

EVRTOSProject

V1

Generated by Doxygen 1.8.16

1 Deprecated List	1
2 Module Index	3
2.1 Modules	3
3 Data Structure Index	5
3.1 Data Structures	5
4 File Index	7
4.1 File List	7
5 Module Documentation	9
5.1 State Engine	9
5.1.1 Detailed Description	10
5.1.2 Macro Definition Documentation	10
5.1.2.1 MAX_NUM_MANAGED_TASKS	10
5.1.3 Typedef Documentation	10
5.1.3.1 state_change_daemon_args	10
5.1.4 Function Documentation	11
5.1.4.1 uvSVCTaskManager()	11
5.1.5 Variable Documentation	11
5.1.5.1 _next_svc_task_id	11
5.1.5.2 _next_task_id	12
5.1.5.3 _svc_task_register	12
5.1.5.4 _task_register	12
5.1.5.5 default_os_settings	12
5.1.5.6 previous_state	12
5.1.5.7 SCD_active	13
5.1.5.8 scd_handle_ptr	13
5.1.5.9 state_change_queue	13
5.1.5.10 svc_task_manager	13
5.1.5.11 task_manager	13
5.1.5.12 task_name_lut	13
5.1.5.13 vehicle_state	13
5.2 State Engine API	14
5.2.1 Detailed Description	16
5.2.2 Macro Definition Documentation	16
5.2.2.1 UV_TASK_AWAITING_DELETION	16
5.2.2.2 UV_TASK_DEADLINE_FIRM	16
5.2.2.3 UV_TASK_DEADLINE_HARD	16
5.2.2.4 UV_TASK_DEADLINE_MASK	16
5.2.2.5 UV_TASK_DEADLINE_NOT_ENFORCED	17
5.2.2.6 UV_TASK_DEFER_DELETION	17
5.2.2.7 UV_TASK_DELAYING	17

5.2.2.8 UV_TASK_DORMANT_SVC	17
5.2.2.9 UV_TASK_ERR_IN_CHILD	17
5.2.2.10 UV_TASK_GENERIC_SVC	17
5.2.2.11 UV_TASK_IS_CHILD	18
5.2.2.12 UV_TASK_IS_ORPHAN	18
5.2.2.13 UV_TASK_IS_PARENT	18
5.2.2.14 UV_TASK_LOG_MEM_USAGE	18
5.2.2.15 UV_TASK_LOG_START_STOP_TIME	18
5.2.2.16 UV_TASK_MANAGER_MASK	18
5.2.2.17 UV_TASK_MISSION_CRITICAL	19
5.2.2.18 UV_TASK_PERIODIC_SVC	19
5.2.2.19 UV_TASK_PRIO_INCREMENTATION	19
5.2.2.20 UV_TASK_SCD_IGNORE	19
5.2.2.21 UV_TASK_VEHICLE_APPLICATION	19
5.2.2.22 uvTaskDelay	19
5.2.2.23 uvTaskDelayUntil	20
5.2.2.24 uvTaskIsDelaying	20
5.2.2.25 uvTaskResetDelayBit	20
5.2.2.26 uvTaskResetDeletionBit	21
5.2.2.27 uvTaskSetDelayBit	21
5.2.2.28 uvTaskSetDeletionBit	21
5.2.3 Typedef Documentation	21
5.2.3.1 task_management_info	21
5.2.3.2 task_priority	21
5.2.3.3 task_status_block	22
5.2.3.4 uv_os_settings	22
5.2.3.5 uv_scd_response	22
5.2.3.6 uv_task_cmd	22
5.2.3.7 uv_task_info	22
5.2.3.8 uv_task_status	22
5.2.3.9 uv_vehicle_state	23
5.2.4 Enumeration Type Documentation	23
5.2.4.1 task_priority	23
5.2.4.2 uv_scd_response_e	23
5.2.4.3 uv_task_cmd_e	24
5.2.4.4 uv_task_state_t	24
5.2.4.5 uv_vehicle_state_t	24
5.2.5 Function Documentation	25
5.2.5.1 changeVehicleState()	25
5.2.5.2 uvCreateTask()	26
5.2.5.3 uvDeInitStateEngine()	26
5.2.5.4 uvInitStateEngine()	26

5.2.5.5 uvStartStateMachine()	27
5.3 State Engine Internals	28
5.3.1 Detailed Description	29
5.3.2 Function Documentation	29
5.3.2.1 __uvPanic()	29
5.3.2.2 _stateChangeDaemon()	30
5.3.2.3 _uvValidateSpecificTask()	31
5.3.2.4 addTaskToTaskRegister()	31
5.3.2.5 killEmAll()	32
5.3.2.6 killSelf()	32
5.3.2.7 processSCDMsg()	32
5.3.2.8 suspendSelf()	33
5.3.2.9 uvAbortTaskDeletion()	33
5.3.2.10 uvCreateServiceTask()	33
5.3.2.11 uvDeleteSVCTask()	34
5.3.2.12 uvDeleteTask()	34
5.3.2.13 uvGetTaskFromName()	34
5.3.2.14 uvGetTaskFromRTOSHandle()	34
5.3.2.15 uvInvokeSCD()	35
5.3.2.16 uvKillTaskViolently()	35
5.3.2.17 uvRestartSVCTask()	35
5.3.2.18 uvScheduleTaskDeletion()	36
5.3.2.19 uvSecureVehicle()	36
5.3.2.20 uvStartSVCTask()	36
5.3.2.21 uvStartTask()	37
5.3.2.22 uvSuspendSVCTask()	37
5.3.2.23 uvSuspendTask()	37
5.3.2.24 uvTaskCrashHandler()	38
5.3.2.25 uvTaskManager()	38
5.3.2.26 uvValidateManagedTasks()	39
5.4 UVFR Utilities	40
5.4.1 Detailed Description	40
5.5 Utility Macros	41
5.5.1 Detailed Description	41
5.5.2 Macro Definition Documentation	41
5.5.2.1 _BV	41
5.5.2.2 _BV_16	42
5.5.2.3 _BV_32	42
5.5.2.4 _BV_8	42
5.5.2.5 deserializeBigE16	42
5.5.2.6 deserializeBigE32	42
5.5.2.7 deserializeSmallE16	43

5.5.2.8 deserializeSmalle32	43
5.5.2.9 endianSwap	43
5.5.2.10 endianSwap16	43
5.5.2.11 endianSwap32	43
5.5.2.12 endianSwap8	44
5.5.2.13 false	44
5.5.2.14 isPowerOfTwo	44
5.5.2.15 safePtrRead	44
5.5.2.16 safePtrWrite	44
5.5.2.17 serializeBigE16	45
5.5.2.18 serializeBigE32	45
5.5.2.19 serializeSmalle16	45
5.5.2.20 serializeSmalle32	45
5.5.2.21 setBits	45
5.5.2.22 true	46
5.6 UVFR Vehicle Commands	47
5.7 CMSIS	48
5.7.1 Detailed Description	48
5.8 Stm32f4xx_system	49
5.8.1 Detailed Description	49
5.9 STM32F4xx_System_Private_Includes	50
5.9.1 Detailed Description	50
5.9.2 Macro Definition Documentation	50
5.9.2.1 HSE_VALUE	50
5.9.2.2 HSI_VALUE	50
5.10 STM32F4xx_System_Private_TypesDefinitions	51
5.11 STM32F4xx_System_Private_Defines	52
5.12 STM32F4xx_System_Private_Macros	53
5.13 STM32F4xx_System_Private_Variables	54
5.13.1 Detailed Description	54
5.13.2 Variable Documentation	54
5.13.2.1 AHBPrescTable	54
5.13.2.2 APBPrescTable	54
5.13.2.3 SystemCoreClock	54
5.14 STM32F4xx_System_Private_FunctionPrototypes	55
5.15 STM32F4xx_System_Private_Functions	56
5.15.1 Detailed Description	56
5.15.2 Function Documentation	56
5.15.2.1 SystemCoreClockUpdate()	56
5.15.2.2 SystemInit()	56

6.1 access_control_info Union Reference	57
6.1.1 Detailed Description	57
6.1.2 Field Documentation	57
6.1.2.1 bin_semaphore	57
6.1.2.2 mutex	57
6.1.2.3 semaphore	58
6.2 bms_settings_t Struct Reference	58
6.2.1 Detailed Description	58
6.2.2 Field Documentation	58
6.2.2.1 mc_CAN_timeout	58
6.3 daq_child_task Struct Reference	58
6.3.1 Detailed Description	59
6.3.2 Field Documentation	59
6.3.2.1 meta_task_handle	59
6.3.2.2 param_list	59
6.3.2.3 period	59
6.3.2.4 treenode	59
6.4 daq_datapoint Struct Reference	60
6.4.1 Detailed Description	60
6.4.2 Field Documentation	60
6.4.2.1 can_id	60
6.4.2.2 period	60
6.4.2.3 type	60
6.5 daq_loop_args Struct Reference	61
6.5.1 Detailed Description	61
6.5.2 Field Documentation	61
6.5.2.1 datapoints	61
6.5.2.2 minimum_daq_period	61
6.5.2.3 padding	61
6.5.2.4 padding2	62
6.5.2.5 throttle_daq_to_preserve_performance	62
6.6 daq_param_list_node Struct Reference	62
6.6.1 Detailed Description	62
6.6.2 Field Documentation	62
6.6.2.1 next	62
6.6.2.2 param_idx	63
6.7 driving_loop_args Struct Reference	63
6.7.1 Detailed Description	63
6.7.2 Field Documentation	63
6.7.2.1 absolute_max_acc_pwr	64
6.7.2.2 absolute_max_accum_current	64
6.7.2.3 absolute_max_motor_rpm	64

6.7.2.4 absolute_max_motor_torque	64
6.7.2.5 accum_regen_soc_threshold	64
6.7.2.6 apps_bottom	65
6.7.2.7 apps_implausibility_recovery_threshold	65
6.7.2.8 apps_plausibility_check_threshold	65
6.7.2.9 apps_top	65
6.7.2.10 bps_implausibility_recovery_threshold	65
6.7.2.11 bps_plausibility_check_threshold	66
6.7.2.12 dmodes	66
6.7.2.13 max_accum_current_5s	66
6.7.2.14 max_apps_offset	66
6.7.2.15 max_apps_value	66
6.7.2.16 max_BPS_value	67
6.7.2.17 min_apps_offset	67
6.7.2.18 min_apps_value	67
6.7.2.19 min_BPS_value	67
6.7.2.20 num_driving_modes	67
6.7.2.21 period	68
6.7.2.22 regen_rpm_cutoff	68
6.8 drivingLoopArgs Struct Reference	68
6.8.1 Detailed Description	68
6.9 drivingMode Struct Reference	68
6.9.1 Detailed Description	69
6.9.2 Field Documentation	69
6.9.2.1 control_map_fn	69
6.9.2.2 dm_name	69
6.9.2.3 flags	69
6.9.2.4 map_fn_params	70
6.9.2.5 max_acc_pwr	70
6.9.2.6 max_current	70
6.9.2.7 max_motor_torque	70
6.10 drivingModeParams Union Reference	70
6.10.1 Detailed Description	70
6.11 exp_torque_map_args Struct Reference	71
6.11.1 Detailed Description	71
6.11.2 Field Documentation	71
6.11.2.1 gamma	71
6.11.2.2 offset	71
6.12 linear_torque_map_args Struct Reference	71
6.12.1 Detailed Description	72
6.12.2 Field Documentation	72
6.12.2.1 offset	72

6.12.2.2 slope	72
6.13 motor_controller_settings Struct Reference	72
6.13.1 Detailed Description	72
6.13.2 Field Documentation	73
6.13.2.1 mc_CAN_timeout	73
6.14 p_status Struct Reference	73
6.14.1 Detailed Description	73
6.14.2 Field Documentation	73
6.14.2.1 activation_time	73
6.14.2.2 peripheral_status	74
6.15 rbnode Struct Reference	74
6.15.1 Detailed Description	74
6.15.2 Field Documentation	74
6.15.2.1 color	74
6.15.2.2 data	75
6.15.2.3 left	75
6.15.2.4 parent	75
6.15.2.5 right	75
6.16 rbtree Struct Reference	76
6.16.1 Detailed Description	76
6.16.2 Field Documentation	76
6.16.2.1 compare	76
6.16.2.2 count	76
6.16.2.3 destroy	77
6.16.2.4 min	77
6.16.2.5 nil	77
6.16.2.6 print	77
6.16.2.7 root	78
6.17 s_curve_torque_map_args Struct Reference	78
6.17.1 Detailed Description	78
6.17.2 Field Documentation	78
6.17.2.1 a	78
6.17.2.2 b	79
6.17.2.3 c	79
6.18 state_change_daemon_args Struct Reference	79
6.18.1 Detailed Description	79
6.18.2 Field Documentation	79
6.18.2.1 meta_task_handle	79
6.19 task_management_info Struct Reference	80
6.19.1 Detailed Description	80
6.19.2 Field Documentation	80
6.19.2.1 parent_msg_queue	80

6.19.2.2 task_handle	80
6.20 task_status_block Struct Reference	81
6.20.1 Detailed Description	81
6.20.2 Field Documentation	81
6.20.2.1 task_high_water_mark	81
6.20.2.2 task_report_time	81
6.21 uv_binary_semaphore_info Struct Reference	81
6.21.1 Detailed Description	82
6.21.2 Field Documentation	82
6.21.2.1 handle	82
6.22 uv_CAN_msg Struct Reference	82
6.22.1 Detailed Description	82
6.22.2 Field Documentation	82
6.22.2.1 data	83
6.22.2.2 dlc	83
6.22.2.3 flags	83
6.22.2.4 msg_id	83
6.23 uv_init_struct Struct Reference	83
6.23.1 Detailed Description	84
6.23.2 Field Documentation	84
6.23.2.1 use_default_settings	84
6.24 uv_init_task_args Struct Reference	84
6.24.1 Detailed Description	84
6.24.2 Field Documentation	85
6.24.2.1 init_info_queue	85
6.24.2.2 meta_task_handle	85
6.24.2.3 specific_args	85
6.25 uv_init_task_response Struct Reference	85
6.25.1 Detailed Description	86
6.25.2 Field Documentation	86
6.25.2.1 device	86
6.25.2.2 errmsg	86
6.25.2.3 nchar	86
6.25.2.4 status	86
6.26 uv_internal_params Struct Reference	87
6.26.1 Detailed Description	87
6.26.2 Field Documentation	87
6.26.2.1 e_code	87
6.26.2.2 init_params	87
6.26.2.3 peripheral_status	87
6.26.2.4 vehicle_settings	88
6.27 uv_mutex_info Struct Reference	88

6.27.1 Detailed Description	88
6.27.2 Field Documentation	88
6.27.2.1 handle	88
6.28 uv_os_settings Struct Reference	88
6.28.1 Detailed Description	89
6.28.2 Field Documentation	89
6.28.2.1 max_svc_task_period	89
6.28.2.2 max_task_period	89
6.28.2.3 min_task_period	89
6.28.2.4 svc_task_manager_period	89
6.28.2.5 task_manager_period	90
6.29 uv_scd_response Struct Reference	90
6.29.1 Detailed Description	90
6.29.2 Field Documentation	90
6.29.2.1 meta_id	90
6.29.2.2 response_val	90
6.30 uv_semaphore_info Struct Reference	91
6.30.1 Detailed Description	91
6.30.2 Field Documentation	91
6.30.2.1 handle	91
6.31 uv_task_info Struct Reference	91
6.31.1 Detailed Description	92
6.31.2 Field Documentation	92
6.31.2.1 active_states	92
6.31.2.2 cmd_data	92
6.31.2.3 deletion_delay	92
6.31.2.4 deletion_states	93
6.31.2.5 parent	93
6.31.2.6 stack_size	93
6.31.2.7 suspension_states	93
6.31.2.8 task_args	94
6.31.2.9 task_flags	94
6.31.2.10 task_function	94
6.31.2.11 task_handle	95
6.31.2.12 task_id	95
6.31.2.13 task_name	95
6.31.2.14 task_period	95
6.31.2.15 task_priority	96
6.31.2.16 task_rx_mailbox	96
6.31.2.17 task_state	96
6.31.2.18 tmi	96
6.32 uv_task_msg_t Struct Reference	97

6.32.1 Detailed Description	97
6.32.2 Field Documentation	97
6.32.2.1 intended_recipient	97
6.32.2.2 message_size	97
6.32.2.3 message_type	98
6.32.2.4 msg_contents	98
6.32.2.5 sender	98
6.32.2.6 time_sent	98
6.33 uv_vehicle_settings Struct Reference	98
6.33.1 Detailed Description	99
6.33.2 Field Documentation	99
6.33.2.1 bms_settings	99
6.33.2.2 daq_settings	99
6.33.2.3 driving_loop_settings	99
6.33.2.4 imd_settings	99
6.33.2.5 is_default	100
6.33.2.6 mc_settings	100
6.33.2.7 os_settings	100
6.33.2.8 pdu_settings	100
6.34 veh_gen_info Struct Reference	100
6.34.1 Detailed Description	100
7 File Documentation	101
7.1 Core/Inc/adc.h File Reference	101
7.1.1 Detailed Description	102
7.1.2 Macro Definition Documentation	102
7.1.2.1 ADC1_BUF_LEN	102
7.1.2.2 ADC1_CHNL_CNT	102
7.1.2.3 ADC1_MAX_VOLT	102
7.1.2.4 ADC1_MIN_VOLT	102
7.1.2.5 ADC1_SAMPLES	103
7.1.2.6 ADC2_BUF_LEN	103
7.1.2.7 ADC2_CHNL_CNT	103
7.1.2.8 ADC2_MAX_VOLT	103
7.1.2.9 ADC2_MIN_VOLT	103
7.1.2.10 ADC2_SAMPLES	103
7.1.3 Function Documentation	104
7.1.3.1 MX_ADC1_Init()	104
7.1.3.2 MX_ADC2_Init()	104
7.1.4 Variable Documentation	104
7.1.4.1 hadc1	105
7.1.4.2 hadc2	105

7.2 Core/Inc/bms.h File Reference	105
7.2.1 Macro Definition Documentation	105
7.2.1.1 DEFAULT_BMS_CAN_TIMEOUT	106
7.2.2 Typedef Documentation	106
7.2.2.1 bms_settings_t	106
7.2.3 Function Documentation	106
7.2.3.1 BMS_Init()	106
7.3 Core/Inc/can.h File Reference	106
7.3.1 Detailed Description	107
7.3.2 Macro Definition Documentation	107
7.3.2.1 CAN_RX_DAEMON_NAME	107
7.3.2.2 CAN_TX_DAEMON_NAME	107
7.3.3 Typedef Documentation	108
7.3.3.1 uv_CAN_msg	108
7.3.3.2 uv_status	108
7.3.4 Function Documentation	108
7.3.4.1 CANbusTxSvcDaemon()	108
7.3.4.2 HAL_CAN_RxFifo0MsgPendingCallback()	108
7.3.4.3 HAL_CAN_RxFifo1MsgPendingCallback()	109
7.3.4.4 MX_CAN2_Init()	109
7.3.4.5 uvSendCanMSG()	109
7.3.5 Variable Documentation	109
7.3.5.1 hcan2	109
7.4 Core/Inc/constants.h File Reference	110
7.4.1 Enumeration Type Documentation	110
7.4.1.1 CAN_IDs	110
7.4.2 Variable Documentation	110
7.4.2.1 RxData	110
7.4.2.2 RxHeader	111
7.4.2.3 TxData	111
7.4.2.4 TxHeader	111
7.4.2.5 TxMailbox	111
7.5 Core/Inc/daq.h File Reference	112
7.5.1 Macro Definition Documentation	113
7.5.1.1 _NUM_LOGGABLE_PARAMS	113
7.5.2 Typedef Documentation	113
7.5.2.1 daq_child_task	113
7.5.2.2 daq_datapoint	113
7.5.2.3 daq_loop_args	113
7.5.2.4 daq_param_list_node	113
7.5.3 Enumeration Type Documentation	113
7.5.3.1 data_type	113

