#), [hadc3](#), and [processADCBuffer\(\)](#).

Here is the call graph for this function:

7.77.3.3 HAL_ADC_LevelOutOfWindowCallback()

```
void HAL_ADC_LevelOutOfWindowCallback (
    ADC_HandleTypeDef * hadc)
```

Definition at line 317 of file [main.c](#).

References [adc1_APPS1](#), [adc1_APPS2](#), and [hadc1](#).

7.77.3.4 HAL_GPIO_EXTI_Callback()

```
void HAL_GPIO_EXTI_Callback (
    uint16_t GPIO_Pin)
```

Definition at line 308 of file [main.c](#).

7.77.3.5 HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
    TIM_HandleTypeDef * htim)
```

Period elapsed callback in non blocking mode.

Note

This function is called when TIM1 interrupt took place, inside [HAL_TIM_IRQHandler\(\)](#). It makes a direct call to [HAL_IncTick\(\)](#) to increment a global variable "uwTick" used as application time base.

Parameters

<i>htim</i>	: TIM handle
-------------	--------------

Return values

<i>None</i>	
-------------	--

Definition at line 341 of file [main.c](#).

References [htim3](#).

7.77.3.6 main()

```
int main (
    void )
```

The application entry point.

Return values

<i>int</i>	
------------	--

Definition at line 111 of file [main.c](#).

References [adc1_APPS1](#), [adc1_APPS2](#), [handleCANbusError\(\)](#), [hcan2](#), [init_settings](#), [init_task_handle](#), [MX_ADC1_Init\(\)](#), [MX_ADC2_Init\(\)](#), [MX_ADC3_Init\(\)](#), [MX_CAN1_Init\(\)](#), [MX_CAN2_Init\(\)](#), [MX_DMA_Init\(\)](#), [MX_GPIO_Init\(\)](#), [MX_TIM11_Init\(\)](#), [MX_TIM3_Init\(\)](#), [SystemClock_Config\(\)](#), [SystemCoreClock](#), [TxData](#), [TxHeader](#), [TxMailbox](#), [Update_RPM\(\)](#), [uvAssert](#), and [uvInit\(\)](#).

Here is the call graph for this function:

7.77.3.7 SystemClock_Config()

```
void SystemClock_Config (
    void )
```

System Clock Configuration.

Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

Definition at line 236 of file [main.c](#).

References [Error_Handler\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:

7.77.4 Variable Documentation

7.77.4.1 adc1_APPS1

```
uint16_t adc1_APPS1
```

Definition at line 67 of file [main.c](#).

Referenced by [HAL_ADC_LevelOutOfWindowCallback\(\)](#), [initDrivingLoop\(\)](#), [main\(\)](#), [processADCBuffer\(\)](#), and [StartDrivingLoop\(\)](#).

7.77.4.2 adc1_APPS2

```
uint16_t adc1_APPS2
```

Definition at line 68 of file [main.c](#).

Referenced by [HAL_ADC_LevelOutOfWindowCallback\(\)](#), [initDrivingLoop\(\)](#), [main\(\)](#), [processADCBuffer\(\)](#), and [StartDrivingLoop\(\)](#).

7.77.4.3 adc1_BPS1

```
uint16_t adc1_BPS1
```

Definition at line [69](#) of file [main.c](#).

Referenced by [initDrivingLoop\(\)](#), [processADCBuffer\(\)](#), and [StartDrivingLoop\(\)](#).

7.77.4.4 adc1_BPS2

```
uint16_t adc1_BPS2
```

Definition at line [70](#) of file [main.c](#).

Referenced by [initDrivingLoop\(\)](#), [processADCBuffer\(\)](#), and [StartDrivingLoop\(\)](#).

7.77.4.5 adc1_pack_curr

```
uint16_t adc1_pack_curr
```

Definition at line [71](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.6 adc2_CoolantFlow

```
uint16_t adc2_CoolantFlow
```

Definition at line [76](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.7 adc2_CoolantTemp1