7.5.3.2 loggable_params	114
7.5.4 Function Documentation	115
7.5.4.1 daqMasterTask()	115
7.5.4.2 initDaqTask()	115
7.5.5 Variable Documentation	115
7.5.5.1 param_LUT	115
7.6 Core/Inc/dash.h File Reference	116
7.6.1 Enumeration Type Documentation	116
7.6.1.1 dash_can_ids	116
7.6.2 Function Documentation	116
7.6.2.1 Update_Batt_Temp()	116
7.6.2.2 Update_RPM()	117
7.6.2.3 Update_State_Of_Charge()	117
7.7 Core/Inc/dma.h File Reference	117
7.7.1 Detailed Description	117
7.7.2 Function Documentation	118
7.7.2.1 MX_DMA_Init()	118
7.8 Core/Inc/driving_loop.h File Reference	118
7.8.1 Macro Definition Documentation	119
7.8.1.1 DEFAULT_PERIOD	119
7.8.2 Typedef Documentation	119
7.8.2.1 driving_loop_args	119
7.8.2.2 drivingMode	119
7.8.2.3 drivingModeParams	120
7.8.2.4 exp_torque_map_args	120
7.8.2.5 linear_torque_map_args	120
7.8.2.6 MC_POWER	120
7.8.2.7 MC_RPM	120
7.8.2.8 MC_Torque	120
7.8.2.9 s_curve_torque_map_args	120
7.8.3 Enumeration Type Documentation	120
7.8.3.1 DL_internal_state	120
7.8.3.2 map_mode	121
7.8.4 Function Documentation	121
7.8.4.1 initDrivingLoop()	121
7.8.4.2 StartDrivingLoop()	122
7.9 Core/Inc/errorLUT.h File Reference	123
7.9.1 Macro Definition Documentation	123
7.9.1.1 _NUM_ERRORS_	124
7.10 Core/Inc/FreeRTOSConfig.h File Reference	124
7.10.1 Macro Definition Documentation	125
7.10.1.1 configASSERT	125

7.10.1.2 configCHECK_FOR_STACK_OVERFLOW [1/2]	125
7.10.1.3 configCHECK_FOR_STACK_OVERFLOW [2/2]	125
7.10.1.4 configCPU_CLOCK_HZ	126
7.10.1.5 configENABLE_BACKWARD_COMPATIBILITY	126
7.10.1.6 configENABLE_FPU	126
7.10.1.7 configENABLE_MPU	126
7.10.1.8 configKERNEL_INTERRUPT_PRIORITY	126
7.10.1.9 configLIBRARY_LOWEST_INTERRUPT_PRIORITY	126
7.10.1.10 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY	127
7.10.1.11 configMAX_CO_ROUTINE_PRIORITIES	127
7.10.1.12 configMAX_PRIORITIES	127
7.10.1.13 configMAX_SYSCALL_INTERRUPT_PRIORITY	127
7.10.1.14 configMAX_TASK_NAME_LEN	127
7.10.1.15 configMESSAGE_BUFFER_LENGTH_TYPE	127
7.10.1.16 configMINIMAL_STACK_SIZE	128
7.10.1.17 configPRIO_BITS	128
7.10.1.18 configQUEUE_REGISTRY_SIZE	128
7.10.1.19 configRECORD_STACK_HIGH_ADDRESS	128
7.10.1.20 configSUPPORT_DYNAMIC_ALLOCATION	128
7.10.1.21 configSUPPORT_STATIC_ALLOCATION	128
7.10.1.22 configTICK_RATE_HZ	129
7.10.1.23 configTIMER_QUEUE_LENGTH	129
7.10.1.24 configTIMER_TASK_PRIORITY	129
7.10.1.25 configTIMER_TASK_STACK_DEPTH	129
7.10.1.26 configTOTAL_HEAP_SIZE	129
7.10.1.27 configUSE_16_BIT_TICKS	129
7.10.1.28 configUSE_APPLICATION_TASK_TAG	130
7.10.1.29 configUSE_CO_ROUTINES	130
7.10.1.30 configUSE_COUNTING_SEMAPHORES	130
7.10.1.31 configUSE_IDLE_HOOK	130
7.10.1.32 configUSE_MALLOC_FAILED_HOOK [1/2]	130
7.10.1.33 configUSE_MALLOC_FAILED_HOOK [2/2]	130
7.10.1.34 configUSE_MUTEXES	131
7.10.1.35 configUSE_PORT_OPTIMISED_TASK_SELECTION	131
7.10.1.36 configUSE_PREEMPTION	131
7.10.1.37 configUSE_TICK_HOOK	131
7.10.1.38 configUSE_TIMERS	131
7.10.1.39 INCLUDE_eTaskGetState	131
7.10.1.40 INCLUDE_pcTaskGetTaskName	132
7.10.1.41 INCLUDE_uxTaskGetStackHighWaterMark	132
7.10.1.42 INCLUDE_uxTaskGetStackHighWaterMark2	132
7.10.1.43 INCLUDE_uxTaskPriorityGet	132

7.10.1.44 INCLUDE_vTaskCleanUpResources	132
7.10.1.45 INCLUDE_vTaskDelay	132
7.10.1.46 INCLUDE_vTaskDelayUntil	133
7.10.1.47 INCLUDE_vTaskDelete	133
7.10.1.48 INCLUDE_vTaskPrioritySet	133
7.10.1.49 INCLUDE_vTaskSuspend	133
7.10.1.50 INCLUDE_xEventGroupSetBitFromISR	133
7.10.1.51 INCLUDE_xQueueGetMutexHolder	133
7.10.1.52 INCLUDE_xSemaphoreGetMutexHolder	134
7.10.1.53 INCLUDE_xTaskAbortDelay	134
7.10.1.54 INCLUDE_xTaskDelayUntil	134
7.10.1.55 INCLUDE_xTaskGetCurrentTaskHandle	134
7.10.1.56 INCLUDE_xTaskGetHandle	134
7.10.1.57 INCLUDE_xTaskGetSchedulerState	134
7.10.1.58 INCLUDE_xTimerPendFunctionCall	135
7.10.1.59 vPortSVCHandler	135
7.10.1.60 xPortPendSVHandler	135
7.10.1.61 xPortSysTickHandler	135
7.11 Core/Inc/gpio.h File Reference	135
7.11.1 Detailed Description	136
7.11.2 Function Documentation	136
7.11.2.1 MX_GPIO_Init()	136
7.12 Core/Inc/imd.h File Reference	136
7.12.1 Enumeration Type Documentation	137
7.12.1.1 imd_error_flags	137
7.12.1.2 imd_high_resolution_measurements	137
7.12.1.3 imd_manufacturer_requests	138
7.12.1.4 imd_status_bits	138
7.12.1.5 imd_status_requests	139
7.12.2 Function Documentation	139
7.12.2.1 IMD_Check_Battery_Voltage()	139
7.12.2.2 IMD_Check_Error_Flags()	140
7.12.2.3 IMD_Check_Isolation_Capacitances()	140
7.12.2.4 IMD_Check_Isolation_Resistances()	140
7.12.2.5 IMD_Check_Isolation_State()	140
7.12.2.6 IMD_Check_Max_Battery_Working_Voltage()	141
7.12.2.7 IMD_Check_Part_Name()	141
7.12.2.8 IMD_Check_Safety_Touch_Current()	141
7.12.2.9 IMD_Check_Safety_Touch_Energy()	141
7.12.2.10 IMD_Check_Serial_Number()	142
7.12.2.11 IMD_Check_Status_Bits()	142
7.12.2.12 IMD_Check_Temperature()	142

7.12.2.13	IMD_Check_Uptime()	142
7.12.2.14	IMD_Check_Version()	143
7.12.2.15	IMD_Check_Voltages_Vp_and_Vn()	143
7.12.2.16	IMD_Parse_Message()	143
7.12.2.17	IMD_Request_Status()	143
7.12.2.18	IMD_Startup()	144
7.12.2.19	initIMD()	144
7.13	Core/Inc/main.h File Reference	144
7.13.1	Detailed Description	145
7.13.2	Macro Definition Documentation	145
7.13.2.1	Blue_LED_GPIO_Port	145
7.13.2.2	Blue_LED_Pin	145
7.13.2.3	Orange_LED_GPIO_Port	145
7.13.2.4	Orange_LED_Pin	145
7.13.2.5	Red_LED_GPIO_Port	146
7.13.2.6	Red_LED_Pin	146
7.13.2.7	Start_Button_Input_EXTI_IRQn	146
7.13.2.8	Start_Button_Input_GPIO_Port	146
7.13.2.9	Start_Button_Input_Pin	146
7.13.3	Function Documentation	146
7.13.3.1	Error_Handler()	146
7.14	Core/Inc/motor_controller.h File Reference	147
7.14.1	Macro Definition Documentation	148
7.14.1.1	DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT	148
7.14.2	Typedef Documentation	149
7.14.2.1	motor_controller_settings	149
7.14.3	Enumeration Type Documentation	149
7.14.3.1	motor_controller_current_parameters	149
7.14.3.2	motor_controller_io	149
7.14.3.3	motor_controller_limp_mode	149
7.14.3.4	motor_controller_measurements	150
7.14.3.5	motor_controller_motor_constants	150
7.14.3.6	motor_controller_PI_values	150
7.14.3.7	motor_controller_repeating_time	151
7.14.3.8	motor_controller_speed_parameters	151
7.14.3.9	motor_controller_startup	151
7.14.3.10	motor_controller_status_information_errors_warnings	152
7.14.3.11	motor_controller_temperatures	153
7.14.4	Function Documentation	153
7.14.4.1	MC_Check_Error_Warning()	153
7.14.4.2	MC_Check_Firmware()	153
7.14.4.3	MC_Check_Serial_Number()	154

7.14.4.4 MC_Parse_Message()	154
7.14.4.5 MC_Request_Data()	154
7.14.4.6 MC_Send_Data()	154
7.14.4.7 MC_Speed_Control()	154
7.14.4.8 MC_Startup()	155
7.14.4.9 MC_Torque_Control()	155
7.15 Core/Inc/odometer.h File Reference	155
7.15.1 Function Documentation	155
7.15.1.1 initOdometer()	155
7.15.1.2 odometerTask()	156
7.16 Core/Inc/oled.h File Reference	156
7.16.1 Function Documentation	156
7.16.1.1 amogus()	156
7.16.1.2 oled_config()	157
7.16.1.3 oled_Write()	157
7.16.1.4 oled_Write_Cmd()	157
7.16.1.5 oled_Write_Data()	157
7.16.1.6 refresh_OLED()	157
7.16.1.7 wait()	157
7.17 Core/Inc/pdu.h File Reference	157
7.17.1 Enumeration Type Documentation	158
7.17.1.1 pdu_messages_20A	158
7.17.1.2 pdu_messages_5A	159
7.17.2 Function Documentation	159
7.17.2.1 initPDU()	159
7.17.2.2 PDU_disable_brake_light()	159
7.17.2.3 PDU_disable_coolant_pump()	160
7.17.2.4 PDU_disable_cooling_fans()	160
7.17.2.5 PDU_disable_motor_controller()	160
7.17.2.6 PDU_disable_shutdown_circuit()	160
7.17.2.7 PDU_enable_brake_light()	160
7.17.2.8 PDU_enable_coolant_pump()	161
7.17.2.9 PDU_enable_cooling_fans()	161
7.17.2.10 PDU_enable_motor_controller()	161
7.17.2.11 PDU_enable_shutdown_circuit()	161
7.17.2.12 PDU_speaker_chirp()	161
7.18 Core/Inc/rb_tree.h File Reference	162
7.18.1 Macro Definition Documentation	163
7.18.1.1 BLACK	163
7.18.1.2 RB_APPLY	163
7.18.1.3 RB_DUP	163
7.18.1.4 RB_FIRST	163

7.18.1.5 RB_IEMPTY	163
7.18.1.6 RB_MIN	164
7.18.1.7 RB_MINIMAL	164
7.18.1.8 RB_NIL	164
7.18.1.9 RB_ROOT	164
7.18.1.10 RED	164
7.18.2 Typedef Documentation	164
7.18.2.1 rbnode	164
7.18.3 Enumeration Type Documentation	164
7.18.3.1 rbtraversal	164
7.18.4 Function Documentation	165
7.18.4.1 rbApplyNode()	165
7.18.4.2 rbCheckBlackHeight()	165
7.18.4.3 rbCheckOrder()	165
7.18.4.4 rbCreate()	166
7.18.4.5 rbDelete()	166
7.18.4.6 rbDestroy()	166
7.18.4.7 rbFind()	166
7.18.4.8 rbInsert()	167
7.18.4.9 rbPrint()	167
7.18.4.10 rbSuccessor()	167
7.19 Core/Inc/spi.h File Reference	167
7.19.1 Detailed Description	168
7.19.2 Function Documentation	168
7.19.2.1 MX_SPI1_Init()	168
7.19.3 Variable Documentation	168
7.19.3.1 hspi1	168
7.20 Core/Inc/stm32f4xx_hal_conf.h File Reference	169
7.20.1 Detailed Description	171
7.20.2 Macro Definition Documentation	171
7.20.2.1 assert_param	172
7.20.2.2 DATA_CACHE_ENABLE	172
7.20.2.3 DP83848_PHY_ADDRESS	172
7.20.2.4 ETH_RX_BUF_SIZE	172
7.20.2.5 ETH_RXBUFNB	172
7.20.2.6 ETH_TX_BUF_SIZE	172
7.20.2.7 ETH_TXBUFNB	173
7.20.2.8 EXTERNAL_CLOCK_VALUE	173
7.20.2.9 HAL_ADC_MODULE_ENABLED	173
7.20.2.10 HAL_CAN_MODULE_ENABLED	173
7.20.2.11 HAL_CORTEX_MODULE_ENABLED	173
7.20.2.12 HAL_DMA_MODULE_ENABLED	173

7.20.2.13 HAL_EXTI_MODULE_ENABLED	174
7.20.2.14 HAL_FLASH_MODULE_ENABLED	174
7.20.2.15 HAL_GPIO_MODULE_ENABLED	174
7.20.2.16 HAL_MODULE_ENABLED	174
7.20.2.17 HAL_PWR_MODULE_ENABLED	174
7.20.2.18 HAL_RCC_MODULE_ENABLED	174
7.20.2.19 HAL_SPI_MODULE_ENABLED	175
7.20.2.20 HAL_TIM_MODULE_ENABLED	175
7.20.2.21 HSE_STARTUP_TIMEOUT	175
7.20.2.22 HSE_VALUE	175
7.20.2.23 HSI_VALUE	175
7.20.2.24 INSTRUCTION_CACHE_ENABLE	176
7.20.2.25 LSE_STARTUP_TIMEOUT	176
7.20.2.26 LSE_VALUE	176
7.20.2.27 LSI_VALUE	176
7.20.2.28 MAC_ADDR0	176
7.20.2.29 MAC_ADDR1	177
7.20.2.30 MAC_ADDR2	177
7.20.2.31 MAC_ADDR3	177
7.20.2.32 MAC_ADDR4	177
7.20.2.33 MAC_ADDR5	177
7.20.2.34 PHY_AUTONEGO_COMPLETE	177
7.20.2.35 PHY_AUTONEGOTIATION	178
7.20.2.36 PHY_BCR	178
7.20.2.37 PHY_BSR	178
7.20.2.38 PHY_CONFIG_DELAY	178
7.20.2.39 PHY_DUPLEX_STATUS	178
7.20.2.40 PHY_FULLDUPLEX_100M	179
7.20.2.41 PHY_FULLDUPLEX_10M	179
7.20.2.42 PHY_HALFDUPLEX_100M	179
7.20.2.43 PHY_HALFDUPLEX_10M	179
7.20.2.44 PHY_ISOLATE	179
7.20.2.45 PHY_JABBER_DETECTION	180
7.20.2.46 PHY_LINKED_STATUS	180
7.20.2.47 PHY_LOOPBACK	180
7.20.2.48 PHY_POWERDOWN	180
7.20.2.49 PHY_READ_TO	180
7.20.2.50 PHY_RESET	181
7.20.2.51 PHY_RESET_DELAY	181
7.20.2.52 PHY_RESTART_AUTONEGOTIATION	181
7.20.2.53 PHY_SPEED_STATUS	181
7.20.2.54 PHY_SR	181

7.20.2.55 PHY_WRITE_TO	182
7.20.2.56 PREFETCH_ENABLE	182
7.20.2.57 TICK_INT_PRIORITY	182
7.20.2.58 USE_HAL_ADC_REGISTER_CALLBACKS	182
7.20.2.59 USE_HAL_CAN_REGISTER_CALLBACKS	182
7.20.2.60 USE_HAL_CEC_REGISTER_CALLBACKS	182
7.20.2.61 USE_HAL_CRYP_REGISTER_CALLBACKS	183
7.20.2.62 USE_HAL_DAC_REGISTER_CALLBACKS	183
7.20.2.63 USE_HAL_DCMI_REGISTER_CALLBACKS	183
7.20.2.64 USE_HAL_DFSDM_REGISTER_CALLBACKS	183
7.20.2.65 USE_HAL_DMA2D_REGISTER_CALLBACKS	183
7.20.2.66 USE_HAL_DSI_REGISTER_CALLBACKS	183
7.20.2.67 USE_HAL_ETH_REGISTER_CALLBACKS	184
7.20.2.68 USE_HAL_FMPI2C_REGISTER_CALLBACKS	184
7.20.2.69 USE_HAL_FMPSPMBUS_REGISTER_CALLBACKS	184
7.20.2.70 USE_HAL_HASH_REGISTER_CALLBACKS	184
7.20.2.71 USE_HAL_HCD_REGISTER_CALLBACKS	184
7.20.2.72 USE_HAL_I2C_REGISTER_CALLBACKS	184
7.20.2.73 USE_HAL_I2S_REGISTER_CALLBACKS	185
7.20.2.74 USE_HAL_IRDA_REGISTER_CALLBACKS	185
7.20.2.75 USE_HAL_LPTIM_REGISTER_CALLBACKS	185
7.20.2.76 USE_HAL_LTDC_REGISTER_CALLBACKS	185
7.20.2.77 USE_HAL_MMC_REGISTER_CALLBACKS	185
7.20.2.78 USE_HAL_NAND_REGISTER_CALLBACKS	185
7.20.2.79 USE_HAL_NOR_REGISTER_CALLBACKS	186
7.20.2.80 USE_HAL_PCCARD_REGISTER_CALLBACKS	186
7.20.2.81 USE_HAL_PCD_REGISTER_CALLBACKS	186
7.20.2.82 USE_HAL_QSPI_REGISTER_CALLBACKS	186
7.20.2.83 USE_HAL_RNG_REGISTER_CALLBACKS	186
7.20.2.84 USE_HAL_RTC_REGISTER_CALLBACKS	186
7.20.2.85 USE_HAL_SAI_REGISTER_CALLBACKS	187
7.20.2.86 USE_HAL_SD_REGISTER_CALLBACKS	187
7.20.2.87 USE_HAL_SDRAM_REGISTER_CALLBACKS	187
7.20.2.88 USE_HAL_SMARTCARD_REGISTER_CALLBACKS	187
7.20.2.89 USE_HAL_SMBUS_REGISTER_CALLBACKS	187
7.20.2.90 USE_HAL_SPDIFRX_REGISTER_CALLBACKS	187
7.20.2.91 USE_HAL_SPI_REGISTER_CALLBACKS	188
7.20.2.92 USE_HAL_SRAM_REGISTER_CALLBACKS	188
7.20.2.93 USE_HAL_TIM_REGISTER_CALLBACKS	188
7.20.2.94 USE_HAL_UART_REGISTER_CALLBACKS	188
7.20.2.95 USE_HAL_USART_REGISTER_CALLBACKS	188
7.20.2.96 USE_HAL_WWDG_REGISTER_CALLBACKS	188