```
uint16_t adc2_CoolantTemp1
```

Definition at line [74](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.8 adc2_CoolantTemp2

```
uint16_t adc2_CoolantTemp2
```

Definition at line [75](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.9 adc2_steering_pos

```
uint16_t adc2_steering_pos
```

Definition at line [77](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.10 adc3_damper_FL

```
uint16_t adc3_damper_FL
```

Definition at line [80](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.11 adc3_damper_FR

```
uint16_t adc3_damper_FR
```

Definition at line [81](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.12 adc3_damper_RL

```
uint16_t adc3_damper_RL
```

Definition at line [82](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.13 adc3_damper_RR

```
uint16_t adc3_damper_RR
```

Definition at line [83](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#).

7.77.4.14 adc_buf1

```
volatile uint32_t adc_buf1[5]
```

Definition at line [65](#) of file [main.c](#).

Referenced by [processADCBuffer\(\)](#), and [StartADCTask\(\)](#).

7.77.4.15 adc_buf2

```
volatile uint32_t adc_buf2[7]
```

Definition at line 73 of file [main.c](#).

Referenced by [processADCBuffer\(\)](#), and [StartADCTask\(\)](#).

7.77.4.16 adc_buf3

```
volatile uint32_t adc_buf3[4]
```

Definition at line 79 of file [main.c](#).

Referenced by [processADCBuffer\(\)](#), and [StartADCTask\(\)](#).

7.77.4.17 init_settings

```
uv_init_struct init_settings
```

Definition at line 86 of file [main.c](#).

Referenced by [main\(\)](#).

7.77.4.18 init_task_handle

```
TaskHandle_t init_task_handle
```

Definition at line 85 of file [main.c](#).

Referenced by [main\(\)](#), and [uvInit\(\)](#).

7.78 main.c

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 #define __UV_FILENAME__ "main.c"
00020 #define MAIN_C
00021
00022 /* USER CODE END Header */
00023 /* Includes -----*/
00024 #include "main.h"
00025 #include "adc.h"
00026 #include "can.h"
00027 #include "dma.h"
00028 #include "tim.h"
00029 #include "gpio.h"
00030
00031 /* Private includes -----*/
00032 /* USER CODE BEGIN Includes */
00033
00034 // The CAN info is defined in constants
00035
00036 #include "constants.h"
00037 #include "bms.h"
00038 #include "dash.h"
00039 #include "imd.h"
00040 #include "motor_controller.h"
```

```
00041 #include "pdu.h"
00042 #include "../FreeRTOS/Source/CMSIS_RTOS/cmsis_os.h"
00043
00044
00045 /* USER CODE END Includes */
00046
00047 /* Private typedef -----*/
00048 /* USER CODE BEGIN PTD */
00049
00050 /* USER CODE END PTD */
00051
00052 /* Private define -----*/
00053 /* USER CODE BEGIN PD */
00054 #define DEBUG_CAN_IN_MAIN 0
00055 /* USER CODE END PD */
00056
00057 /* Private macro -----*/
00058 /* USER CODE BEGIN PM */
00059
00060 /* USER CODE END PM */
00061
00062 /* Private variables -----*/
00063
00064 /* USER CODE BEGIN PV */
00065 volatile uint32_t adc_buf1[5]; // ADC1 - high priority readings
00066
00067 uint16_t adc1_APPS1; //These are the locations for the sensor inputs for APPS and BPS
00068 uint16_t adc1_APPS2;
00069 uint16_t adc1_BPS1;
00070 uint16_t adc1_BPS2;
00071 uint16_t adc1_pack_curr;
00072
00073 volatile uint32_t adc_buf2[7]; // ADC2 - lower priority readings
00074 uint16_t adc2_CoolantTemp1;
00075 uint16_t adc2_CoolantTemp2;
00076 uint16_t adc2_CoolantFlow;
00077 uint16_t adc2_steering_pos;
00078
00079 volatile uint32_t adc_buf3[4];
00080 uint16_t adc3_damper_FL;
00081 uint16_t adc3_damper_FR;
00082 uint16_t adc3_damper_RL;
00083 uint16_t adc3_damper_RR;
00084
00085 TaskHandle_t init_task_handle;
00086 uv_init_struct init_settings;
00087
00088 //int adc_conv_complete_flag = 0;
00089
00090
00091 //State of da vehicle
00092 //enum uv_vehicle_state_t vehicle_state = UV_INIT; //TODO: Define new state machine logic
00093
00094 /* USER CODE END PV */
00095
00096 /* Private function prototypes -----*/
00097 void SystemClock_Config(void);
00098 /* USER CODE BEGIN PFP */
00099
00100 /* USER CODE END PFP */
00101
00102 /* Private user code -----*/
00103 /* USER CODE BEGIN 0 */
00104
00105 /* USER CODE END 0 */
00106
00111 int main(void)
00112 {
00113
00114 /* USER CODE BEGIN 1 */
00115
00116 /* USER CODE END 1 */
00117
00118 /* MCU Configuration-----*/
00119
00120 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
00121 HAL_Init();
00122
00123 /* USER CODE BEGIN Init */
00124 // The CAN clock must be enabled
00125 __HAL_RCC_CAN2_CLK_ENABLE();
00126 __HAL_RCC_CAN1_CLK_ENABLE();
00127 /* USER CODE END Init */
00128
00129 /* Configure the system clock */
00130 SystemClock_Config();
00131
```

```

00132 /* USER CODE BEGIN SysInit */
00133 uvAssert((1+1) == 2);
00134 SysTick_Config(SystemCoreClock / 1000);
00135 /* USER CODE END SysInit */
00136
00137 /* Initialize all configured peripherals */
00138 MX_GPIO_Init();
00139 MX_DMA_Init();
00140 MX_CAN2_Init();
00141 MX_ADC1_Init();
00142 MX_TIM3_Init();
00143 MX_ADC2_Init();
00144 MX_ADC3_Init();
00145 MX_CAN1_Init();
00146 MX_TIM11_Init();
00147 /* USER CODE BEGIN 2 */
00148 //HAL_ADC_Start_DMA(&hadcl1, (uint32_t*)adc_buf1, ADC1_BUF_LEN);
00149 //HAL_TIM_Base_Start_IT(&htim3); This is getting disabled, since measuring temp will now be an RTOS
task
00150 // Just to prove we reached this point in the code
00151 //MX_FREERTOS_Init();
00152 //ANYTHING WE NEED TO DO BEFORE THE KERNEL TAKES OVER SHOULD HAPPEN HERE:
00153
00154 //HAL_NVIC_PriorityGroupConfig( NVIC_PRIORITYGROUP_4 ); //;< This one is a fun function to do some
NVIC tomfoolery because we need to
00155 #if DEBUG_CAN_IN_MAIN
00156 while(1{
00157     HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_14);
00158     TxData[0] = 0b10101010;
00159     TxData[1] = 0b10101010;
00160     TxData[2] = 0b10101010;
00161     TxData[3] = 1;
00162     TxData[4] = 2;
00163     TxData[5] = 3;
00164     TxData[6] = 0b10101010;
00165     TxData[7] = 0b10101010;
00166
00167     HAL_StatusTypeDef can_send_status;
00168
00169             //vTaskDelay(400);
00170
00171     HAL_Delay(200);
00172
00173     TxHeader.IDE = CAN_ID_EXT;
00174     TxHeader.ExtId = 0x1234;
00175
00176
00177     TxHeader.DLC = 8;
00178
00179     //No need for a critical section when we out here in main without a scheduler running :
00180     //taskENTER_CRITICAL();
00181     can_send_status = HAL_CAN_AddTxMessage(&hcan2, &TxHeader, TxData, &TxMailbox);
00182     //taskEXIT_CRITICAL();
00183
00184     if (can_send_status != HAL_OK){
00185         /* Transmission request Error */
00186         //uvPanic("Unable to Transmit CAN msg",can_send_status);
00187         handleCANbusError(&hcan2, 0);
00188     }
00189 }
00190
00191 }
00192 #endif
00193 /* USER CODE END 2 */
00194
00195 /* Infinite loop */
00196 /* USER CODE BEGIN WHILE */
00197
00198 BaseType_t x_task_return =
xTaskCreate(uvInit,"init",512,&init_settings,osPriorityNormal,&init_task_handle);
00199     if(x_task_return != pdPASS){
00200         while(1{
00201             //Program hangs itself, like bro, we couldnt even create the INITIALISATION task, thats
fucked
00202         }
00203     }
00204
00205 vTaskStartScheduler();
00206
00207     //Update_Batt_Temp(69); // temp debugging
00208
00209
00210     while (1) //we should deadass never reach this point in the code lol
00211 {
00212     // For debugging purposes
00213     HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_13);
00214