7.20.2.97 USE_RTOS	189
7.20.2.98 USE_SPI_CRC	189
7.20.2.99 VDD_VALUE	189
7.21 Core/Inc/stm32f4xx_it.h File Reference	189
7.21.1 Detailed Description	190
7.21.2 Function Documentation	190
7.21.2.1 BusFault_Handler()	190
7.21.2.2 CAN2_RX0_IRQHandler()	190
7.21.2.3 CAN2_RX1_IRQHandler()	190
7.21.2.4 CAN2_TX_IRQHandler()	191
7.21.2.5 DebugMon_Handler()	191
7.21.2.6 DMA2_Stream0_IRQHandler()	191
7.21.2.7 EXTI0_IRQHandler()	191
7.21.2.8 HardFault_Handler()	192
7.21.2.9 MemManage_Handler()	192
7.21.2.10 NMI_Handler()	192
7.21.2.11 TIM1_UP_TIM10_IRQHandler()	192
7.21.2.12 UsageFault_Handler()	192
7.22 Core/Inc/temp_monitoring.h File Reference	193
7.22.1 Function Documentation	193
7.22.1.1 initTempMonitor()	193
7.22.1.2 tempMonitorTask()	193
7.23 Core/Inc/tim.h File Reference	194
7.23.1 Detailed Description	194
7.23.2 Function Documentation	194
7.23.2.1 MX_TIM3_Init()	194
7.23.3 Variable Documentation	194
7.23.3.1 htim3	195
7.24 Core/Inc/uvfr_global_config.h File Reference	195
7.24.1 Macro Definition Documentation	195
7.24.1.1 ECUMASTER_PMU	195
7.24.1.2 STM32_F407	195
7.24.1.3 STM32_H7xx	195
7.24.1.4 USE_OS_MEM_MGMT	196
7.24.1.5 UV19_PDU	196
7.24.1.6 UV_MALLOC_LIMIT	196
7.25 Core/Inc/uvfr_settings.h File Reference	196
7.25.1 Macro Definition Documentation	197
7.25.1.1 ENABLE_FLASH_SETTINGS	197
7.25.2 Typedef Documentation	197
7.25.2.1 uv_vehicle_settings	197
7.25.2.2 veh_gen_info	197

7.25.3 Function Documentation	197
7.25.3.1 nukeSettings()	197
7.25.3.2 uvSettingsInit()	198
7.25.4 Variable Documentation	198
7.25.4.1 current_vehicle_settings	198
7.26 Core/Inc/uvfr_state_engine.h File Reference	198
7.26.1 Macro Definition Documentation	201
7.26.1.1 _LONGEST_SC_TIME	201
7.26.1.2 _SC_DAEMON_PERIOD	201
7.26.1.3 _UV_DEFAULT_TASK_INSTANCES	201
7.26.1.4 _UV_DEFAULT_TASK_PERIOD	202
7.26.1.5 _UV_DEFAULT_TASK_STACK_SIZE	202
7.26.1.6 _UV_MIN_TASK_PERIOD	202
7.26.1.7 SVC_TASK_MAX_CHECKIN_PERIOD	202
7.26.2 Typedef Documentation	202
7.26.2.1 uv_status	202
7.26.2.2 uv_task_id	202
7.26.2.3 uv_timespan_ms	203
7.26.3 Function Documentation	203
7.26.3.1 getSVCTaskID()	203
7.26.3.2 updateRunningTasks()	203
7.26.3.3 uvGetTaskById()	203
7.26.3.4 uvRegisterTask()	203
7.27 Core/Inc/uvfr_utils.h File Reference	204
7.27.1 Detailed Description	206
7.27.2 Macro Definition Documentation	207
7.27.2.1 INIT_CHECK_PERIOD	207
7.27.2.2 MAX_INIT_TIME	207
7.27.2.3 USE_OLED_DEBUG	207
7.27.2.4 UV_CAN1	207
7.27.2.5 UV_CAN2	207
7.27.2.6 UV_CAN_CHANNEL_MASK	208
7.27.2.7 UV_CAN_DYNAMIC_MEM	208
7.27.2.8 UV_CAN_EXTENDED_ID	208
7.27.3 Typedef Documentation	208
7.27.3.1 access_control_info	208
7.27.3.2 access_control_type	208
7.27.3.3 bool	208
7.27.3.4 p_status	209
7.27.3.5 uv_CAN_msg	209
7.27.3.6 uv_ext_device_id	209
7.27.3.7 uv_init_struct	209

7.27.3.8 uv_init_task_args	209
7.27.3.9 uv_init_task_response	209
7.27.3.10 uv_internal_params	210
7.27.3.11 uv_msg_type	210
7.27.3.12 uv_status	210
7.27.3.13 uv_task_cmd	210
7.27.3.14 uv_task_id	210
7.27.3.15 uv_task_msg	210
7.27.3.16 uv_timespan_ms	211
7.27.4 Enumeration Type Documentation	211
7.27.4.1 access_control_t	211
7.27.4.2 uv_driving_mode_t	211
7.27.4.3 uv_external_device	211
7.27.4.4 uv_msg_type_t	212
7.27.4.5 uv_status_t	212
7.27.5 Function Documentation	213
7.27.5.1 __uvInitPanic()	213
7.27.5.2 uvInit()	213
7.27.5.3 uvIsPTRValid()	215
7.27.6 Variable Documentation	215
7.27.6.1 global_context	216
7.28 Core/Inc/uvfr_vehicle_commands.h File Reference	216
7.28.1 Macro Definition Documentation	216
7.28.1.1 uvHonkHorn	216
7.28.1.2 uvOpenSDC [1/2]	216
7.28.1.3 uvOpenSDC [2/2]	217
7.28.1.4 uvStartCoolantPump	217
7.28.1.5 uvStartFans	217
7.28.1.6 uvStopCoolantPump	217
7.28.1.7 uvStopFans	217
7.29 Core/Src/adc.c File Reference	217
7.29.1 Detailed Description	218
7.29.2 Function Documentation	218
7.29.2.1 HAL_ADC_MspDeInit()	218
7.29.2.2 HAL_ADC_MspInit()	219
7.29.2.3 MX_ADC1_Init()	219
7.29.2.4 MX_ADC2_Init()	219
7.29.3 Variable Documentation	220
7.29.3.1 hadc1	220
7.29.3.2 hadc2	220
7.29.3.3 hdma_adc1	220
7.30 Core/Src/bms.c File Reference	220

7.30.1 Function Documentation	221
7.30.1.1 BMS_Init()	221
7.31 Core/Src/can.c File Reference	221
7.31.1 Detailed Description	222
7.31.2 Macro Definition Documentation	222
7.31.2.1 HAL_CAN_ERROR_INVALID_CALLBACK	222
7.31.3 Function Documentation	222
7.31.3.1 CANbusRxSVCDaemon()	222
7.31.3.2 CANbusTxSvcDaemon()	223
7.31.3.3 HAL_CAN_MspDeInit()	223
7.31.3.4 HAL_CAN_MspInit()	223
7.31.3.5 HAL_CAN_RxFifo0MsgPendingCallback()	223
7.31.3.6 HAL_CAN_RxFifo1MsgPendingCallback()	224
7.31.3.7 handleCANbusError()	224
7.31.3.8 MX_CAN2_Init()	224
7.31.3.9 uvSendCanMSG()	224
7.31.4 Variable Documentation	225
7.31.4.1 hcan2	225
7.31.4.2 Tx_msg_queue	225
7.32 Core/Src/constants.c File Reference	225
7.32.1 Variable Documentation	225
7.32.1.1 RxData	226
7.32.1.2 RxHeader	226
7.32.1.3 TxData	226
7.32.1.4 TxHeader	226
7.32.1.5 TxMailbox	227
7.33 Core/Src/daq.c File Reference	227
7.33.1 Macro Definition Documentation	227
7.33.1.1 _SRC_UVFR_DAQ	227
7.33.2 Function Documentation	228
7.33.2.1 daqMasterTask()	228
7.33.2.2 daqSubTask()	228
7.33.2.3 deleteDaqSubTask()	228
7.33.2.4 deleteParamList()	228
7.33.2.5 initDaqTask()	229
7.33.2.6 startDaqSubTasks()	229
7.33.2.7 stopDaqSubTasks()	229
7.33.3 Variable Documentation	229
7.33.3.1 param_LUT	229
7.34 Core/Src/dash.c File Reference	229
7.34.1 Function Documentation	230
7.34.1.1 Update_Batt_Temp()	230

7.34.1.2 Update_RPM()	230
7.34.1.3 Update_State_Of_Charge()	230
7.35 Core/Src/dma.c File Reference	230
7.35.1 Detailed Description	231
7.35.2 Function Documentation	231
7.35.2.1 MX_DMA_Init()	231
7.36 Core/Src/driving_loop.c File Reference	231
7.36.1 Detailed Description	232
7.36.2 Function Documentation	232
7.36.2.1 initDrivingLoop()	232
7.36.2.2 StartDrivingLoop()	232
7.36.3 Variable Documentation	233
7.36.3.1 adc1_APPS1	233
7.36.3.2 adc1_APPS2	234
7.36.3.3 adc1_BPS1	234
7.36.3.4 adc1_BPS2	234
7.37 Core/Src/freertos.c File Reference	234
7.37.1 Function Documentation	235
7.37.1.1 MX_FREERTOS_Init()	235
7.37.1.2 StartDefaultTask()	235
7.37.1.3 vApplicationGetIdleTaskMemory()	236
7.37.1.4 vApplicationGetTimerTaskMemory()	236
7.37.1.5 vApplicationIdleHook()	236
7.37.1.6 vApplicationMallocFailedHook()	236
7.37.1.7 vApplicationStackOverflowHook()	236
7.37.1.8 vApplicationTickHook()	237
7.37.2 Variable Documentation	237
7.37.2.1 defaultTaskHandle	237
7.37.2.2 init_settings	237
7.37.2.3 init_task_handle	237
7.37.2.4 xIdleStack	238
7.37.2.5 xIdleTaskTCBBuffer	238
7.37.2.6 xTimerStack	238
7.37.2.7 xTimerTaskTCBBuffer	238
7.38 Core/Src/gpio.c File Reference	238
7.38.1 Detailed Description	239
7.38.2 Function Documentation	239
7.38.2.1 MX_GPIO_Init()	239
7.39 Core/Src/imd.c File Reference	239
7.39.1 Function Documentation	240
7.39.1.1 IMD_Check_Battery_Voltage()	241
7.39.1.2 IMD_Check_Error_Flags()	241

7.39.1.3 IMD_Check_Isolation_Capacitances()	241
7.39.1.4 IMD_Check_Isolation_Resistances()	241
7.39.1.5 IMD_Check_Isolation_State()	242
7.39.1.6 IMD_Check_Max_Battery_Working_Voltage()	242
7.39.1.7 IMD_Check_Part_Name()	242
7.39.1.8 IMD_Check_Safety_Touch_Current()	242
7.39.1.9 IMD_Check_Safety_Touch_Energy()	243
7.39.1.10 IMD_Check_Serial_Number()	243
7.39.1.11 IMD_Check_Status_Bits()	243
7.39.1.12 IMD_Check_Temperature()	243
7.39.1.13 IMD_Check_Uptime()	244
7.39.1.14 IMD_Check_Version()	244
7.39.1.15 IMD_Check_Voltages_Vp_and_Vn()	244
7.39.1.16 IMD_Parse_Message()	244
7.39.1.17 IMD_Request_Status()	245
7.39.1.18 IMD_Startup()	245
7.39.1.19 initIMD()	245
7.39.2 Variable Documentation	245
7.39.2.1 IMD_error_flags_requested	245
7.39.2.2 IMD_Expected_Part_Name	246
7.39.2.3 IMD_Expected_Serial_Number	246
7.39.2.4 IMD_Expected_Version	246
7.39.2.5 IMD_High_Uncertainty	246
7.39.2.6 IMD_Part_Name_0_Set	246
7.39.2.7 IMD_Part_Name_1_Set	247
7.39.2.8 IMD_Part_Name_2_Set	247
7.39.2.9 IMD_Part_Name_3_Set	247
7.39.2.10 IMD_Part_Name_Set	247
7.39.2.11 IMD_Read_Part_Name	247
7.39.2.12 IMD_Read_Serial_Number	248
7.39.2.13 IMD_Read_Version	248
7.39.2.14 IMD_Serial_Number_0_Set	248
7.39.2.15 IMD_Serial_Number_1_Set	248
7.39.2.16 IMD_Serial_Number_2_Set	248
7.39.2.17 IMD_Serial_Number_3_Set	249
7.39.2.18 IMD_Serial_Number_Set	249
7.39.2.19 IMD_status_bits	249
7.39.2.20 IMD_Temperature	249
7.39.2.21 IMD_Version_0_Set	249
7.39.2.22 IMD_Version_1_Set	250
7.39.2.23 IMD_Version_2_Set	250
7.39.2.24 IMD_Version_Set	250

7.40 Core/Src/main.c File Reference	250
7.40.1 Detailed Description	251
7.40.2 Macro Definition Documentation	251
7.40.2.1 DEBUG_CAN_IN_MAIN	251
7.40.3 Function Documentation	252
7.40.3.1 Error_Handler()	252
7.40.3.2 HAL_ADC_ConvCpltCallback()	252
7.40.3.3 HAL_ADC_LevelOutOfWindowCallback()	252
7.40.3.4 HAL_GPIO_EXTI_Callback()	253
7.40.3.5 HAL_TIM_PeriodElapsedCallback()	253
7.40.3.6 main()	253
7.40.3.7 MX_FREERTOS_Init()	254
7.40.3.8 SystemClock_Config()	254
7.40.4 Variable Documentation	254
7.40.4.1 adc1_APPS1	255
7.40.4.2 adc1_APPS2	255
7.40.4.3 adc1_BPS1	255
7.40.4.4 adc1_BPS2	255
7.40.4.5 adc2_CoolantFlow	255
7.40.4.6 adc2_CoolantTemp	256
7.40.4.7 adc_buf1	256
7.40.4.8 adc_buf2	256
7.41 Core/Src/motor_controller.c File Reference	256
7.41.1 Function Documentation	257
7.41.1.1 MC_Check_Error_Warning()	257
7.41.1.2 MC_Check_Firmware()	257
7.41.1.3 MC_Check_Serial_Number()	257
7.41.1.4 MC_Parse_Message()	258
7.41.1.5 MC_Request_Data()	258
7.41.1.6 MC_Send_Data()	258
7.41.1.7 MC_Startup()	258
7.41.1.8 MC_Torque_Control()	259
7.41.1.9 MC_Validate()	259
7.41.2 Variable Documentation	259
7.41.2.1 desired_motor_speed	259
7.41.2.2 max_motor_speed	259
7.41.2.3 MC_Expected_FW_Version	259
7.41.2.4 MC_Expected_Serial_Number	259
7.42 Core/Src/odometer.c File Reference	260
7.42.1 Function Documentation	260
7.42.1.1 initOdometer()	260
7.42.1.2 odometerTask()	260

7.43 Core/Src/oled.c File Reference	261
7.44 Core/Src/pdu.c File Reference	261
7.44.1 Function Documentation	261
7.44.1.1 initPDU()	261
7.44.1.2 PDU_disable_brake_light()	262
7.44.1.3 PDU_disable_coolant_pump()	262
7.44.1.4 PDU_disable_cooling_fans()	262
7.44.1.5 PDU_disable_motor_controller()	262
7.44.1.6 PDU_disable_shutdown_circuit()	262
7.44.1.7 PDU_enable_brake_light()	263
7.44.1.8 PDU_enable_coolant_pump()	263
7.44.1.9 PDU_enable_cooling_fans()	263
7.44.1.10 PDU_enable_motor_controller()	263
7.44.1.11 PDU_enable_shutdown_circuit()	263
7.44.1.12 PDU_speaker_chirp()	264
7.45 Core/Src/rb_tree.c File Reference	264
7.45.1 Function Documentation	264
7.45.1.1 checkBlackHeight()	265
7.45.1.2 checkOrder()	265
7.45.1.3 deleteRepair()	265
7.45.1.4 destroyAllNodes()	265
7.45.1.5 insertRepair()	266
7.45.1.6 print()	266
7.45.1.7 rb_apply()	266
7.45.1.8 rbCheckBlackHeight()	266
7.45.1.9 rbCheckOrder()	267
7.45.1.10 rbCreate()	267
7.45.1.11 rbDelete()	267
7.45.1.12 rbDestroy()	267
7.45.1.13 rbFind()	268
7.45.1.14 rbInsert()	268
7.45.1.15 rbPrint()	268
7.45.1.16 rbSuccessor()	268
7.45.1.17 rotateLeft()	269
7.45.1.18 rotateRight()	269
7.46 Core/Src/spi.c File Reference	269
7.46.1 Detailed Description	270
7.46.2 Function Documentation	270
7.46.2.1 HAL_SPI_MspDeInit()	270
7.46.2.2 HAL_SPI_MspInit()	270
7.46.2.3 MX_SPI1_Init()	270
7.46.3 Variable Documentation	271

7.46.3.1 hspi1	271
7.47 Core/Src/stm32f4xx_hal_msp.c File Reference	271
7.47.1 Detailed Description	271
7.47.2 Function Documentation	271
7.47.2.1 HAL_MspInit()	271
7.48 Core/Src/stm32f4xx_hal_timebase_tim.c File Reference	272
7.48.1 Detailed Description	272
7.48.2 Function Documentation	272
7.48.2.1 HAL_InitTick()	272
7.48.2.2 HAL_ResumeTick()	273
7.48.2.3 HAL_SuspendTick()	273
7.48.3 Variable Documentation	273
7.48.3.1 htim1	274
7.49 Core/Src/stm32f4xx_it.c File Reference	274
7.49.1 Detailed Description	275
7.49.2 Function Documentation	275
7.49.2.1 BusFault_Handler()	275
7.49.2.2 CAN2_RX0_IRQHandler()	275
7.49.2.3 CAN2_RX1_IRQHandler()	275
7.49.2.4 CAN2_TX_IRQHandler()	276
7.49.2.5 DebugMon_Handler()	276
7.49.2.6 DMA2_Stream0_IRQHandler()	276
7.49.2.7 EXTI0_IRQHandler()	276
7.49.2.8 HardFault_Handler()	277
7.49.2.9 MemManage_Handler()	277
7.49.2.10 NMI_Handler()	277
7.49.2.11 TIM1_UP_TIM10_IRQHandler()	277
7.49.2.12 UsageFault_Handler()	277
7.49.3 Variable Documentation	278
7.49.3.1 hcan2	278
7.49.3.2 hdma_adc1	278
7.49.3.3 htim1	278
7.50 Core/Src/syscalls.c File Reference	278
7.50.1 Detailed Description	279
7.50.2 Function Documentation	279
7.50.2.1 __attribute__()	280
7.50.2.2 __io_getchar()	280
7.50.2.3 __io_putchar()	280
7.50.2.4 _close()	280
7.50.2.5 _execve()	280
7.50.2.6 _exit()	281
7.50.2.7 _fork()	281

7.50.2.8 _fstat()	281
7.50.2.9 _getpid()	281
7.50.2.10 _isatty()	281
7.50.2.11 _kill()	282
7.50.2.12 _link()	282
7.50.2.13 _lseek()	282
7.50.2.14 _open()	282
7.50.2.15 _stat()	282
7.50.2.16 _times()	283
7.50.2.17 _unlink()	283
7.50.2.18 _wait()	283
7.50.2.19 initialise_monitor_handles()	283
7.50.3 Variable Documentation	283
7.50.3.1 environ	283
7.51 Core/Src/systemem.c File Reference	284
7.51.1 Detailed Description	284
7.51.2 Function Documentation	284
7.51.2.1 _sbrk()	284
7.51.3 Variable Documentation	285
7.51.3.1 __sbrk_heap_end	285
7.52 Core/Src/system_stm32f4xx.c File Reference	285
7.52.1 Detailed Description	286
7.53 Core/Src/temp_monitoring.c File Reference	286
7.53.1 Function Documentation	286
7.53.1.1 initTempMonitor()	287
7.53.1.2 tempMonitorTask()	287
7.54 Core/Src/tim.c File Reference	287
7.54.1 Detailed Description	288
7.54.2 Function Documentation	288
7.54.2.1 HAL_TIM_Base_MspDeInit()	288
7.54.2.2 HAL_TIM_Base_MspInit()	288
7.54.2.3 MX_TIM3_Init()	288
7.54.3 Variable Documentation	289
7.54.3.1 htim3	289
7.55 Core/Src/uvfr_settings.c File Reference	289
7.55.1 Macro Definition Documentation	289
7.55.1.1 SRC_UVFR_SETTINGS_C_	290
7.55.2 Function Documentation	290
7.55.2.1 nukeSettings()	290
7.55.2.2 setupDefaultSettings()	290
7.55.2.3 uvSettingsInit()	290
7.55.2.4 uvSettingsProgrammerTask()	291