```

```

00215 // into string and store in dma_result_buffer character array
00216     Update_RPM(adcl_APPS1);
00217     HAL_Delay(1000);
00218     Update_RPM(adcl_APPS2);
00219     HAL_Delay(1000);
00220
00221
00222
00223
00224
00225     /* USER CODE END WHILE */
00226
00227     /* USER CODE BEGIN 3 */
00228 }
00229 /* USER CODE END 3 */
00230 }
00231
00232 void SystemClock_Config(void)
00233 {
00234     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
00235     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
00236
00237     __HAL_RCC_PWR_CLK_ENABLE();
00238     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
00239
00240     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00241     RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
00242     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00243     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00244     RCC_OscInitStruct.PLL.PLLM = 25;
00245     RCC_OscInitStruct.PLL.PLLN = 336;
00246     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
00247     RCC_OscInitStruct.PLL.PLLQ = 4;
00248     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
00249     {
00250         Error_Handler();
00251     }
00252
00253     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
00254             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
00255     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00256     RCC_ClkInitStruct.AHCLKDivider = RCC_SYSCLK_DIV1;
00257     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
00258     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
00259
00260     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
00261     {
00262         Error_Handler();
00263     }
00264
00265     /* USER CODE BEGIN 4 */
00266
00267     // called when timer 3 period elapsed
00268     //void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
00269     //
00270     //    // check that timer 3 created the interrupt
00271     //    if (htim == &htim3){
00272     //
00273     //        // start a single round of ADC2 conversions
00274     //        HAL_ADC_Start_DMA(&hadc2, (uint32_t*)adc_buf2, ADC2_BUF_LEN);
00275     //
00276     //        // restart timer
00277     //        HAL_TIM_Base_Start_IT(&htim3);
00278     //    }
00279     //}
00280
00281
00282     // Called when adc dma buffer is completely filled
00283     void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
00284         if(hadc == &hadc1){
00285             processADCBuffer(1);
00286         }else if(hadc == &hadc2){
00287             processADCBuffer(2);
00288         }else if(hadc == &hadc3){
00289             processADCBuffer(3);
00290         }
00291     }
00292
00293
00294     // EXTI gpio pin a0 External Interrupt ISR Handler
00295     void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
00296     {
00297         if(GPIO_Pin == GPIO_PIN_0) // If The INT Source Is EXTI Line0 (A0 Pin)
00298         {
00299             //ready_to_drive = 1;
00300         }
00301
00302
00303
00304
00305
00306
00307     // EXTI gpio pin a0 External Interrupt ISR Handler
00308     void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
00309     {
00310         if(GPIO_Pin == GPIO_PIN_0) // If The INT Source Is EXTI Line0 (A0 Pin)
00311         {
00312             //ready_to_drive = 1;
00313         }
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
0194
```

```

00314 }
00315
00316 // Analog Watchdog Out-of-Range handler, ADC conversion values range from 0 to 4095?? why does this
00317 // exist still - Byron
00318 void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc) {
00319     if(hadc->Instance == ADC1){
00320         // triggered for voltages below 0.5V (below 400) or above 4.5V (above 3674)
00321         HAL_GPIO_WritePin(Red_LED_GPIO_Port,Red_LED_Pin, GPIO_PIN_SET);
00322         HAL_ADC_Stop_DMA(&hadc1);
00323         adc1_APPS1 = 0;
00324         adc1_APPS2 = 0;
00325     if(hadc->Instance == ADC2){
00326         // handle coolant sensors short error
00327     }
00328 }
00329
00330
00331 /* USER CODE END 4 */
00332
00333 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
00334 {
00335     /* USER CODE BEGIN Callback 0 */
00336
00337     /* USER CODE END Callback 0 */
00338     if (htim->Instance == TIM1) {
00339         HAL_IncTick();
00340     }
00341     /* USER CODE BEGIN Callback 1 */
00342     if (htim == &htim3){
00343
00344         /* start a single round of ADC2 conversions
00345         //HAL_ADC_Start_DMA(&hadc2, (uint32_t*)adc_buf2, ADC2_BUF_LEN);
00346
00347         // restart timer
00348         HAL_TIM_Base_Start_IT(&htim3);
00349     }
00350     /* USER CODE END Callback 1 */
00351 }
00352
00353 void Error_Handler(void)
00354 {
00355     /* USER CODE BEGIN Error_Handler_Debug */
00356     HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12); //Me when the error is not being handled
00357     /* User can add his own implementation to report the HAL error return state */
00358     __disable_irq();
00359     while (1)
00360     {
00361     }
00362     /* USER CODE END Error_Handler_Debug */
00363 }
00364
00365 #ifdef USE_FULL_ASSERT
00366 void assert_failed(uint8_t *file, uint32_t line)
00367 {
00368     /* USER CODE BEGIN 6 */
00369     /* User can add his own implementation to report the file name and line number,
00370      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
00371     /* USER CODE END 6 */
00372 }
00373
00374 #endif /* USE_FULL_ASSERT */

```

7.79 Core/Src/motor_controller.c File Reference

```

#include "motor_controller.h"
#include "can.h"
#include "uvfr_utils.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "uvfr_settings.h"
#include "cmsis_os.h"
Include dependency graph for motor_controller.c:

```

Macros

- #define mc_settings (current_vehicle_settings->mc_settings)

Functions

- `uint16_t sendTorqueToMotorController (float T_filtered)`
Sends a filtered torque command to the motor controller over CAN.
- `void MC_Request_Data (uint8_t RegID)`
Sends a CAN request to retrieve a specific register from the motor controller.
- `uv_status MC_Set_Param (uint8_t RegID, uint16_t d)`
Sends a parameter write command to the motor controller.
- `uv_status MC_SetAndVerify_Param (uint8_t reg_id, uint16_t set_val)`
Writes a value to a motor controller register and verifies the write.
- `void Parse_Bamocar_Response (uv_CAN_msg *msg)`
Parses a 32-bit value from a CAN message in little-endian format.
- `void ProcessMotorControllerResponse (uv_CAN_msg *msg)`
Processes a CAN response from the motor controller.
- `void MC_EnableCyclicSpeedTransmission (uint8_t interval_ms)`
Enables or disables cyclic transmission of selected motor controller parameters.
- `void MC_Startup (void *args)`
Initializes the motor controller.
- `void MC_Shutdown (void)`
Safely shuts down the motor controller.

Variables

- `uv_vehicle_settings * current_vehicle_settings`
- `QueueHandle_t CAN_Rx_Queue`
- `uv_CAN_msg last_mc_response`
- `TickType_t last_driver_input_time = 0`
- `int16_t mc_speed_rpm = 0`
- `int16_t mc_current = 0`
- `int16_t mc_torque_cmd = 0`
- `int16_t mc_motor_temp = 0`
- `int16_t mc_igbt_temp = 0`
- `motor_controller_settings mc_default_settings`

7.79.1 Macro Definition Documentation

7.79.1.1 mc_settings

```
#define mc_settings (current_vehicle_settings->mc_settings)
```

Definition at line 16 of file `motor_controller.c`.