7.55.3 Variable Documentation	291
7.55.3.1 current_vehicle_settings	291
7.56 Core/Src/uvfr_state_engine.c File Reference	291
7.56.1 Detailed Description	293
7.56.2 Macro Definition Documentation	293
7.56.2.1 UVFR_STATE_MACHINE_IMPLIMENTATION	293
7.57 Core/Src/uvfr_utils.c File Reference	294
7.57.1 Macro Definition Documentation	294
7.57.1.1 UV_UTILS_SRC_IMPLIMENTATION	294
7.57.2 Function Documentation	295
7.57.2.1 __uvFreeCriticalSection()	295
7.57.2.2 __uvFreeOS()	295
7.57.2.3 __uvInitPanic()	295
7.57.2.4 __uvMallocCriticalSection()	296
7.57.2.5 __uvMallocOS()	296
7.57.2.6 setup_extern_devices()	296
7.57.2.7 uvInit()	297
7.57.2.8 uvIsPTRValid()	299
7.57.2.9 uvUtilsReset()	299
7.57.3 Variable Documentation	299
7.57.3.1 init_task_handle	299
7.57.3.2 TxData	299
7.58 Core/Src/uvfr_vehicle_commands.c File Reference	299

Chapter 1

Deprecated List

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.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

State Engine	9
State Engine API	14
State Engine Internals	28
UVFR Utilities	40
Utility Macros	41
UVFR Vehicle Commands	47
CMSIS	48
Stm32f4xx_system	49
STM32F4xx_System_Private_Includes	50
STM32F4xx_System_Private_TypesDefinitions	51
STM32F4xx_System_Private_Defines	52
STM32F4xx_System_Private_Macros	53
STM32F4xx_System_Private_Variables	54
STM32F4xx_System_Private_FunctionPrototypes	55
STM32F4xx_System_Private_Functions	56

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

access_control_info	57
bms_settings_t	58
daq_child_task	58
daq_datapoint	
This struct holds info of what needs to be logged	60
daq_loop_args	61
daq_param_list_node	62
driving_loop_args	63
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	68
drivingMode	
This is where the driving mode and the drivingModeParams are at	68
drivingModeParams	
This struct is designed to hold information about each drivingmode's map params	70
exp_torque_map_args	
Struct to hold parameters used in an exponential torque map	71
linear_torque_map_args	71
motor_controllor_settings	72
p_status	73
rbnode	
Node of a Red-Black binary search tree	74
rbtree	
Struct representing a binary search tree	76
s_curve_torque_map_args	
Struct for s-curve parameters for torque	78
state_change_daemon_args	79
task_management_info	
Struct to contain data about a parent task	80
task_status_block	
Information about the task	81
uv_binary_semaphore_info	81
uv_CAN_msg	
Representative of a CAN message	82

uv_init_struct	83
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	84
uv_init_task_response	
Struct representing the response of one of the initialization tasks	85
uv_internal_params	
Data used by the uvfr_utils library to do what it needs to do :)	87
uv_mutex_info	88
uv_os_settings	
Settings that dictate state engine behavior	88
uv_scd_response	90
uv_semaphore_info	91
uv_task_info	
This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine	91
uv_task_msg_t	
Struct containing a message between two tasks	97
uv_vehicle_settings	98
veh_gen_info	100

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	101
Core/Inc/ bms.h	105
Core/Inc/ can.h	
This file contains all the function prototypes for the can.c file	106
Core/Inc/ constants.h	110
Core/Inc/ daq.h	112
Core/Inc/ dash.h	116
Core/Inc/ dma.h	
This file contains all the function prototypes for the dma.c file	117
Core/Inc/ driving_loop.h	118
Core/Inc/ errorLUT.h	123
Core/Inc/ FreeRTOSConfig.h	124
Core/Inc/ gpio.h	
This file contains all the function prototypes for the gpio.c file	135
Core/Inc/ imd.h	136
Core/Inc/ main.h	
: Header for main.c file. This file contains the common defines of the application	144
Core/Inc/ motor_controller.h	147
Core/Inc/ odometer.h	155
Core/Inc/ oled.h	156
Core/Inc/ pdu.h	157
Core/Inc/ rb_tree.h	162
Core/Inc/ spi.h	
This file contains all the function prototypes for the spi.c file	167
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	169
Core/Inc/ stm32f4xx_it.h	
This file contains the headers of the interrupt handlers	189
Core/Inc/ temp_monitoring.h	193
Core/Inc/ tim.h	
This file contains all the function prototypes for the tim.c file	194
Core/Inc/ uvfr_global_config.h	195
Core/Inc/ uvfr_settings.h	196

Core/Inc/ uvfr_state_engine.h	198
Core/Inc/ uvfr_utils.h	204
Core/Inc/ uvfr_vehicle_commands.h	216
Core/Src/ adc.c	
This file provides code for the configuration of the ADC instances	217
Core/Src/ bms.c	220
Core/Src/ can.c	
This file provides code for the configuration of the CAN instances	221
Core/Src/ constants.c	225
Core/Src/ daq.c	227
Core/Src/ dash.c	229
Core/Src/ dma.c	
This file provides code for the configuration of all the requested memory to memory DMA transfers	230
Core/Src/ driving_loop.c	
File containing the meat and potatoes driving loop thread, and all supporting functions	231
Core/Src/ freertos.c	234
Core/Src/ gpio.c	
This file provides code for the configuration of all used GPIO pins	238
Core/Src/ imd.c	239
Core/Src/ main.c	
: Main program body	250
Core/Src/ motor_controller.c	256
Core/Src/ odometer.c	260
Core/Src/ oled.c	261
Core/Src/ pdu.c	261
Core/Src/ rb_tree.c	264
Core/Src/ spi.c	
This file provides code for the configuration of the SPI instances	269
Core/Src/ stm32f4xx_hal_msp.c	
This file provides code for the MSP Initialization and de-Initialization codes	271
Core/Src/ stm32f4xx_hal_timebase_tim.c	
HAL time base based on the hardware TIM	272
Core/Src/ stm32f4xx_it.c	
Interrupt Service Routines	274
Core/Src/ syscalls.c	
STM32CubeIDE Minimal System calls file	278
Core/Src/ systemem.c	
STM32CubeIDE System Memory calls file	284
Core/Src/ system_stm32f4xx.c	
CMSIS Cortex-M4 Device Peripheral Access Layer System Source File	285
Core/Src/ temp_monitoring.c	286
Core/Src/ tim.c	
This file provides code for the configuration of the TIM instances	287
Core/Src/ uvfr_settings.c	289
Core/Src/ uvfr_state_engine.c	
File containing the implementation of the vehicle's state engine and error handling infrastructure	291
Core/Src/ uvfr_utils.c	294
Core/Src/ uvfr_vehicle_commands.c	299

Chapter 5

Module Documentation

5.1 State Engine

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

Modules

- [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

- void [uvSVCTaskManager](#) (void *args)
oversees all of the service tasks, and makes sure that theyre alright

Variables

- static `uv_task_id_next_task_id` = 0
- static `uv_task_info * _task_register` = NULL
- static `uv_task_id_next_svc_task_id` = 0
- static `uv_task_info * _svc_task_register` = NULL
- `TaskHandle_t * scd_handle_ptr`
- static volatile `bool SCD_active` = false
- static `QueueHandle_t state_change_queue` = NULL
- `rbtree * task_name_lut` = NULL
- 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
- `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 necessary 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 20 of file `uvfr_state_engine.c`.

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 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 1284 of file uvfr_state_engine.c.

References `__uvInitPanic()`, `_task_register`, `uv_task_info::active_states`, `CAN_TX_DAEMON_NAME`, `CANbusTx↔SvcDaemon()`, `uv_task_info::task_function`, `task_management_info::task_handle`, `uv_task_info::task_name`, `uv↔CreateServiceTask()`, and `uvStartTask()`.

Referenced by `uvStartStateMachine()`.

5.1.5 Variable Documentation

5.1.5.1 _next_svc_task_id

```
uv_task_id _next_svc_task_id = 0 [static]
```

Definition at line 28 of file uvfr_state_engine.c.

Referenced by `uvCreateServiceTask()`.

5.1.5.2 `_next_task_id`

```
uv_task_id _next_task_id = 0 [static]
```

Definition at line 25 of file `uvfr_state_engine.c`.

Referenced by `_stateChangeDaemon()`, `killEmAll()`, `uvCreateServiceTask()`, `uvCreateTask()`, and `uvValidateManagedTasks()`.

5.1.5.3 `_svc_task_register`

```
uv_task_info* _svc_task_register = NULL [static]
```

Definition at line 29 of file `uvfr_state_engine.c`.

5.1.5.4 `_task_register`

```
uv_task_info* _task_register = NULL [static]
```

Definition at line 26 of file `uvfr_state_engine.c`.

Referenced by `_stateChangeDaemon()`, `_uvValidateSpecificTask()`, `killEmAll()`, `proccessSCDMsg()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvInitStateEngine()`, and `uvSVCTaskManager()`.

5.1.5.5 `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,  
}
```

Definition at line 45 of file `uvfr_state_engine.c`.

Referenced by `setupDefaultSettings()`.

5.1.5.6 `previous_state`

```
enum uv_vehicle_state_t previous_state = UV_BOOT
```

Definition at line 40 of file `uvfr_state_engine.c`.

Referenced by `changeVehicleState()`, and `uvStartStateMachine()`.

5.1.5.7 SCD_active

```
volatile bool SCD_active = false [static]
```

Definition at line 34 of file uvfr_state_engine.c.

Referenced by _stateChangeDaemon().

5.1.5.8 scd_handle_ptr

```
TaskHandle_t* scd_handle_ptr
```

Definition at line 31 of file uvfr_state_engine.c.

5.1.5.9 state_change_queue

```
QueueHandle_t state_change_queue = NULL [static]
```

Definition at line 35 of file uvfr_state_engine.c.

Referenced by _stateChangeDaemon(), killSelf(), and suspendSelf().

5.1.5.10 svc_task_manager

```
uv_task_info* svc_task_manager = NULL
```

Definition at line 43 of file uvfr_state_engine.c.

Referenced by uvInitStateEngine(), and uvStartStateMachine().

5.1.5.11 task_manager

```
uv_task_info* task_manager = NULL
```

Definition at line 42 of file uvfr_state_engine.c.

Referenced by uvInitStateEngine(), and uvStartStateMachine().

5.1.5.12 task_name_lut

```
rbtree* task_name_lut = NULL
```

Definition at line 37 of file uvfr_state_engine.c.

5.1.5.13 vehicle_state

```
enum uv_vehicle_state_t vehicle_state = UV_BOOT
```

Definition at line 39 of file uvfr_state_engine.c.

Referenced by _stateChangeDaemon(), changeVehicleState(), daqMasterTask(), and uvStartStateMachine().

5.2 State Engine API

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

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](#) 0b00000000000000011
- #define [UV_TASK_GENERIC_SVC](#) 0x0001U<<(2)
- #define [UV_TASK_MANAGER_MASK](#) 0b00000000000000011
- #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) (t->task_flags|=[UV_TASK_AWAITING_DELETION](#))
- #define [uvTaskResetDeletionBit](#)(t) (t->task_flags &=(~[UV_TASK_AWAITING_DELETION](#)))
- #define [uvTaskSetDelayBit](#)(t) (t->task_flags|=[UV_TASK_DELAYING](#))
- #define [uvTaskResetDelayBit](#)(t) (t->task_flags &=(~[UV_TASK_DELAYING](#)))
- #define [uvTaskIsDelaying](#)(t) ((t->task_flags & [UV_TASK_DELAYING](#)) == [UV_TASK_DELAYING](#))
- #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 necessary 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.

Functions

- `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 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 uvDelInitStateEngine` ()
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 gnarliness here, but not unacceptable levels. Pray this thing doesn't hang itself.

5.2.1 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

5.2.2 Macro Definition Documentation

5.2.2.1 UV_TASK_AWAITING_DELETION

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

Definition at line 193 of file uvfr_state_engine.h.

5.2.2.2 UV_TASK_DEADLINE_FIRM

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

Definition at line 197 of file uvfr_state_engine.h.

5.2.2.3 UV_TASK_DEADLINE_HARD

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

Definition at line 198 of file uvfr_state_engine.h.

5.2.2.4 UV_TASK_DEADLINE_MASK

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

Definition at line 199 of file uvfr_state_engine.h.

5.2.2.5 UV_TASK_DEADLINE_NOT_ENFORCED

```
#define UV_TASK_DEADLINE_NOT_ENFORCED 0x00
```

Definition at line 195 of file uvfr_state_engine.h.

5.2.2.6 UV_TASK_DEFER_DELETION

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

Definition at line 194 of file uvfr_state_engine.h.

5.2.2.7 UV_TASK_DELAYING

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

Definition at line 201 of file uvfr_state_engine.h.

5.2.2.8 UV_TASK_DORMANT_SVC

```
#define UV_TASK_DORMANT_SVC 0b0000000000000011
```

Definition at line 183 of file uvfr_state_engine.h.

5.2.2.9 UV_TASK_ERR_IN_CHILD

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

Definition at line 192 of file uvfr_state_engine.h.

5.2.2.10 UV_TASK_GENERIC_SVC

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

Definition at line 184 of file uvfr_state_engine.h.

5.2.2.11 UV_TASK_IS_CHILD

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

Definition at line 190 of file uvfr_state_engine.h.

5.2.2.12 UV_TASK_IS_ORPHAN

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

Definition at line 191 of file uvfr_state_engine.h.

5.2.2.13 UV_TASK_IS_PARENT

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

Definition at line 189 of file uvfr_state_engine.h.

5.2.2.14 UV_TASK_LOG_MEM_USAGE

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

Definition at line 187 of file uvfr_state_engine.h.

5.2.2.15 UV_TASK_LOG_START_STOP_TIME

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

Definition at line 186 of file uvfr_state_engine.h.

5.2.2.16 UV_TASK_MANAGER_MASK

```
#define UV_TASK_MANAGER_MASK 0b0000000000000011
```

Definition at line 185 of file uvfr_state_engine.h.

5.2.2.17 UV_TASK_MISSION_CRITICAL

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

Definition at line 200 of file uvfr_state_engine.h.

5.2.2.18 UV_TASK_PERIODIC_SVC

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

Definition at line 182 of file uvfr_state_engine.h.

5.2.2.19 UV_TASK_PRIO_INCREMENTATION

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

Definition at line 196 of file uvfr_state_engine.h.

5.2.2.20 UV_TASK_SCD_IGNORE

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

Definition at line 188 of file uvfr_state_engine.h.

5.2.2.21 UV_TASK_VEHICLE_APPLICATION

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

Definition at line 181 of file uvfr_state_engine.h.

5.2.2.22 uvTaskDelay

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

Value:

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

State engine aware vTaskDelay wrapper.

Parameters

<i>x</i>	
<i>t</i>	is how long to delay in ticks

Definition at line 274 of file uvfr_state_engine.h.

5.2.2.23 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 286 of file uvfr_state_engine.h.

5.2.2.24 uvTaskIsDelaying

```
#define uvTaskIsDelaying(
    t ) ((t->task_flags&UV_TASK_DELAYING)==UV_TASK_DELAYING)
```

Definition at line 267 of file uvfr_state_engine.h.

5.2.2.25 uvTaskResetDelayBit

```
#define uvTaskResetDelayBit(
    t ) (t->task_flags&=(~UV_TASK_DELAYING))
```

Definition at line 265 of file uvfr_state_engine.h.

5.2.2.26 uvTaskResetDeletionBit

```
#define uvTaskResetDeletionBit(  
    t ) (t->task_flags &= (~UV_TASK_AWAITING_DELETION))
```

Definition at line 261 of file uvfr_state_engine.h.

5.2.2.27 uvTaskSetDelayBit

```
#define uvTaskSetDelayBit(  
    t ) (t->task_flags|=UV_TASK_DELAYING)
```

Definition at line 263 of file uvfr_state_engine.h.

5.2.2.28 uvTaskSetDeletionBit

```
#define uvTaskSetDeletionBit(  
    t ) (t->task_flags|=UV_TASK_AWAITING_DELETION)
```

Definition at line 260 of file uvfr_state_engine.h.

5.2.3 Typedef Documentation

5.2.3.1 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

5.2.3.2 task_priority

```
typedef enum task_priority task_priority
```

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

5.2.3.3 task_status_block

```
typedef struct task_status_block task_status_block
```

Information about the task.

5.2.3.4 uv_os_settings

```
typedef struct uv_os_settings uv_os_settings
```

Settings that dictate state engine behavior.

5.2.3.5 uv_scd_response

```
typedef struct uv_scd_response uv_scd_response
```

5.2.3.6 uv_task_cmd

```
typedef enum uv_task_cmd_e uv_task_cmd
```

Special commands used to start and shutdown tasks.

5.2.3.7 uv_task_info

```
typedef struct uv_task_info uv_task_info
```

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

5.2.3.8 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.

5.2.3.9 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

5.2.4 Enumeration Type Documentation

5.2.4.1 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 135 of file uvfr_state_engine.h.

5.2.4.2 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 106 of file uvfr_state_engine.h.

5.2.4.3 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 96 of file uvfr_state_engine.h.

5.2.4.4 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 124 of file uvfr_state_engine.h.

5.2.4.5 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 81 of file uvfr_state_engine.h.

5.2.5 Function Documentation

5.2.5.1 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

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

Return values

<i>returns</i>	a memeber 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 86 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 `daqMasterTask()`, and `uvInit()`.

5.2.5.2 uvCreateTask()

```
uv_task_info* uvCreateTask ( )
```

This function gets called when you want to create a task, and register it with the task register. There's some gnarliness 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 249 of file `uvfr_state_engine.c`.

References `_next_task_id`, `_task_register`, `_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()`, and `initTempMonitor()`.

5.2.5.3 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 239 of file `uvfr_state_engine.c`.

References `killEmAll()`.

5.2.5.4 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 151 of file `uvfr_state_engine.c`.

References `__uvInitPanic()`, `_task_register`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `M_↵`
`AX_NUM_MANAGED_TASKS`, `svc_task_manager`, `task_manager`, `UV_OK`, and `uvCreateServiceTask()`.

Referenced by `uvInit()`.

5.2.5.5 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 179 of file uvfr_state_engine.c.

References previous_state, uv_task_info::stack_size, svc_task_manager, uv_task_info::task_flags, uv_task_info::task_function, uv_task_info::task_handle, 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, uvSVCManager(), uvTaskManager(), uvValidateManagedTasks(), and vehicle_state.

Referenced by uvInit().

5.3 State Engine Internals

Functions

- [uv_status addTaskToTaskRegister](#) ([uv_task_id](#) id, [uint8_t](#) assign_to_whom)
- [uv_status _uvValidateSpecificTask](#) ([uv_task_id](#) 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, [uv_task_info](#) *t)
: This is a function that starts tasks which are already registered in the system
- [static uv_status uvKillTaskViolently](#) ([uv_task_info](#) *t)
if a task refuses to comply with the SCD, then it has no choice but to be deleted. There is nothing that can be done.
- [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 uvSecureVehicle](#) ()
Function to put vehicle into safe state.
- [void __uvPanic](#) ([char](#) *msg, [uint8_t](#) msg_len, [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.
- [static uv_status proccessSCDMsg](#) ([uv_scd_response](#) *msg)
Helper function for the SCD, that proccesses a message, and double checks to make sure the task that sent the message isn't straight up lying to us.
- [void _stateChangeDaemon](#) ([void](#) *args) [PRIVILEGED_FUNCTION](#)
This collects all the data changing from different tasks, and makes sure that everything works properly.
- [uv_status uvInvokeSCD](#) ([void](#) *scd_params)
used to wake up the SCD
- [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](#) *task_name)
- [uv_task_info](#) * [uvGetTaskFromRTOSHandle](#) ([TaskHandle_t](#) t_handle)
Returns the pointer to the task info structure.

- `uv_status killEmAll ()`
The name should be pretty self explanatory.
- `void uvTaskManager (void *args) PRIVILEGED_FUNCTION`
The big papa task that deals with handling all of the others.

5.3.1 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.

5.3.2 Function Documentation

5.3.2.1 __uvPanic()

```
void __uvPanic (
    char * msg,
    uint8_t msg_len,
    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 697 of file uvfr_state_engine.c.