Referenced by `MC_EnableCyclicSpeedTransmission()`, `MC_Request_Data()`, `MC_Set_Param()`, `MC_Startup()`, and `sendTorqueToMotorController()`.

7.79.2 Function Documentation

7.79.2.1 MC_EnableCyclicSpeedTransmission()

```
void MC_EnableCyclicSpeedTransmission (
    uint8_t interval_ms)
```

Enables or disables cyclic transmission of selected motor controller parameters.

Sends a series of CAN messages to configure the Bamocar controller to periodically transmit values such as speed, current, torque, temperatures, and error flags.

The message format is: [0x3D, RegID, interval_ms] Where interval_ms is the repeat interval in milliseconds (1–254). Passing 0xFF disables transmission for that register.

Parameters

<i>interval_ms</i>	Transmission interval in milliseconds (1–254). Use 0xFF to disable.
--------------------	---

Definition at line 407 of file [motor_controller.c](#).

References [CURRENT_ACTUAL](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [igbt_temperature](#), [LOGIMAP_ERRORS](#), [M_out](#), [max_motor_temp](#), [mc_settings](#), [motor_temperature](#), [uv_CAN_msg::msg_id](#), [N_actual](#), [uvSendCanMSG\(\)](#), and [warning_motor_temp](#).

Referenced by [MC_Shutdown\(\)](#), and [MC_Startup\(\)](#).

Here is the call graph for this function:

7.79.2.2 MC_Request_Data()

```
void MC_Request_Data (
    uint8_t RegID)
```

Sends a CAN request to retrieve a specific register from the motor controller.

The request message is formatted as: [0x3D, RegID, 0], which should trigger an immediate reply.

Definition at line 110 of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [mc_settings](#), [uv_CAN_msg::msg_id](#), [UV_OK](#), and [uvSendCanMSG\(\)](#).

Referenced by [MC_SetAndVerify_Param\(\)](#), [MC_Shutdown\(\)](#), and [MC_Startup\(\)](#).

Here is the call graph for this function:

7.79.2.3 MC_Set_Param()

```
uv_status MC_Set_Param (
    uint8_t RegID,
    uint16_t d)
```

Sends a parameter write command to the motor controller.

Constructs a 3-byte CAN message to set the value of a specific register on the motor controller. The message format is: [RegID, LSB(data), MSB(data), 0x00]

This function does not verify the result—use [MC_SetAndVerify_Param\(\)](#) for value confirmation.

Parameters

<i>RegID</i>	The register address to be written.
<i>d</i>	The 16-bit value to write to the register.

Returns

UV_OK on success, UV_ERROR if the CAN transmission fails.

Definition at line 141 of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [mc_settings](#), [uv_CAN_msg::msg_id](#), [UV_ERROR](#), [UV_OK](#), and [uvSendCanMSG\(\)](#).

Referenced by [MC_SetAndVerify_Param\(\)](#), [MC_Shutdown\(\)](#), and [MC_Startup\(\)](#).

Here is the call graph for this function:

7.79.2.4 MC_SetAndVerify_Param()

```
uv_status MC_SetAndVerify_Param (
    uint8_t reg_id,
    uint16_t set_val)
```

Writes a value to a motor controller register and verifies the write.

Sends a parameter set command to the specified register, then requests the value back and compares the response to ensure the write succeeded. Uses little-endian 16-bit parsing for verification.

This is a safer alternative to [MC_Set_Param\(\)](#) when reliability is critical.

Parameters

<i>reg_id</i>	The target register address to write.
<i>set_val</i>	The 16-bit value to write to the register.

Returns

UV_OK if the write and verification succeeded, otherwise UV_ERROR.

Definition at line [175](#) of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), [last_mc_response](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [UV_ERROR](#), and [UV_OK](#).

Here is the call graph for this function:

7.79.2.5 MC_Shutdown()

```
void MC_Shutdown (
    void )
```

Safely shuts down the motor controller.

This function stops all periodic CAN feedback, zeroes the torque or speed command (depending on control mode), disables the motor controller, and optionally requests the error/warning register for post-shutdown diagnostics.

It ensures the controller is left in a known, safe state after driving stops. Use this before powering down or transitioning to an inactive state.

Definition at line [548](#) of file [motor_controller.c](#).

References [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), and [motor_controller_errors_warnings](#).

Referenced by [performSafetyChecks\(\)](#), and [uvSecureVehicle\(\)](#).

Here is the call graph for this function:

7.79.2.6 MC_Startup()

```
void MC_Startup (
    void * args)
```

Initializes the motor controller.

This function performs all necessary startup steps to prepare the Bamocar motor controller for operation. It is typically called during system initialization from [uvfr_utils.c](#).

The routine performs the following actions:

- Toggles a GPIO pin for debug indication
- Registers a CAN RX handler for controller responses
- Enables cyclic transmission of critical feedback parameters
- Sends initialization commands (based on Bamocar Example 11)
- Requests metadata (serial number, firmware version)
- Optionally sets and verifies controller parameters
- Sends status to the system init queue
- Suspends itself after completion

Parameters

<i>args</i>	Pointer to uv_init_task_args , used to send back init status.
-------------	---

Definition at line [458](#) of file [motor_controller.c](#).

References [uv_init_task_response::device](#), [FIRMWARE_VERSION_REGISTER](#), [uv_init_task_args::init_info_queue](#), [insertCANMessageHandler\(\)](#), [MC_EnableCyclicSpeedTransmission\(\)](#), [MC_Request_Data\(\)](#), [MC_Set_Param\(\)](#), [mc_settings](#), [MOTOR_CONTROLLER](#), [ProcessMotorControllerResponse\(\)](#), [SERIAL_NUMBER_REGISTER](#), [uv_init_task_response::status](#), and [UV_OK](#).

Referenced by [uvInit\(\)](#).

Here is the call graph for this function:

7.79.2.7 Parse_Bamocar_Response()

```
void Parse_Bamocar_Response (
    uv_CAN_msg * msg)
```

Parses a 32-bit value from a CAN message in little-endian format.

This example assumes that the data bytes are stored as: data[0] = LSB, data[3] = MSB.

Definition at line [221](#) of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), and [uv_CAN_msg::dlc](#).

7.79.2.8 ProcessMotorControllerResponse()

```
void ProcessMotorControllerResponse (
    uv_CAN_msg * msg)
```

Processes a CAN response from the motor controller.

This function decodes a received CAN message by examining the register ID in the first byte (data[0]) and handling the response accordingly.

It performs little-endian parsing to extract values for speed, current, torque, temperatures, and error flags. Critical errors will trigger uvPanic() via the error handler.

The full message is also stored in a global buffer (last_mc_response) for later access and verification routines.

Parameters

<i>msg</i>	Pointer to the received CAN message from the motor controller.
------------	--

Definition at line 324 of file [motor_controller.c](#).

References [CURRENT_ACTUAL](#), [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [igbt_temperature](#), [last_mc_response](#), [LOGIMAP_ERRORS](#), [LOGIMAP_IO](#), [M_out](#), [mc_current](#), [mc_igbt_temp](#), [mc_motor_temp](#), [mc_speed_rpm](#), [mc_torque_cmd](#), [motor_controller_errors_warnings](#), [motor_temperature](#), [N_actual](#), and [POS_ACTUAL](#).

Referenced by [MC_Startup\(\)](#).

7.79.2.9 sendTorqueToMotorController()

```
uint16_t sendTorqueToMotorController (
    float T_filtered)
```

Sends a filtered torque command to the motor controller over CAN.

This function scales the input torque (in Nm) to the Bamocar's expected format, clamps it within the valid operating range, and transmits it as a direct torque command using the N_set register. The final 16-bit value is sent via CAN using the configured transmit ID.

Parameters

<i>T_filtered</i>	The final torque value in Nm after filtering (typically 0–230 Nm).
-------------------	--

Returns

0 if successful, 1 if transmission failed.