References uvSecureVehicle().

5.3.2.2 _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

<code>args</code>	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){//definatly not supposed to happen
                uvPanic("null scd response", 0);
            }
            if(processSCDMsg(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 868 of file uvfr_state_engine.c.

References `_LONGEST_SC_TIME`, `_next_task_id`, `_SC_DAEMON_PERIOD`, `_task_register`, `uv_task_info::active_states`, `uv_task_info::deletion_states`, `uv_scd_response::meta_id`, `proccessSCDMsg()`, `SCD_active`, `state_change_queue`, `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()`.

5.3.2.3 _uvValidateSpecificTask()

```

uv_status _uvValidateSpecificTask (
    uv_task_id id )

```

make sure the parameters of a task_info struct is valid

Definition at line 308 of file uvfr_state_engine.c.

References `_task_register`, `uv_task_info::active_states`, `uv_task_info::deletion_states`, `uv_task_info::suspension_states`, `uv_task_info::task_flags`, `uv_task_info::task_function`, `uv_task_info::task_name`, `UV_ERROR`, `UV_OK`, `UV_TASK_MANAGER_MASK`, and `UV_TASK_VEHICLE_APPLICATION`.

Referenced by `addTaskToTaskRegister()`, and `uvValidateManagedTasks()`.

5.3.2.4 addTaskToTaskRegister()

```

uv_status addTaskToTaskRegister (
    uv_task_id id,
    uint8_t assign_to_whom )

```

Definition at line 295 of file uvfr_state_engine.c.

References `_uvValidateSpecificTask()`, and `UV_OK`.

5.3.2.5 killEmAll()

```
uv_status killEmAll ( )
```

The name should be pretty self explanatory.

Definition at line 446 of file uvfr_state_engine.c.

References `_BV_32`, `_next_task_id`, `_task_register`, `UV_ERROR`, `UV_OK`, and `uvDeleteTask()`.

Referenced by `uvDeInitStateEngine()`.

5.3.2.6 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 715 of file uvfr_state_engine.c.

References `uv_task_info::cmd_data`, `uv_scd_response::meta_id`, `uv_scd_response::response_val`, `state_change_queue`, `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 `daqMasterTask()`, `odometerTask()`, `StartDrivingLoop()`, and `tempMonitorTask()`.

5.3.2.7 processSCDMsg()

```
static uv_status processSCDMsg (
    uv_scd_response * msg ) [static]
```

Helper function for the SCD, that processes a message, and double checks to make sure the task that sent the message isn't straight up lying to us.

This function is responsible for the following functionality:

- Make sure that the message claims that the deletion or suspension of a task is successful
- If a task claims that it is deleted, or suspended, then we must verify that this is the case

Get the id of the message, then use that to index the `_task_register` Mission critical stuff that stops ev from driving into a wall

Definition at line 799 of file uvfr_state_engine.c.

References `_task_register`, `uv_scd_response::meta_id`, `uv_scd_response::response_val`, `uv_task_info::task_handle`, `uv_task_info::task_state`, `UV_COULDNT_DELETE`, `UV_COULDNT_SUSPEND`, `UV_ERROR`, `UV_OK`, `UV_SUCCESSFUL_DELETION`, `UV_SUCCESSFUL_SUSPENSION`, `UV_TASK_DELETED`, and `UV_UNSAFE_STATE`.

Referenced by `_stateChangeDaemon()`.

5.3.2.8 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 756 of file uvfr_state_engine.c.

References uv_task_info::cmd_data, uv_scd_response::meta_id, uv_scd_response::response_val, state_change_queue, 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 daqMasterTask(), odometerTask(), StartDrivingLoop(), and tempMonitorTask().

5.3.2.9 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 551 of file uvfr_state_engine.c.

References UV_ERROR, and UV_OK.

5.3.2.10 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 1138 of file uvfr_state_engine.c.

References _next_svc_task_id, _next_task_id, _task_register, 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, and UV_TASK_NOT_STARTED.

Referenced by uvInitStateEngine(), and uvSVCTaskManager().

5.3.2.11 uvDeleteSVCTask()

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

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

Definition at line 1232 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().

5.3.2.12 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 491 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 uvTasksDelaying.

Referenced by _stateChangeDaemon(), and killEmAll().

5.3.2.13 uvGetTaskFromName()

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

Sometimes you just gotta deal with it lol

Definition at line 1340 of file uvfr_state_engine.c.

5.3.2.14 uvGetTaskFromRTOSHandle()

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

Returns the pointer to the task info structure.

Parameters

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

Return values

A	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 1352 of file `uvfr_state_engine.c`.

5.3.2.15 uvInvokeSCD()

```
uv_status uvInvokeSCD (
    void * scd_params ) [inline]
```

used to wake up the SCD

This is only called from `uvTaskManager` to wake up the SCD

Definition at line 1049 of file `uvfr_state_engine.c`.

5.3.2.16 uvKillTaskViolently()

```
static uv_status uvKillTaskViolently (
    uv_task_info * t ) [static]
```

if a task refuses to comply with the SCD, then it has no choice but to be deleted. There is nothing that can be done.

You will not win against the operating system. The first thing that needs to happen, is we will tell the kernel to release any resources owned by the task.

Definition at line 467 of file `uvfr_state_engine.c`.

References `UV_OK`.

Referenced by `uvRestartSVCTask()`.

5.3.2.17 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 1260 of file `uvfr_state_engine.c`.

References `UV_ERROR`, `UV_OK`, `uvDeleteSVCTask()`, `uvKillTaskViolently()`, and `uvStartSVCTask()`.

5.3.2.18 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 563 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().

5.3.2.19 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 669 of file uvfr_state_engine.c.

Referenced by __uvPanic().

5.3.2.20 uvStartSVCTask()

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

Function to start a service task specifically.

Definition at line 1178 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_SUS↵ PENDED.

Referenced by uvRestartSVCTask().

5.3.2.21 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

The function `osThreadCreate` returns null if it fails to create a thread. If that happens, we really do have a problem, so we will be returning an error value

Definition at line 365 of file `uvfr_state_engine.c`.

References `_BV_32`, `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 `uvSVCTaskManager()`.

5.3.2.22 uvSuspendSVCTask()

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

Function that suspends a service task.

Definition at line 1217 of file `uvfr_state_engine.c`.

References `uv_task_info::task_state`, `UV_ABORTED`, `UV_ERROR`, `UV_OK`, and `UV_TASK_SUSPENDED`.

5.3.2.23 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 590 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 `uvTasksDelaying`.

Referenced by `_stateChangeDaemon()`.

5.3.2.24 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 647 of file `uvfr_state_engine.c`.

References `uv_task_info::task_flags`, and `UV_TASK_MISSION_CRITICAL`.

5.3.2.25 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 is of utmost importance that this one DOES NOT CRASH. EVER. Wait for incoming instructions from tasks

Definition at line 1065 of file `uvfr_state_engine.c`.

References `uv_task_info::tmi`.

Referenced by `uvStartStateMachine()`.

5.3.2.26 uvValidateManagedTasks()

```
uv_status uvValidateManagedTasks ( )
```

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

Definition at line 343 of file uvfr_state_engine.c.

References `_next_task_id`, `_uvValidateSpecificTask()`, and `UV_OK`.

Referenced by `uvStartStateMachine()`.

5.4 UVFR Utilities

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

Modules

- [Utility Macros](#)
handy macros that perform very common functionality

5.4.1 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...

5.5 Utility Macros

handy macros that perform very common functionality

Macros

- `#define _BV(x) _BV_16(x)`
- `#define _BV_8(x) ((uint8_t)(0x01U >> x))`
- `#define _BV_16(x) ((uint16_t)(0x01U >> x))`
- `#define _BV_32(x) ((uint32_t)(0x01U >> x))`
- `#define endianSwap(x) endianSwap16(x)`
- `#define endianSwap8(x) x`
- `#define endianSwap16(x) (((x & 0x00FF)<<8) | ((x & 0xFF00)>>8))`
- `#define endianSwap32(x) (((x & 0x000000FF)<<16)|((x & 0x0000FF00)<<8)|((x & 0x00FF0000)>>8)|((x & 0xFF000000)>>16))`
- `#define deserializeSmallE16(x, i) ((x[i])|(x[i+1]<<8))`
- `#define deserializeSmallE32(x, i) ((x[i])|(x[i+1]<<8)|(x[i+2]<<16)|(x[i+3]<<24))`
- `#define deserializeBigE16(x, i) ((x[i]<<8)|(x[i+1]))`
- `#define deserializeBigE32(x, i) ((x[i]<<24)|(x[i+1]<<16)|(x[i+2]<<8)|(x[i+3]))`
- `#define serializeSmallE16(x, d, i) x[i]=d&0x00FF; x[i+1]=(d&0xFF00)>>8`
- `#define serializeSmallE32(x, d, i) x[i]=d&0x000000FF; x[i+1]=(d&0x0000FF00)>>8; x[i+2]=(d&0x00FF0000)>>16; x[i+3]=(d&0xFF000000)>>24`
- `#define serializeBigE16(x, d, i) x[i+1]=d&0x00FF; x[i]=(d&0xFF00)>>8`
- `#define serializeBigE32(x, d, i) x[i+3]=d&0x000000FF; x[i+2]=(d&0x0000FF00)>>8; x[i+1]=(d&0x00FF0000)>>16; x[i]=(d&0xFF000000)>>24`
- `#define setBits(x, msk, data) x=(x&(~msk)|data)`
macro to set bits of an int without touching the ones we dont want to edit
- `#define isPowerOfTwo(x) (x&&!(x&(x-1)))`
Returns a truthy value if "x" is a power of two.
- `#define safePtrRead(x) (*(x)?x:uvPanic("nullptr_deref",0))`
lil treat to help us avoid the dreaded null pointer dereference
- `#define safePtrWrite(p, x) (*(p)?p:&x)`
- `#define false 0`
- `#define true !false`

5.5.1 Detailed Description

handy macros that perform very common functionality

5.5.2 Macro Definition Documentation

5.5.2.1 _BV

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

Definition at line 69 of file uvfr_utils.h.

5.5.2.2 `_BV_16`

```
#define _BV_16(  
    x ) ((uint16_t) (0x01U >> x))
```

Definition at line 71 of file `uvfr_utils.h`.

5.5.2.3 `_BV_32`

```
#define _BV_32(  
    x ) ((uint32_t) (0x01U >> x))
```

Definition at line 72 of file `uvfr_utils.h`.

5.5.2.4 `_BV_8`

```
#define _BV_8(  
    x ) ((uint8_t) (0x01U >> x))
```

Definition at line 70 of file `uvfr_utils.h`.

5.5.2.5 `deserializeBigE16`

```
#define deserializeBigE16(  
    x,  
    i ) ((x[i]<<8)|(x[i+1]))
```

Definition at line 81 of file `uvfr_utils.h`.

5.5.2.6 `deserializeBigE32`

```
#define deserializeBigE32(  
    x,  
    i ) ((x[i]<<24)|(x[i+1]<<16)|(x[i+2]<<8)|(x[i+3]))
```

Definition at line 82 of file `uvfr_utils.h`.

5.5.2.7 deserializeSmallE16

```
#define deserializeSmallE16(  
    x,  
    i ) ((x[i])|(x[i+1]<<8))
```

Definition at line 79 of file uvfr_utils.h.

5.5.2.8 deserializeSmallE32

```
#define deserializeSmallE32(  
    x,  
    i ) ((x[i])|(x[i+1]<<8)|(x[i+2]<<16)|(x[i+3]<<24))
```

Definition at line 80 of file uvfr_utils.h.

5.5.2.9 endianSwap

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

Definition at line 74 of file uvfr_utils.h.

5.5.2.10 endianSwap16

```
#define endianSwap16(  
    x ) (((x & 0x00FF)<<8) | ((x & 0xFF00)>>8))
```

Definition at line 76 of file uvfr_utils.h.

5.5.2.11 endianSwap32

```
#define endianSwap32(  
    x ) (((x & 0x000000FF)<<16)|((x & 0x0000FF00)<<8)|((x & 0x00FF0000)>>8)|((x &  
0xFF000000)>>16))
```

Definition at line 77 of file uvfr_utils.h.

5.5.2.12 endianSwap8

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

Definition at line 75 of file uvfr_utils.h.

5.5.2.13 false

```
#define false 0
```

Wish.com Boolean

Definition at line 127 of file uvfr_utils.h.

5.5.2.14 isPowerOfTwo

```
#define isPowerOfTwo(  
    x ) (x&&!(x&(x-1)))
```

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

Definition at line 117 of file uvfr_utils.h.

5.5.2.15 safePtrRead

```
#define safePtrRead(  
    x ) (*(x)?x:uvPanic("nullptr_deref",0))
```

lil treat to help us avoid the dreaded null pointer dereference

Definition at line 122 of file uvfr_utils.h.

5.5.2.16 safePtrWrite

```
#define safePtrWrite(  
    p,  
    x ) (*(p)?p:&x)
```

Definition at line 123 of file uvfr_utils.h.

5.5.2.17 serializeBigE16

```
#define serializeBigE16(
    x,
    d,
    i ) x[i+1]=d&0x00FF; x[i]=(d&0xFF00)>>8
```

Definition at line 86 of file uvfr_utils.h.

5.5.2.18 serializeBigE32

```
#define serializeBigE32(
    x,
    d,
    i ) 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 87 of file uvfr_utils.h.

5.5.2.19 serializeSmallE16

```
#define serializeSmallE16(
    x,
    d,
    i ) x[i]=d&0x00FF; x[i+1]=(d&0xFF00)>>8
```

Definition at line 84 of file uvfr_utils.h.

5.5.2.20 serializeSmallE32

```
#define serializeSmallE32(
    x,
    d,
    i ) 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 85 of file uvfr_utils.h.

5.5.2.21 setBits

```
#define setBits(
    x,
    msk,
    data ) 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 0b00100010
uint8_t data = 0x0F; // val is 0b00001111
//now we deploy the macro
setBits(num,mask,data);
//now, num = 0b11010010
```

Definition at line 112 of file uvfr_utils.h.

5.5.2.22 true

```
#define true !false
```

Definition at line 128 of file uvfr_utils.h.

5.6 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

5.7 CMSIS

Modules

- [Stm32f4xx_system](#)

5.7.1 Detailed Description

5.8 Stm32f4xx_system

Modules

- [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](#)

5.8.1 Detailed Description

5.9 STM32F4xx_System_Private_Includes

Macros

- #define [HSE_VALUE](#) ((uint32_t)25000000)
- #define [HSI_VALUE](#) ((uint32_t)16000000)

5.9.1 Detailed Description

5.9.2 Macro Definition Documentation

5.9.2.1 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.

5.9.2.2 HSI_VALUE

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

Value of the Internal oscillator in Hz

Definition at line 55 of file system_stm32f4xx.c.

5.10 STM32F4xx_System_Private_TypeDefinitions

5.11 STM32F4xx_System_Private_Defines

5.12 STM32F4xx_System_Private_Macros

5.13 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}

5.13.1 Detailed Description

5.13.2 Variable Documentation

5.13.2.1 AHBPrescTable

```
const uint8_t AHBPrescTable[16] = {0, 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()`.

5.13.2.2 APBPrescTable

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

Definition at line 139 of file `system_stm32f4xx.c`.

5.13.2.3 SystemCoreClock

```
uint32_t SystemCoreClock = 16000000
```

Definition at line 137 of file `system_stm32f4xx.c`.

Referenced by `SystemCoreClockUpdate()`.

5.14 STM32F4xx_System_Private_FunctionPrototypes

5.15 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.

5.15.1 Detailed Description

5.15.2 Function Documentation

5.15.2.1 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`.

5.15.2.2 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 `access_control_info` Union Reference

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

Data Fields

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

6.1.1 Detailed Description

Definition at line 239 of file `uvfr_utils.h`.

6.1.2 Field Documentation

6.1.2.1 `bin_semaphore`

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

Definition at line 241 of file `uvfr_utils.h`.

6.1.2.2 `mutex`

```
struct uv\_mutex\_info access_control_info::mutex
```

Definition at line 240 of file `uvfr_utils.h`.

6.1.2.3 semaphore

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

Definition at line 242 of file uvfr_utils.h.

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

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

6.2 bms_settings_t Struct Reference

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

Data Fields

- uint32_t [mc_CAN_timeout](#)

6.2.1 Detailed Description

Definition at line 13 of file bms.h.

6.2.2 Field Documentation

6.2.2.1 mc_CAN_timeout

```
uint32_t bms_settings_t::mc_CAN_timeout
```

Definition at line 14 of file bms.h.

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

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

6.3 daq_child_task Struct Reference

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

Data Fields

- [rbnode](#) `treenode`
- `TaskHandle_t` [meta_task_handle](#)
- [daq_param_list_node](#) ** `param_list`
- `uint32_t` [period](#)

6.3.1 Detailed Description

Definition at line 80 of file `daq.h`.

6.3.2 Field Documentation

6.3.2.1 meta_task_handle

```
TaskHandle_t daq_child_task::meta_task_handle
```

Definition at line 82 of file `daq.h`.

6.3.2.2 param_list

```
daq_param_list_node** daq_child_task::param_list
```

Definition at line 83 of file `daq.h`.

6.3.2.3 period

```
uint32_t daq_child_task::period
```

Definition at line 84 of file `daq.h`.

6.3.2.4 treenode

```
rbnode daq_child_task::treenode
```

Definition at line 81 of file `daq.h`.

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

- `Core/Inc/daq.h`

6.4 daq_datapoint Struct Reference

This struct holds info of what needs to be logged.

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

Data Fields

- uint16_t [can_id](#)
- uint8_t [period](#)
- uint8_t [type](#)

6.4.1 Detailed Description

This struct holds info of what needs to be logged.

Definition at line 66 of file daq.h.

6.4.2 Field Documentation

6.4.2.1 can_id

```
uint16_t daq_datapoint::can_id
```

Definition at line 67 of file daq.h.

6.4.2.2 period

```
uint8_t daq_datapoint::period
```

Definition at line 68 of file daq.h.

6.4.2.3 type

```
uint8_t daq_datapoint::type
```

Definition at line 69 of file daq.h.

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

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

6.5 daq_loop_args Struct Reference

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

Data Fields

- `uint8_t throttle_daq_to_preserve_performance`
- `uint8_t minimum_daq_period`
- `uint16_t padding`
- `uint32_t padding2`
- `daq_datapoint datapoints` [`MAX_LOGGABLE_PARAMS`]

6.5.1 Detailed Description

Definition at line 72 of file daq.h.

6.5.2 Field Documentation

6.5.2.1 datapoints

```
daq_datapoint daq_loop_args::datapoints[MAX_LOGGABLE_PARAMS]
```

Definition at line 77 of file daq.h.

6.5.2.2 minimum_daq_period

```
uint8_t daq_loop_args::minimum_daq_period
```

Definition at line 74 of file daq.h.

6.5.2.3 padding

```
uint16_t daq_loop_args::padding
```

Definition at line 75 of file daq.h.

6.5.2.4 padding2