Definition at line 69 of file [motor_controller.c](#).

References [uv_CAN_msg::data](#), [uv_CAN_msg::dlc](#), [uv_CAN_msg::flags](#), [mc_settings](#), [uv_CAN_msg::msg_id](#), [N_set](#), [UV_OK](#), and [uvSendCanMSG\(\)](#).

Referenced by [StartDrivingLoop\(\)](#).

Here is the call graph for this function:

7.79.3 Variable Documentation

7.79.3.1 CAN_Rx_Queue

```
QueueHandle_t CAN_Rx_Queue [extern]
```

7.79.3.2 current_vehicle_settings

```
uv_vehicle_settings* current_vehicle_settings [extern]
```

Definition at line 25 of file [uvfr_settings.c](#).

Referenced by [setupDefaultSettings\(\)](#), [uvLoadSettingsFromFlash\(\)](#), and [uvSettingsInit\(\)](#).

7.79.3.3 last_driver_input_time

```
TickType_t last_driver_input_time = 0
```

Definition at line 24 of file [motor_controller.c](#).

Referenced by [StartDrivingLoop\(\)](#).

7.79.3.4 last_mc_response

```
uv_CAN_msg last_mc_response
```

Definition at line 22 of file [motor_controller.c](#).

Referenced by [MC_SetAndVerify_Param\(\)](#), and [ProcessMotorControllerResponse\(\)](#).

7.79.3.5 mc_current

```
int16_t mc_current = 0
```

Definition at line 28 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.79.3.6 mc_default_settings

```
motor_controller_settings mc_default_settings
```

Initial value:

```
= {
    .can_id_tx          = 0x201,
    .can_id_rx          = 0x181,
    .mc_CAN_timeout     = 2,
    .proportional_gain  = 10,
    .integral_time_constant = 400,
    .integral_memory_max = 60,

    .max_speed           = 12357,
    .max_current          = 13107,
    .cont_current          = 7864,
    .max_torque            = 32767,
    .max_motor_temp        = 32767,
    .warning_motor_temp    = 32767,

    .mc_bus              = CAN_BUS_1
}
```

Definition at line 38 of file [motor_controller.c](#).

Referenced by [setupDefaultSettings\(\)](#), and [uvResetFlashToDefault\(\)](#).

7.79.3.7 mc_igbt_temp

```
int16_t mc_igbt_temp = 0
```

Definition at line 31 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.79.3.8 mc_motor_temp

```
int16_t mc_motor_temp = 0
```

Definition at line 30 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.79.3.9 mc_speed_rpm

```
int16_t mc_speed_rpm = 0
```

Definition at line 27 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.79.3.10 mc_torque_cmd

```
int16_t mc_torque_cmd = 0
```

Definition at line 29 of file [motor_controller.c](#).

Referenced by [ProcessMotorControllerResponse\(\)](#).

7.80 motor_controller.c

[Go to the documentation of this file.](#)

```
00001 /* motor_controller.c */
00002
00003 #include "motor_controller.h"
00004 #include "can.h"           // For uvSendCanMSG, uv_CAN_msg, etc.
00005 #include "uvfr_utils.h"    // For uvPanic, etc.
00006 #include <stdlib.h>
00007 #include <string.h>
00008 #include <stdio.h>
00009 #include "uvfr_settings.h"
00010 #include "cmsis_os.h"       // For vTaskSuspend
00011
00012 extern uv_vehicle_settings* current_vehicle_settings;
00013 extern QueueHandle_t CAN_Rx_Queue;
00014
00015 // Redirect all mc_settings.x to actual config struct
00016 #define mc_settings (current_vehicle_settings->mc_settings)
00017 //register for motor controller errors and warnings so we can make them cyclic
00018 //##define motor_controller_errors_warnings 0x82
00019
00020
00021 //global variable for the last mc response
00022 uv_CAN_msg last_mc_response;
00023 //global variable for timeout
```