```
uint32_t daq_loop_args::padding2
```

Definition at line 76 of file daq.h.

6.5.2.5 throttle_daq_to_preserve_performance

```
uint8_t daq_loop_args::throttle_daq_to_preserve_performance
```

Definition at line 73 of file daq.h.

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

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

6.6 daq_param_list_node Struct Reference

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

Data Fields

- uint16_t [param_idx](#)
- struct [daq_param_list_node](#) * next

6.6.1 Detailed Description

Definition at line 58 of file daq.h.

6.6.2 Field Documentation

6.6.2.1 next

```
struct daq\_param\_list\_node* daq_param_list_node::next
```

Definition at line 60 of file daq.h.

6.6.2.2 param_idx

```
uint16_t daq_param_list_node::param_idx
```

Definition at line 59 of file daq.h.

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

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

6.7 driving_loop_args Struct Reference

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

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 [max_apps_value](#)
- uint16_t [min_BPS_value](#)
- uint16_t [max_BPS_value](#)
- uint16_t [apps_top](#)
- uint16_t [apps_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]

6.7.1 Detailed Description

Definition at line 108 of file driving_loop.h.

6.7.2 Field Documentation

6.7.2.1 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.

6.7.2.2 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.

6.7.2.3 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.

6.7.2.4 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.

6.7.2.5 accum_regen_soc_threshold

```
uint8_t driving_loop_args::accum_regen_soc_threshold
```

Vehicle will not regen if above this SOC

Definition at line 138 of file driving_loop.h.

6.7.2.6 apps_bottom

```
uint16_t driving_loop_args::apps_bottom
```

Min APPS input value, representing 0% throttle

Definition at line 128 of file driving_loop.h.

6.7.2.7 apps_implausibility_recovery_threshold

```
uint16_t driving_loop_args::apps_implausibility_recovery_threshold
```

Threshold for brake position

Definition at line 134 of file driving_loop.h.

6.7.2.8 apps_plausibility_check_threshold

```
uint16_t driving_loop_args::apps_plausibility_check_threshold
```

Threshold for accelerator position with

Definition at line 130 of file driving_loop.h.

Referenced by StartDrivingLoop().

6.7.2.9 apps_top

```
uint16_t driving_loop_args::apps_top
```

Max APPS input value, representing 100% throttle

Definition at line 127 of file driving_loop.h.

6.7.2.10 bps_implausibility_recovery_threshold

```
uint16_t driving_loop_args::bps_implausibility_recovery_threshold
```

Threshold for accellerator pedal position to recover fron APPS check

Definition at line 133 of file driving_loop.h.

6.7.2.11 bps_plausibility_check_threshold

```
uint16_t driving_loop_args::bps_plausibility_check_threshold
```

Brake pressure threshold for APPS

Definition at line 131 of file driving_loop.h.

Referenced by StartDrivingLoop().

6.7.2.12 dmodes

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

These are various driving modes

Definition at line 141 of file driving_loop.h.

6.7.2.13 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.

6.7.2.14 max_apps_offset

```
uint16_t driving_loop_args::max_apps_offset
```

maximum APPS offset

Definition at line 121 of file driving_loop.h.

Referenced by StartDrivingLoop().

6.7.2.15 max_apps_value

```
uint16_t driving_loop_args::max_apps_value
```

for detecting disconnects and short circuits

Definition at line 123 of file driving_loop.h.

Referenced by StartDrivingLoop().

6.7.2.16 max_BPS_value

```
uint16_t driving_loop_args::max_BPS_value
```

are the brakes valid?

Definition at line 125 of file driving_loop.h.

Referenced by StartDrivingLoop().

6.7.2.17 min_apps_offset

```
uint16_t driving_loop_args::min_apps_offset
```

minimum APPS offset

Definition at line 120 of file driving_loop.h.

6.7.2.18 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.

6.7.2.19 min_BPS_value

```
uint16_t driving_loop_args::min_BPS_value
```

are the brakes valid?

Definition at line 124 of file driving_loop.h.

6.7.2.20 num_driving_modes

```
uint8_t driving_loop_args::num_driving_modes
```

How many modes are actually populated

Definition at line 136 of file driving_loop.h.

6.7.2.21 period

```
uint8_t driving_loop_args::period
```

how often does the driving loop execute

Definition at line 137 of file driving_loop.h.

6.7.2.22 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](#)

6.8 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>
```

6.8.1 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](#)

6.9 drivingMode Struct Reference

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

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

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](#)

6.9.1 Detailed Description

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

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

6.9.2 Field Documentation

6.9.2.1 control_map_fn

```
uint8_t drivingMode::control_map_fn
```

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

6.9.2.2 dm_name

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

Name of mode, 15 chars + /0

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

6.9.2.3 flags

```
uint16_t drivingMode::flags
```

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

6.9.2.4 map_fn_params

```
drivingModeParams drivingMode::map_fn_params
```

Definition at line 94 of file driving_loop.h.

6.9.2.5 max_acc_pwr

```
uint32_t drivingMode::max_acc_pwr
```

Definition at line 87 of file driving_loop.h.

6.9.2.6 max_current

```
uint32_t drivingMode::max_current
```

Definition at line 89 of file driving_loop.h.

6.9.2.7 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](#)

6.10 drivingModeParams Union Reference

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

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

6.10.1 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](#)

6.11 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](#)

6.11.1 Detailed Description

struct to hold parameters used in an exponential torque map

Definition at line 56 of file `driving_loop.h`.

6.11.2 Field Documentation

6.11.2.1 gamma

```
float exp_torque_map_args::gamma
```

Definition at line 58 of file `driving_loop.h`.

6.11.2.2 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](#)

6.12 linear_torque_map_args Struct Reference

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

Data Fields

- `int32_t` [offset](#)
- `float` [slope](#)

6.12.1 Detailed Description

Definition at line 48 of file `driving_loop.h`.

6.12.2 Field Documentation

6.12.2.1 offset

```
int32_t linear_torque_map_args::offset
```

Definition at line 49 of file `driving_loop.h`.

6.12.2.2 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`

6.13 motor_controller_settings Struct Reference

```
#include <motor_controller.h>
```

Data Fields

- `uint32_t` [mc_CAN_timeout](#)

6.13.1 Detailed Description

Definition at line 146 of file `motor_controller.h`.

6.13.2 Field Documentation

6.13.2.1 mc_CAN_timeout

```
uint32_t motor_controllor_settings::mc_CAN_timeout
```

Definition at line 147 of file motor_controller.h.

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

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

6.14 p_status Struct Reference

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

Data Fields

- [uv_status peripheral_status](#)
- TickType_t [activation_time](#)

6.14.1 Detailed Description

Definition at line 302 of file uvfr_utils.h.

6.14.2 Field Documentation

6.14.2.1 activation_time

```
TickType_t p_status::activation_time
```

Definition at line 304 of file uvfr_utils.h.

6.14.2.2 peripheral_status

```
uv_status p_status::peripheral_status
```

Definition at line 303 of file `uvfr_utils.h`.

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

- `Core/Inc/uvfr_utils.h`

6.15 rbnode Struct Reference

Node of a Red-Black binary search tree.

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

Data Fields

- struct `rbnode` * `left`
- struct `rbnode` * `right`
- struct `rbnode` * `parent`
- void * `data`
- char `color`

6.15.1 Detailed Description

Node of a Red-Black binary search tree.

Definition at line 27 of file `rb_tree.h`.

6.15.2 Field Documentation

6.15.2.1 color

```
char rbnode::color
```

The color of the node (internal use only)

Definition at line 32 of file `rb_tree.h`.

Referenced by `checkBlackHeight()`, `deleteRepair()`, `insertRepair()`, `print()`, `rbCreate()`, `rbDelete()`, and `rbInsert()`.

6.15.2.2 data

```
void* rbnode::data
```

Pointer to some data contained by the tree

Definition at line 31 of file `rb_tree.h`.

Referenced by `checkOrder()`, `destroyAllNodes()`, `print()`, `rb_apply()`, `rbCreate()`, `rbDelete()`, `rbFind()`, and `rbInsert()`.

6.15.2.3 left

```
struct rbnode* rbnode::left
```

Left sub-tree

Definition at line 28 of file `rb_tree.h`.

Referenced by `checkBlackHeight()`, `checkOrder()`, `deleteRepair()`, `destroyAllNodes()`, `insertRepair()`, `print()`, `rb_↔apply()`, `rbCreate()`, `rbDelete()`, `rbFind()`, `rbInsert()`, `rbSuccessor()`, `rotateLeft()`, and `rotateRight()`.

6.15.2.4 parent

```
struct rbnode* rbnode::parent
```

Parent of node

Definition at line 30 of file `rb_tree.h`.

Referenced by `checkBlackHeight()`, `deleteRepair()`, `destroyAllNodes()`, `insertRepair()`, `rbCreate()`, `rbDelete()`, `rb_↔Insert()`, `rbSuccessor()`, `rotateLeft()`, and `rotateRight()`.

6.15.2.5 right

```
struct rbnode* rbnode::right
```

Right sub-tree

Definition at line 29 of file `rb_tree.h`.

Referenced by `checkBlackHeight()`, `checkOrder()`, `deleteRepair()`, `destroyAllNodes()`, `insertRepair()`, `print()`, `rb_↔apply()`, `rbCreate()`, `rbDelete()`, `rbFind()`, `rbInsert()`, `rbSuccessor()`, `rotateLeft()`, and `rotateRight()`.

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

- `Core/Inc/rb_tree.h`

6.16 rbtree Struct Reference

struct representing a binary search tree

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

Data Fields

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

6.16.1 Detailed Description

struct representing a binary search tree

Definition at line 39 of file `rb_tree.h`.

6.16.2 Field Documentation

6.16.2.1 `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 `checkOrder()`, `rbCreate()`, `rbFind()`, and `rbInsert()`.

6.16.2.2 `count`

```
int rbtree::count
```

number of items stored in the tree

Definition at line 53 of file `rb_tree.h`.

Referenced by `destroyAllNodes()`, `rbCreate()`, `rbDelete()`, and `rbInsert()`.

6.16.2.3 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 destroyAllNodes(), rbCreate(), rbDelete(), and rbInsert().

6.16.2.4 min

```
rbnode* rbtree::min
```

Pointer to minimum element

Definition at line 50 of file rb_tree.h.

Referenced by rbCreate(), rbDelete(), and rbInsert().

6.16.2.5 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().

6.16.2.6 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.

6.16.2.7 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](#)

6.17 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]

6.17.1 Detailed Description

struct for s-curve parameters for torque

Definition at line 66 of file driving_loop.h.

6.17.2 Field Documentation

6.17.2.1 a

```
int32_t s_curve_torque_map_args::a
```

Definition at line 67 of file driving_loop.h.

6.17.2.2 b

```
int32_t s_curve_torque_map_args::b
```

Definition at line 68 of file driving_loop.h.

6.17.2.3 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](#)

6.18 state_change_daemon_args Struct Reference

Data Fields

- `TaskHandle_t` [meta_task_handle](#)

6.18.1 Detailed Description

Definition at line 58 of file uvfr_state_engine.c.

6.18.2 Field Documentation

6.18.2.1 meta_task_handle

```
TaskHandle_t state_change_daemon_args::meta_task_handle
```

Definition at line 59 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](#)

6.19 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](#)

6.19.1 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 154 of file uvfr_state_engine.h.

6.19.2 Field Documentation

6.19.2.1 parent_msg_queue

```
QueueHandle_t task_management_info::parent_msg_queue
```

Definition at line 156 of file uvfr_state_engine.h.

6.19.2.2 task_handle

```
TaskHandle_t task_management_info::task_handle
```

Actual handle of parent

Definition at line 155 of file uvfr_state_engine.h.

Referenced by uvSVCTaskManager().

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

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

6.20 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](#)

6.20.1 Detailed Description

Information about the task.

Definition at line 162 of file uvfr_state_engine.h.

6.20.2 Field Documentation

6.20.2.1 task_high_water_mark

```
uint32_t task_status_block::task_high_water_mark
```

Definition at line 163 of file uvfr_state_engine.h.

6.20.2.2 task_report_time

```
TickType_t task_status_block::task_report_time
```

Definition at line 164 of file uvfr_state_engine.h.

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

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

6.21 uv_binary_semaphore_info Struct Reference

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

Data Fields

- SemaphoreHandle_t [handle](#)

6.21.1 Detailed Description

Definition at line 229 of file uvfr_utils.h.

6.21.2 Field Documentation

6.21.2.1 handle

```
SemaphoreHandle_t uv_binary_semaphore_info::handle
```

Definition at line 230 of file uvfr_utils.h.

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

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

6.22 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]

6.22.1 Detailed Description

Representative of a CAN message.

Definition at line 255 of file uvfr_utils.h.

6.22.2 Field Documentation

6.22.2.1 data

```
uint8_t uv_CAN_msg::data[8]
```

Definition at line 262 of file uvfr_utils.h.

Referenced by CANbusTxSvcDaemon().

6.22.2.2 dlc

```
uint8_t uv_CAN_msg::dlc
```

Definition at line 260 of file uvfr_utils.h.

Referenced by CANbusTxSvcDaemon().

6.22.2.3 flags

```
uint8_t uv_CAN_msg::flags
```

Definition at line 256 of file uvfr_utils.h.

Referenced by CANbusTxSvcDaemon().

6.22.2.4 msg_id

```
uint32_t uv_CAN_msg::msg_id
```

Definition at line 261 of file uvfr_utils.h.

Referenced by CANbusTxSvcDaemon().

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

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

6.23 uv_init_struct Struct Reference

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

Data Fields

- [bool use_default_settings](#)

6.23.1 Detailed Description

contains info relevant to initializing the vehicle

Definition at line 269 of file `uvfr_utils.h`.

6.23.2 Field Documentation

6.23.2.1 use_default_settings

`bool uv_init_struct::use_default_settings`

Definition at line 270 of file `uvfr_utils.h`.

Referenced by `MX_FREERTOS_Init()`.

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

- `Core/Inc/uvfr_utils.h`

6.24 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](#)

6.24.1 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 314 of file `uvfr_utils.h`.

6.24.2 Field Documentation

6.24.2.1 init_info_queue

QueueHandle_t uv_init_task_args::init_info_queue

Definition at line 316 of file uvfr_utils.h.

Referenced by BMS_Init(), initIMD(), initPDU(), MC_Startup(), and uvInit().

6.24.2.2 meta_task_handle

TaskHandle_t uv_init_task_args::meta_task_handle

Definition at line 317 of file uvfr_utils.h.

Referenced by BMS_Init(), initIMD(), initPDU(), MC_Startup(), and uvInit().

6.24.2.3 specific_args

void* uv_init_task_args::specific_args

Definition at line 315 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](#)

6.25 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](#)

6.25.1 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 340 of file `uvfr_utils.h`.

6.25.2 Field Documentation

6.25.2.1 device

```
uv_ext_device_id uv_init_task_response::device
```

Definition at line 342 of file `uvfr_utils.h`.

Referenced by `uvInit()`.

6.25.2.2 errmsg

```
char* uv_init_task_response::errmsg
```

Definition at line 344 of file `uvfr_utils.h`.

Referenced by `uvInit()`.

6.25.2.3 nchar

```
uint8_t uv_init_task_response::nchar
```

Definition at line 343 of file `uvfr_utils.h`.

Referenced by `uvInit()`.

6.25.2.4 status

```
uv_status uv_init_task_response::status
```

Definition at line 341 of file `uvfr_utils.h`.

Referenced by `uvInit()`.

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

- `Core/Inc/uvfr_utils.h`

6.26 uv_internal_params Struct Reference

Data used by the uvfr_utils library to do what it needs to do :)

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

Data Fields

- [uv_init_struct](#) * [init_params](#)
- [uv_vehicle_settings](#) * [vehicle_settings](#)
- [p_status](#) [peripheral_status](#) [8]
- [uint16_t](#) [e_code](#) [8]

6.26.1 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 326 of file uvfr_utils.h.

6.26.2 Field Documentation

6.26.2.1 e_code

```
uint16_t uv_internal_params::e_code[8]
```

Definition at line 330 of file uvfr_utils.h.

6.26.2.2 init_params

```
uv_init_struct* uv_internal_params::init_params
```

Definition at line 327 of file uvfr_utils.h.

6.26.2.3 peripheral_status

```
p_status uv_internal_params::peripheral_status[8]
```

Definition at line 329 of file uvfr_utils.h.

6.26.2.4 vehicle_settings

```
uv_vehicle_settings* uv_internal_params::vehicle_settings
```

Definition at line 328 of file uvfr_utils.h.

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

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

6.27 uv_mutex_info Struct Reference

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

Data Fields

- SemaphoreHandle_t [handle](#)

6.27.1 Detailed Description

Definition at line 224 of file uvfr_utils.h.

6.27.2 Field Documentation

6.27.2.1 handle

```
SemaphoreHandle_t uv_mutex_info::handle
```

Definition at line 225 of file uvfr_utils.h.

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

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

6.28 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](#)

6.28.1 Detailed Description

Settings that dictate state engine behavior.

Definition at line 171 of file uvfr_state_engine.h.

6.28.2 Field Documentation

6.28.2.1 max_svc_task_period

TickType_t uv_os_settings::max_svc_task_period

Definition at line 174 of file uvfr_state_engine.h.

6.28.2.2 max_task_period

TickType_t uv_os_settings::max_task_period

Definition at line 175 of file uvfr_state_engine.h.

6.28.2.3 min_task_period

TickType_t uv_os_settings::min_task_period

Definition at line 176 of file uvfr_state_engine.h.

6.28.2.4 svc_task_manager_period

TickType_t uv_os_settings::svc_task_manager_period

Definition at line 172 of file uvfr_state_engine.h.

6.28.2.5 task_manager_period

TickType_t uv_os_settings::task_manager_period

Definition at line 173 of file uvfr_state_engine.h.

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

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

6.29 uv_scd_response Struct Reference

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

Data Fields

- enum [uv_scd_response_e](#) response_val
- [uv_task_id](#) meta_id

6.29.1 Detailed Description

Definition at line 114 of file uvfr_state_engine.h.

6.29.2 Field Documentation

6.29.2.1 meta_id

[uv_task_id](#) uv_scd_response::meta_id

Definition at line 116 of file uvfr_state_engine.h.

Referenced by _stateChangeDaemon(), killSelf(), proccessSCDMsg(), and suspendSelf().

6.29.2.2 response_val

enum [uv_scd_response_e](#) uv_scd_response::response_val

Definition at line 115 of file uvfr_state_engine.h.

Referenced by killSelf(), proccessSCDMsg(), and suspendSelf().

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

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

6.30 uv_semaphore_info Struct Reference

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

Data Fields

- SemaphoreHandle_t [handle](#)

6.30.1 Detailed Description

Definition at line 234 of file uvfr_utils.h.

6.30.2 Field Documentation

6.30.2.1 handle

```
SemaphoreHandle_t uv_semaphore_info::handle
```

Definition at line 235 of file uvfr_utils.h.

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

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

6.31 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>
```

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](#)
- uint16_t [active_states](#)
- uint16_t [deletion_states](#)
- uint16_t [suspension_states](#)
- uint16_t [task_flags](#)

6.31.1 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 209 of file uvfr_state_engine.h.

6.31.2 Field Documentation

6.31.2.1 active_states

```
uint16_t uv_task_info::active_states
```

Definition at line 239 of file uvfr_state_engine.h.

Referenced by _stateChangeDaemon(), _uvValidateSpecificTask(), initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), uvCreateServiceTask(), uvCreateTask(), and uvSVCTaskManager().

6.31.2.2 cmd_data

```
uv_task_cmd uv_task_info::cmd_data
```

how we communicate with the task rn - THIS SUCKS SO BAD

Definition at line 230 of file uvfr_state_engine.h.

Referenced by daqMasterTask(), killSelf(), odometerTask(), StartDrivingLoop(), suspendSelf(), tempMonitorTask(), uvDeleteSVCTask(), uvDeleteTask(), and uvSuspendTask().

6.31.2.3 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 214 of file uvfr_state_engine.h.

6.31.2.4 deletion_states

```
uint16_t uv_task_info::deletion_states
```

Definition at line 240 of file uvfr_state_engine.h.

Referenced by `_stateChangeDaemon()`, `_uvValidateSpecificTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `uvCreateServiceTask()`, and `uvCreateTask()`.

6.31.2.5 parent

```
struct uv_task_info_t* uv_task_info::parent
```

info about the parent of the task

Definition at line 234 of file uvfr_state_engine.h.

Referenced by `uvCreateServiceTask()`, and `uvCreateTask()`.

6.31.2.6 stack_size

```
uint32_t uv_task_info::stack_size
```

Number of words allocated to the stack of the task

Definition at line 220 of file uvfr_state_engine.h.

Referenced by `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvStartStateMachine()`, `uvStartSVCTask()`, and `uvStartTask()`.

6.31.2.7 suspension_states

```
uint16_t uv_task_info::suspension_states
```

Definition at line 241 of file uvfr_state_engine.h.

Referenced by `_stateChangeDaemon()`, `_uvValidateSpecificTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `uvCreateServiceTask()`, and `uvCreateTask()`.

6.31.2.8 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 232 of file uvfr_state_engine.h.

Referenced by `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `StartDrivingLoop()`, and `uv↵StartSVCTask()`.

6.31.2.9 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 Incrimmentation - 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 243 of file uvfr_state_engine.h.

Referenced by `_stateChangeDaemon()`, `_uvValidateSpecificTask()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uv↵ScheduleTaskDeletion()`, `uvStartStateMachine()`, `uvStartSVCTask()`, and `uvTaskCrashHandler()`.

6.31.2.10 task_function

```
TaskFunction_t uv_task_info::task_function
```

Pointer to function that implements the task

Definition at line 216 of file uvfr_state_engine.h.

Referenced by `_uvValidateSpecificTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `uv↵CreateServiceTask()`, `uvCreateTask()`, `uvStartStateMachine()`, `uvStartSVCTask()`, `uvStartTask()`, and `uvSVCTask↵Manager()`.

6.31.2.11 task_handle

```
TaskHandle_t uv_task_info::task_handle
```

Handle of freeRTOS task control block

Definition at line 228 of file uvfr_state_engine.h.

Referenced by `_stateChangeDaemon()`, `killSelf()`, `proccessSCDMsg()`, `suspendSelf()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvDeleteSVCTask()`, `uvDeleteTask()`, `uvStartStateMachine()`, `uvStartSVCTask()`, `uvStartTask()`, and `uvSuspendTask()`.

6.31.2.12 task_id

```
uv_task_id uv_task_info::task_id
```

Detailed description after the member

Definition at line 210 of file uvfr_state_engine.h.

Referenced by `killSelf()`, `suspendSelf()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvDeleteTask()`, `uvScheduleTaskDeletion()`, `uvStartTask()`, and `uvSuspendTask()`.

6.31.2.13 task_name

```
char* uv_task_info::task_name
```

Detailed description after the member

Definition at line 211 of file uvfr_state_engine.h.

Referenced by `_uvValidateSpecificTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvStartStateMachine()`, `uvStartSVCTask()`, `uvStartTask()`, and `uvSVCTaskManager()`.

6.31.2.14 task_period

```
uv_timespan_ms uv_task_info::task_period
```

Maximum period between task execution

Definition at line 213 of file uvfr_state_engine.h.

Referenced by `daqMasterTask()`, `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `odometerTask()`, `StartDrivingLoop()`, `tempMonitorTask()`, and `uvStartStateMachine()`.

6.31.2.15 task_priority

```
osPriority uv_task_info::task_priority
```

Priority of the task. Int between 0 and 7

Definition at line 217 of file uvfr_state_engine.h.

Referenced by `initDaqTask()`, `initDrivingLoop()`, `initOdometer()`, `initTempMonitor()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvStartSVCTask()`, and `uvStartTask()`.

6.31.2.16 task_rx_mailbox

```
MessageBufferHandle_t uv_task_info::task_rx_mailbox
```

Incoming messages for this task

Definition at line 237 of file uvfr_state_engine.h.

6.31.2.17 task_state

```
uv_task_status uv_task_info::task_state
```

Definition at line 225 of file uvfr_state_engine.h.

Referenced by `_stateChangeDaemon()`, `killSelf()`, `proccessSCDMsg()`, `suspendSelf()`, `uvCreateServiceTask()`, `uvCreateTask()`, `uvDeleteSVCTask()`, `uvDeleteTask()`, `uvScheduleTaskDeletion()`, `uvStartSVCTask()`, `uvStartTask()`, `uvSuspendSVCTask()`, and `uvSuspendTask()`.

6.31.2.18 tmi

```
task_management_info* uv_task_info::tmi
```

how we will be communicating in the future

Definition at line 236 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](#)

6.32 uv_task_msg_t Struct Reference

Struct containing a message between two tasks.

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

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](#)

6.32.1 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 286 of file uvfr_utils.h.

6.32.2 Field Documentation

6.32.2.1 intended_recipient

```
uv\_task\_info* uv_task_msg_t::intended_recipient
```

Definition at line 289 of file uvfr_utils.h.

6.32.2.2 message_size

```
size_t uv_task_msg_t::message_size
```

Definition at line 291 of file uvfr_utils.h.

6.32.2.3 message_type

```
uint32_t uv_task_msg_t::message_type
```

Definition at line 287 of file uvfr_utils.h.

6.32.2.4 msg_contents

```
void* uv_task_msg_t::msg_contents
```

Definition at line 292 of file uvfr_utils.h.

6.32.2.5 sender

```
uv_task_info* uv_task_msg_t::sender
```

Definition at line 288 of file uvfr_utils.h.

6.32.2.6 time_sent

```
TickType_t uv_task_msg_t::time_sent
```

Definition at line 290 of file uvfr_utils.h.

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

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

6.33 uv_vehicle_settings Struct Reference

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

Data Fields

- struct [uv_os_settings](#) * [os_settings](#)
- struct [motor_controller_settings](#) * [mc_settings](#)
- [driving_loop_args](#) * [driving_loop_settings](#)
- void * [imd_settings](#)
- [bms_settings_t](#) * [bms_settings](#)
- [daq_loop_args](#) * [daq_settings](#)
- void * [pdu_settings](#)
- uint16_t [is_default](#)

6.33.1 Detailed Description

Definition at line 32 of file uvfr_settings.h.

6.33.2 Field Documentation

6.33.2.1 bms_settings

```
bms_settings_t* uv_vehicle_settings::bms_settings
```

Definition at line 40 of file uvfr_settings.h.

Referenced by uvInit().

6.33.2.2 daq_settings

```
daq_loop_args* uv_vehicle_settings::daq_settings
```

Definition at line 42 of file uvfr_settings.h.

6.33.2.3 driving_loop_settings

```
driving_loop_args* uv_vehicle_settings::driving_loop_settings
```

Definition at line 37 of file uvfr_settings.h.

6.33.2.4 imd_settings

```
void* uv_vehicle_settings::imd_settings
```

Definition at line 39 of file uvfr_settings.h.

Referenced by uvInit().

6.33.2.5 is_default

```
uint16_t uv_vehicle_settings::is_default
```

Bitfield containing info on whether each settings instance is factory default. 0 default, 1 altered

Definition at line 47 of file uvfr_settings.h.

6.33.2.6 mc_settings

```
struct motor\_controller\_settings* uv_vehicle_settings::mc_settings
```

Definition at line 35 of file uvfr_settings.h.

Referenced by uvInit().

6.33.2.7 os_settings

```
struct uv\_os\_settings* uv_vehicle_settings::os_settings
```

Definition at line 34 of file uvfr_settings.h.

Referenced by setupDefaultSettings().

6.33.2.8 pdu_settings

```
void* uv_vehicle_settings::pdu_settings
```

Definition at line 44 of file uvfr_settings.h.

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

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

6.34 veh_gen_info Struct Reference

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

6.34.1 Detailed Description

Definition at line 28 of file uvfr_settings.h.

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

- Core/Inc/[uvfr_settings.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"
```

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)

Variables

- ADC_HandleTypeDef [hadc1](#)
- ADC_HandleTypeDef [hadc2](#)

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.

7.1.2 Macro Definition Documentation

7.1.2.1 ADC1_BUF_LEN

```
#define ADC1_BUF_LEN 40
```

Definition at line 43 of file adc.h.

7.1.2.2 ADC1_CHNL_CNT

```
#define ADC1_CHNL_CNT 4
```

Definition at line 44 of file adc.h.

7.1.2.3 ADC1_MAX_VOLT

```
#define ADC1_MAX_VOLT 2850
```

Definition at line 55 of file adc.h.

7.1.2.4 ADC1_MIN_VOLT

```
#define ADC1_MIN_VOLT 500
```

Definition at line 54 of file adc.h.

7.1.2.5 ADC1_SAMPLES

```
#define ADC1_SAMPLES 10
```

Definition at line 45 of file adc.h.

7.1.2.6 ADC2_BUF_LEN

```
#define ADC2_BUF_LEN 2
```

Definition at line 48 of file adc.h.

7.1.2.7 ADC2_CHNL_CNT

```
#define ADC2_CHNL_CNT 2
```

Definition at line 49 of file adc.h.

7.1.2.8 ADC2_MAX_VOLT

```
#define ADC2_MAX_VOLT 69
```

Definition at line 58 of file adc.h.

7.1.2.9 ADC2_MIN_VOLT

```
#define ADC2_MIN_VOLT 69
```

Definition at line 57 of file adc.h.

7.1.2.10 ADC2_SAMPLES

```
#define ADC2_SAMPLES 1
```

Definition at line 50 of file adc.h.

7.1.3 Function Documentation

7.1.3.1 MX_ADC1_Init()

```
void MX_ADC1_Init (
    void )
```

Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

Configure the analog watchdog

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 32 of file adc.c.

References Error_Handler(), and hadc1.

Referenced by main().

7.1.3.2 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.

Definition at line 118 of file adc.c.

References Error_Handler(), and hadc2.

Referenced by main().

7.1.4 Variable Documentation

7.1.4.1 hadc1

ADC_HandleTypeDef hadc1

Definition at line 27 of file adc.c.

Referenced by HAL_ADC_LevelOutOfWindowCallback(), and MX_ADC1_Init().

7.1.4.2 hadc2

ADC_HandleTypeDef hadc2

Definition at line 28 of file adc.c.

Referenced by HAL_TIM_PeriodElapsedCallback(), and MX_ADC2_Init().

7.2 Core/Inc/bms.h File Reference

```
#include "main.h"
#include "uvfr_utils.h"
```

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)

7.2.1 Macro Definition Documentation

7.2.1.1 DEFAULT_BMS_CAN_TIMEOUT

```
#define DEFAULT_BMS_CAN_TIMEOUT ((uv_timespan_ms)200)
```

Definition at line 11 of file bms.h.

7.2.2 Typedef Documentation

7.2.2.1 bms_settings_t

```
typedef struct bms_settings_t bms_settings_t
```

7.2.3 Function Documentation

7.2.3.1 BMS_Init()

```
void BMS_Init (  
    void * args )
```

Definition at line 11 of file bms.c.

References BMS, uv_init_task_args::init_info_queue, uv_init_task_args::meta_task_handle, and UV_OK.

Referenced by uvInit().

7.3 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"
```

Macros

- `#define CAN_TX_DAEMON_NAME "CanTxDaemon"`
- `#define CAN_RX_DAEMON_NAME "CanRxDaemon"`

Typedefs

- typedef struct [uv_CAN_msg](#) [uv_CAN_msg](#)
- typedef enum [uv_status_t](#) [uv_status](#)

Functions

- 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.

Variables

- CAN_HandleTypeDef [hcan2](#)

7.3.1 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.

7.3.2 Macro Definition Documentation

7.3.2.1 CAN_RX_DAEMON_NAME

```
#define CAN_RX_DAEMON_NAME "CanRxDaemon"
```

Definition at line 41 of file can.h.

7.3.2.2 CAN_TX_DAEMON_NAME

```
#define CAN_TX_DAEMON_NAME "CanTxDaemon"
```

Definition at line 40 of file can.h.

7.3.3 Typedef Documentation

7.3.3.1 uv_CAN_msg

```
typedef struct uv_CAN_msg uv_CAN_msg
```

Definition at line 43 of file can.h.

7.3.3.2 uv_status

```
typedef enum uv_status_t uv_status
```

Definition at line 44 of file can.h.

7.3.4 Function Documentation

7.3.4.1 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

Definition at line 358 of file can.c.

References uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, hcan2, uv_CAN_msg::msg_id, Tx_msg←_queue, TxHeader, TxMailbox, and UV_CAN_EXTENDED_ID.

Referenced by uvSVCTaskManager().

7.3.4.2 HAL_CAN_RxFifo0MsgPendingCallback()

```
void HAL_CAN_RxFifo0MsgPendingCallback (  
    CAN_HandleTypeDef * hcan2 )
```

Definition at line 267 of file can.c.

References Error_Handler(), hcan2, RxData, and RxHeader.

7.3.4.3 HAL_CAN_RxFifo1MsgPendingCallback()

```
void HAL_CAN_RxFifo1MsgPendingCallback (
    CAN_HandleTypeDef * hcan2 )
```

Definition at line 303 of file can.c.

7.3.4.4 MX_CAN2_Init()

```
void MX_CAN2_Init (
    void )
```

Definition at line 119 of file can.c.

References Error_Handler(), hcan2, and TxHeader.

Referenced by main().

7.3.4.5 uvSendCanMSG()

```
uv_status uvSendCanMSG (
    uv_CAN_msg * 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. Check that the CAN Tx daemon is actually active

Definition at line 320 of file can.c.

References CAN_TX_DAEMON_NAME, Tx_msg_queue, UV_ERROR, and UV_OK.

7.3.5 Variable Documentation

7.3.5.1 hcan2

```
CAN_HandleTypeDef hcan2
```

Definition at line 116 of file can.c.

Referenced by IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), PDU_disable_brake↵_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_↵disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_↵fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), tempMonitor↵Task(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.4 Core/Inc/constants.h File Reference

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](#)
- uint8_t [TxData](#) [8]
- uint32_t [TxMailbox](#)
- uint8_t [RxData](#) [8]

7.4.1 Enumeration Type Documentation

7.4.1.1 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 15 of file constants.h.

7.4.2 Variable Documentation

7.4.2.1 RxData

uint8_t [RxData](#)[8]

Definition at line 9 of file constants.c.

Referenced by [HAL_CAN_RxFifo0MsgPendingCallback\(\)](#).

7.4.2.2 RxHeader

```
CAN_RxHeaderTypeDef RxHeader
```

Definition at line 5 of file constants.c.

Referenced by HAL_CAN_RxFifo0MsgPendingCallback().

7.4.2.3 TxData

```
uint8_t TxData[8]
```

Definition at line 7 of file constants.c.

Referenced by IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), tempMonitorTask(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.4.2.4 TxHeader

```
CAN_TxHeaderTypeDef TxHeader
```

Definition at line 4 of file constants.c.

Referenced by CANbusTxSvcDaemon(), IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), MX_CAN2_Init(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), tempMonitorTask(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.4.2.5 TxMailbox

```
uint32_t TxMailbox
```

Definition at line 8 of file constants.c.

Referenced by CANbusTxSvcDaemon(), IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), tempMonitorTask(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.5 Core/Inc/daq.h File Reference

```
#include "uvfr_utils.h"
```

Data Structures

- struct [daq_param_list_node](#)
- struct [daq_datapoint](#)
 - This struct holds info of what needs to be logged.*
- struct [daq_loop_args](#)
- struct [daq_child_task](#)

Macros

- `#define` [_NUM_LOGGABLE_PARAMS](#)

Typedefs

- typedef struct [daq_param_list_node](#) [daq_param_list_node](#)
- typedef struct [daq_datapoint](#) [daq_datapoint](#)
 - This struct holds info of what needs to be logged.*
- typedef struct [daq_loop_args](#) [daq_loop_args](#)
- typedef struct [daq_child_task](#) [daq_child_task](#)

Enumerations

- enum [data_type](#) {
[UV_UINT8](#), [UV_INT8](#), [UV_UINT16](#), [UV_INT16](#),
[UV_UINT32](#), [UV_INT32](#), [UV_FLOAT](#), [UV_DOUBLE](#),
[UV_STRING](#) }
- enum [loggable_params](#) {
[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](#), [ACC_POWER](#), [ACC_POWER_LIMIT](#), [APPS1_ADC_VAL](#),
[APPS2_ADC_VAL](#), [BPS1_ADC_VAL](#), [BPS2_ADC_VAL](#), [ACCELERATOR_PEDAL_RATIO](#),
[BRAKE_PRESSURE_PA](#), [POWER_DERATE_FACTOR](#), [CURRENT_DRIVING_MODE](#), [MAX_LOGGABLE_PARAMS](#)
}

Functions

- enum [uv_status_t](#) [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)

Variables

- void * [param_LUT](#) [126]

7.5.1 Macro Definition Documentation

7.5.1.1 `_NUM_LOGGABLE_PARAMS`

```
#define _NUM_LOGGABLE_PARAMS
```

Definition at line 13 of file daq.h.

7.5.2 Typedef Documentation

7.5.2.1 `daq_child_task`

```
typedef struct daq_child_task daq_child_task
```

7.5.2.2 `daq_datapoint`

```
typedef struct daq_datapoint daq_datapoint
```

This struct holds info of what needs to be logged.

7.5.2.3 `daq_loop_args`

```
typedef struct daq_loop_args daq_loop_args
```

7.5.2.4 `daq_param_list_node`

```
typedef struct daq_param_list_node daq_param_list_node
```

7.5.3 Enumeration Type Documentation

7.5.3.1 `data_type`

```
enum data_type
```

Enumerator

UV_UINT8	
UV_INT8	
UV_UINT16	
UV_INT16	
UV_UINT32	
UV_INT32	
UV_FLOAT	
UV_DOUBLE	
UV_STRING	

Definition at line 15 of file daq.h.

7.5.3.2 loggable_params

```
enum loggable_params
```

Enumerator

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	
ACC_POWER	
ACC_POWER_LIMIT	
APPS1_ADC_VAL	
APPS2_ADC_VAL	
BPS1_ADC_VAL	
BPS2_ADC_VAL	
ACCELERATOR_PEDAL_RATIO	
BRAKE_PRESSURE_PA	
POWER_DERATE_FACTOR	
CURRENT_DRIVING_MODE	
MAX_LOGGABLE_PARAMS	

Definition at line 28 of file daq.h.

7.5.4 Function Documentation

7.5.4.1 daqMasterTask()

```
void daqMasterTask (
    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

```
*/
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 62 of file daq.c.

References `changeVehicleState()`, `uv_task_info::cmd_data`, `killSelf()`, `suspendSelf()`, `uv_task_info::task_period`, `UV_DRIVING`, `UV_ERROR_STATE`, `UV_KILL_CMD`, `UV_READY`, `UV_SUSPEND_CMD`, and `vehicle_state`.

Referenced by `initDaqTask()`.

7.5.4.2 initDaqTask()

```
enum uv_status_t 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

Definition at line 30 of file daq.c.

References `_UV_DEFAULT_TASK_STACK_SIZE`, `uv_task_info::active_states`, `daqMasterTask()`, `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_DRIVING`, `UV_ERROR`, `UV_ERROR_STATE`, `UV_LAUNCH_CONTROL`, `UV_OK`, `UV_READY`, and `uvCreateTask()`.

Referenced by `uvInitStateEngine()`.

7.5.5 Variable Documentation

7.5.5.1 param_LUT

```
void* param_LUT[126]
```

Definition at line 7 of file daq.c.

7.6 Core/Inc/dash.h File Reference

```
#include "main.h"
```

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)

7.6.1 Enumeration Type Documentation

7.6.1.1 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.

7.6.2 Function Documentation

7.6.2.1 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](#).

7.6.2.2 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().

7.6.2.3 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.

7.7 Core/Inc/dma.h File Reference

This file contains all the function prototypes for the [dma.c](#) file.

```
#include "main.h"
```

Functions

- void [MX_DMA_Init](#) (void)

7.7.1 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.

7.7.2 Function Documentation

7.7.2.1 MX_DMA_Init()

```
void MX_DMA_Init (
    void )
```

Enable DMA controller clock

Definition at line 39 of file dma.c.

Referenced by main().

7.8 Core/Inc/driving_loop.h File Reference

```
#include "motor_controller.h"
#include "uvfr_utils.h"
```

Data Structures

- struct [linear_torque_map_args](#)
- struct [exp_torque_map_args](#)
struct to hold parameters used in an exponential torque map
- struct [s_curve_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](#)

Macros

- #define [DEFAULT_PERIOD](#) 50
DL_PERIOD is meant to represent how often the driving loop executes, in ms.

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 meant to represent the different types of pedal map
- enum `DL_internal_state` { `Plausible` = 0x01, `Implausible` = 0x02, `Erroneous` = 0x04 }

Functions

- enum `uv_status_t` `initDrivingLoop` (void *argument)
- void `StartDrivingLoop` (void *argument)
 Function implementing the ledTask thread.

7.8.1 Macro Definition Documentation

7.8.1.1 DEFAULT_PERIOD

```
#define DEFAULT_PERIOD 50
```

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 possible be dynamic in the future.

Just replace the number with the name of the variable, and you're all set.

Definition at line 26 of file `driving_loop.h`.

7.8.2 Typedef Documentation

7.8.2.1 driving_loop_args

```
typedef struct driving_loop_args driving_loop_args
```

7.8.2.2 drivingMode

```
typedef struct drivingMode drivingMode
```

This is where the driving mode and the `drivingModeParams` are at.

7.8.2.3 drivingModeParams

```
typedef union drivingModeParams drivingModeParams
```

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

7.8.2.4 exp_torque_map_args

```
typedef struct exp_torque_map_args exp_torque_map_args
```

struct to hold parameters used in an exponential torque map

7.8.2.5 linear_torque_map_args

```
typedef struct linear_torque_map_args linear_torque_map_args
```

7.8.2.6 MC_POWER

```
typedef uint16_t MC_POWER
```

Definition at line 16 of file driving_loop.h.

7.8.2.7 MC_RPM

```
typedef uint16_t MC_RPM
```

Definition at line 15 of file driving_loop.h.

7.8.2.8 MC_Torque

```
typedef uint16_t MC_Torque
```

Definition at line 14 of file driving_loop.h.

7.8.2.9 s_curve_torque_map_args

```
typedef struct s_curve_torque_map_args s_curve_torque_map_args
```

struct for s-curve parameters for torque

7.8.3 Enumeration Type Documentation

7.8.3.1 DL_internal_state

```
enum DL_internal_state
```


Enumerator

Plausible	
Implausible	
Erroneous	

Definition at line 42 of file driving_loop.h.

7.8.3.2 map_mode

enum `map_mode`

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.

7.8.4 Function Documentation**7.8.4.1 initDrivingLoop()**

```
enum uv_status_t initDrivingLoop (
    void * argument )
```

Definition at line 25 of file driving_loop.c.

References `uv_task_info::active_states`, `uv_task_info::deletion_states`, `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_LAUNCH_CONTROL`, `UV_OK`, `UV_READY`, and `uvCreateTask()`.

Referenced by `uvInitStateEngine()`.

7.8.4.2 StartDrivingLoop()

```
void StartDrivingLoop (  
    void * argument )
```

Function implementing the ledTask thread.

Parameters

<i>argument</i>	Not used for now. Will have configuration settings later.
-----------------	---

Return values

<i>None</i>	This function is made to be the meat and potatoes of the entire vehicle.
-------------	--

The first thing we do here is create some local variables here, to cache whatever variables need cached. We will be caching variables that are used very frequently in every single loop iteration, and are not

This line extracts the specific driving loop parameters as specified in the vehicle settings

```
*/
driving_loop_args* dl_params = (driving_loop_args*) params->task_args;
/**
```

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();
/**
```

Brake Plausibility Check

The way that this works is that if the brake pressure is greater than some threshold, and the accelerator pedal position is also greater than some threshold, the thing will register that a brake implausibility has occurred. This is not very cash money.

If this happens, we want to set the torque/speed output to zero. This will only reset itself once the brakes are set to less than a certain threshold. Honestly evil.

Definition at line 68 of file driving_loop.c.

References `adc1_APPS1`, `adc1_APPS2`, `adc1_BPS1`, `adc1_BPS2`, `driving_loop_args::apps_plausibility_check←_threshold`, `driving_loop_args::bps_plausibility_check_threshold`, `uv_task_info::cmd_data`, `Implausible`, `killSelf()`, `driving_loop_args::max_apps_offset`, `driving_loop_args::max_apps_value`, `driving_loop_args::max_BPS_value`, `Plausible`, `suspendSelf()`, `uv_task_info::task_args`, `uv_task_info::task_period`, `UV_KILL_CMD`, and `UV_SUSPEN←D_CMD`.

Referenced by `initDrivingLoop()`.

7.9 Core/Inc/errorLUT.h File Reference

Macros

- `#define _NUM_ERRORS_ 256`

7.9.1 Macro Definition Documentation

7.9.1.1 `_NUM_ERRORS_`

```
#define _NUM_ERRORS_ 256
```

Definition at line 11 of file errorLUT.h.

7.10 Core/Inc/FreeRTOSConfig.h File Reference

Macros

- #define `configENABLE_FPU` 0
- #define `configENABLE_MPU` 0
- #define `configUSE_PREEMPTION` 1
- #define `configSUPPORT_STATIC_ALLOCATION` 1
- #define `configSUPPORT_DYNAMIC_ALLOCATION` 1
- #define `configUSE_IDLE_HOOK` 0
- #define `configUSE_TICK_HOOK` 1
- #define `configCPU_CLOCK_HZ` (`SystemCoreClock`)
- #define `configTICK_RATE_HZ` ((`TickType_t`)1000)
- #define `configMAX_PRIORITIES` (7)
- #define `configMINIMAL_STACK_SIZE` ((`uint16_t`)128)
- #define `configTOTAL_HEAP_SIZE` ((`size_t`)15360)
- #define `configMAX_TASK_NAME_LEN` (16)
- #define `configUSE_16_BIT_TICKS` 0
- #define `configUSE_MUTEXES` 1
- #define `configQUEUE_REGISTRY_SIZE` 8
- #define `configCHECK_FOR_STACK_OVERFLOW` 2
- #define `configUSE_MALLOC_FAILED_HOOK` 1
- #define `configUSE_APPLICATION_TASK_TAG` 1
- #define `configUSE_COUNTING_SEMAPHORES` 1
- #define `configENABLE_BACKWARD_COMPATIBILITY` 0
- #define `configUSE_PORT_OPTIMISED_TASK_SELECTION` 1
- #define `configRECORD_STACK_HIGH_ADDRESS` 1
- #define `configCHECK_FOR_STACK_OVERFLOW` 2
- #define `configUSE_MALLOC_FAILED_HOOK` 1
- #define `configMESSAGE_BUFFER_LENGTH_TYPE` `size_t`
- #define `configUSE_CO_ROUTINES` 0
- #define `configMAX_CO_ROUTINE_PRIORITIES` (2)
- #define `configUSE_TIMERS` 1
- #define `configTIMER_TASK_PRIORITY` (2)
- #define `configTIMER_QUEUE_LENGTH` 10
- #define `configTIMER_TASK_STACK_DEPTH` 256
- #define `INCLUDE_vTaskPrioritySet` 1
- #define `INCLUDE_uxTaskPriorityGet` 1
- #define `INCLUDE_vTaskDelete` 1
- #define `INCLUDE_vTaskCleanUpResources` 1
- #define `INCLUDE_vTaskSuspend` 1
- #define `INCLUDE_vTaskDelayUntil` 1
- #define `INCLUDE_vTaskDelay` 1
- #define `INCLUDE_xTaskGetSchedulerState` 1
- #define `INCLUDE_xEventGroupSetBitFromISR` 1
- #define `INCLUDE_xTimerPendFunctionCall` 1

- `#define INCLUDE_xQueueGetMutexHolder 1`
- `#define INCLUDE_xSemaphoreGetMutexHolder 1`
- `#define INCLUDE_pcTaskGetTaskName 1`
- `#define INCLUDE_uxTaskGetStackHighWaterMark 1`
- `#define INCLUDE_uxTaskGetStackHighWaterMark2 1`
- `#define INCLUDE_xTaskGetCurrentTaskHandle 1`
- `#define INCLUDE_eTaskGetState 1`
- `#define INCLUDE_xTaskAbortDelay 1`
- `#define INCLUDE_xTaskGetHandle 1`
- `#define configPRIO_BITS 4`
- `#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY 15`
- `#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5`
- `#define configKERNEL_INTERRUPT_PRIORITY (configLIBRARY_LOWEST_INTERRUPT_PRIORITY << (8 - configPRIO_BITS))`
- `#define configMAX_SYSCALL_INTERRUPT_PRIORITY (configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8 - configPRIO_BITS))`
- `#define configASSERT(x) if ((x) == 0) {taskDISABLE_INTERRUPTS(); for(;;);}`
- `#define vPortSVCHandler SVC_Handler`
- `#define xPortPendSVHandler PendSV_Handler`
- `#define xPortSysTickHandler SysTick_Handler`
- `#define INCLUDE_xTaskDelayUntil 1`

7.10.1 Macro Definition Documentation

7.10.1.1 configASSERT

```
#define configASSERT(  
    x ) if ((x) == 0) {taskDISABLE_INTERRUPTS(); for( ;; );}
```

Definition at line 149 of file FreeRTOSConfig.h.

7.10.1.2 configCHECK_FOR_STACK_OVERFLOW [1/2]

```
#define configCHECK_FOR_STACK_OVERFLOW 2
```

Definition at line 81 of file FreeRTOSConfig.h.

7.10.1.3 configCHECK_FOR_STACK_OVERFLOW [2/2]

```
#define configCHECK_FOR_STACK_OVERFLOW 2
```

Definition at line 81 of file FreeRTOSConfig.h.

7.10.1.4 configCPU_CLOCK_HZ

```
#define configCPU_CLOCK_HZ ( SystemCoreClock )
```

Definition at line 63 of file FreeRTOSConfig.h.

7.10.1.5 configENABLE_BACKWARD_COMPATIBILITY

```
#define configENABLE_BACKWARD_COMPATIBILITY 0
```

Definition at line 76 of file FreeRTOSConfig.h.

7.10.1.6 configENABLE_FPU

```
#define configENABLE_FPU 0
```

Definition at line 55 of file FreeRTOSConfig.h.

7.10.1.7 configENABLE_MPU

```
#define configENABLE_MPU 0
```

Definition at line 56 of file FreeRTOSConfig.h.

7.10.1.8 configKERNEL_INTERRUPT_PRIORITY

```
#define configKERNEL_INTERRUPT_PRIORITY ( configLIBRARY_LOWEST_INTERRUPT_PRIORITY << (8 -  
configPRIO_BITS) )
```

Definition at line 141 of file FreeRTOSConfig.h.

7.10.1.9 configLIBRARY_LOWEST_INTERRUPT_PRIORITY

```
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY 15
```

Definition at line 131 of file FreeRTOSConfig.h.

7.10.1.10 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY

```
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5
```

Definition at line 137 of file FreeRTOSConfig.h.

7.10.1.11 configMAX_CO_ROUTINE_PRIORITIES

```
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )
```

Definition at line 91 of file FreeRTOSConfig.h.

7.10.1.12 configMAX_PRIORITIES

```
#define configMAX_PRIORITIES ( 7 )
```

Definition at line 65 of file FreeRTOSConfig.h.

7.10.1.13 configMAX_SYSCALL_INTERRUPT_PRIORITY

```
#define configMAX_SYSCALL_INTERRUPT_PRIORITY ( configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY <<  
( 8 - configPRIO_BITS ) )
```

Definition at line 144 of file FreeRTOSConfig.h.

7.10.1.14 configMAX_TASK_NAME_LEN

```
#define configMAX_TASK_NAME_LEN ( 16 )
```

Definition at line 68 of file FreeRTOSConfig.h.

7.10.1.15 configMESSAGE_BUFFER_LENGTH_TYPE

```
#define configMESSAGE_BUFFER_LENGTH_TYPE size_t
```

Definition at line 86 of file FreeRTOSConfig.h.

7.10.1.16 configMINIMAL_STACK_SIZE

```
#define configMINIMAL_STACK_SIZE ((uint16_t)128)
```

Definition at line 66 of file FreeRTOSConfig.h.

7.10.1.17 configPRIO_BITS

```
#define configPRIO_BITS 4
```

Definition at line 126 of file FreeRTOSConfig.h.

7.10.1.18 configQUEUE_REGISTRY_SIZE

```
#define configQUEUE_REGISTRY_SIZE 8
```

Definition at line 71 of file FreeRTOSConfig.h.

7.10.1.19 configRECORD_STACK_HIGH_ADDRESS

```
#define configRECORD_STACK_HIGH_ADDRESS 1
```

Definition at line 78 of file FreeRTOSConfig.h.

7.10.1.20 configSUPPORT_DYNAMIC_ALLOCATION

```
#define configSUPPORT_DYNAMIC_ALLOCATION 1
```

Definition at line 60 of file FreeRTOSConfig.h.

7.10.1.21 configSUPPORT_STATIC_ALLOCATION

```
#define configSUPPORT_STATIC_ALLOCATION 1
```

Definition at line 59 of file FreeRTOSConfig.h.

7.10.1.22 configTICK_RATE_HZ

```
#define configTICK_RATE_HZ ((TickType_t)1000)
```

Definition at line 64 of file FreeRTOSConfig.h.

7.10.1.23 configTIMER_QUEUE_LENGTH

```
#define configTIMER_QUEUE_LENGTH 10
```

Definition at line 96 of file FreeRTOSConfig.h.

7.10.1.24 configTIMER_TASK_PRIORITY

```
#define configTIMER_TASK_PRIORITY ( 2 )
```

Definition at line 95 of file FreeRTOSConfig.h.

7.10.1.25 configTIMER_TASK_STACK_DEPTH

```
#define configTIMER_TASK_STACK_DEPTH 256
```

Definition at line 97 of file FreeRTOSConfig.h.

7.10.1.26 configTOTAL_HEAP_SIZE

```
#define configTOTAL_HEAP_SIZE ((size_t)15360)
```

Definition at line 67 of file FreeRTOSConfig.h.

7.10.1.27 configUSE_16_BIT_TICKS

```
#define configUSE_16_BIT_TICKS 0
```

Definition at line 69 of file FreeRTOSConfig.h.

7.10.1.28 configUSE_APPLICATION_TASK_TAG

```
#define configUSE_APPLICATION_TASK_TAG 1
```

Definition at line 74 of file FreeRTOSConfig.h.

7.10.1.29 configUSE_CO_ROUTINES

```
#define configUSE_CO_ROUTINES 0
```

Definition at line 90 of file FreeRTOSConfig.h.

7.10.1.30 configUSE_COUNTING_SEMAPHORES

```
#define configUSE_COUNTING_SEMAPHORES 1
```

Definition at line 75 of file FreeRTOSConfig.h.

7.10.1.31 configUSE_IDLE_HOOK

```
#define configUSE_IDLE_HOOK 0
```

Definition at line 61 of file FreeRTOSConfig.h.

7.10.1.32 configUSE_MALLOC_FAILED_HOOK [1/2]

```
#define configUSE_MALLOC_FAILED_HOOK 1
```

Definition at line 82 of file FreeRTOSConfig.h.

7.10.1.33 configUSE_MALLOC_FAILED_HOOK [2/2]

```
#define configUSE_MALLOC_FAILED_HOOK 1
```

Definition at line 82 of file FreeRTOSConfig.h.

7.10.1.34 configUSE_MUTEXES

```
#define configUSE_MUTEXES 1
```

Definition at line 70 of file FreeRTOSConfig.h.

7.10.1.35 configUSE_PORT_OPTIMISED_TASK_SELECTION

```
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1
```

Definition at line 77 of file FreeRTOSConfig.h.

7.10.1.36 configUSE_PREEMPTION

```
#define configUSE_PREEMPTION 1
```

Definition at line 58 of file FreeRTOSConfig.h.

7.10.1.37 configUSE_TICK_HOOK

```
#define configUSE_TICK_HOOK 1
```

Definition at line 62 of file FreeRTOSConfig.h.

7.10.1.38 configUSE_TIMERS

```
#define configUSE_TIMERS 1
```

Definition at line 94 of file FreeRTOSConfig.h.

7.10.1.39 INCLUDE_eTaskGetState

```
#define INCLUDE_eTaskGetState 1
```

Definition at line 117 of file FreeRTOSConfig.h.

7.10.1.40 **INCLUDE_pcTaskGetTaskName**

```
#define INCLUDE_pcTaskGetTaskName 1
```

Definition at line 113 of file FreeRTOSConfig.h.

7.10.1.41 **INCLUDE_uxTaskGetStackHighWaterMark**

```
#define INCLUDE_uxTaskGetStackHighWaterMark 1
```

Definition at line 114 of file FreeRTOSConfig.h.

7.10.1.42 **INCLUDE_uxTaskGetStackHighWaterMark2**

```
#define INCLUDE_uxTaskGetStackHighWaterMark2 1
```

Definition at line 115 of file FreeRTOSConfig.h.

7.10.1.43 **INCLUDE_uxTaskPriorityGet**

```
#define INCLUDE_uxTaskPriorityGet 1
```

Definition at line 102 of file FreeRTOSConfig.h.

7.10.1.44 **INCLUDE_vTaskCleanUpResources**

```
#define INCLUDE_vTaskCleanUpResources 1
```

Definition at line 104 of file FreeRTOSConfig.h.

7.10.1.45 **INCLUDE_vTaskDelay**

```
#define INCLUDE_vTaskDelay 1
```

Definition at line 107 of file FreeRTOSConfig.h.

7.10.1.46 INCLUDE_vTaskDelayUntil

```
#define INCLUDE_vTaskDelayUntil 1
```

Definition at line 106 of file FreeRTOSConfig.h.

7.10.1.47 INCLUDE_vTaskDelete

```
#define INCLUDE_vTaskDelete 1
```

Definition at line 103 of file FreeRTOSConfig.h.

7.10.1.48 INCLUDE_vTaskPrioritySet

```
#define INCLUDE_vTaskPrioritySet 1
```

Definition at line 101 of file FreeRTOSConfig.h.

7.10.1.49 INCLUDE_vTaskSuspend

```
#define INCLUDE_vTaskSuspend 1
```

Definition at line 105 of file FreeRTOSConfig.h.

7.10.1.50 INCLUDE_xEventGroupSetBitFromISR

```
#define INCLUDE_xEventGroupSetBitFromISR 1
```

Definition at line 109 of file FreeRTOSConfig.h.

7.10.1.51 INCLUDE_xQueueGetMutexHolder

```
#define INCLUDE_xQueueGetMutexHolder 1
```

Definition at line 111 of file FreeRTOSConfig.h.

7.10.1.52 INCLUDE_xSemaphoreGetMutexHolder

```
#define INCLUDE_xSemaphoreGetMutexHolder 1
```

Definition at line 112 of file FreeRTOSConfig.h.

7.10.1.53 INCLUDE_xTaskAbortDelay

```
#define INCLUDE_xTaskAbortDelay 1
```

Definition at line 118 of file FreeRTOSConfig.h.

7.10.1.54 INCLUDE_xTaskDelayUntil

```
#define INCLUDE_xTaskDelayUntil 1
```

Definition at line 164 of file FreeRTOSConfig.h.

7.10.1.55 INCLUDE_xTaskGetCurrentTaskHandle

```
#define INCLUDE_xTaskGetCurrentTaskHandle 1
```

Definition at line 116 of file FreeRTOSConfig.h.

7.10.1.56 INCLUDE_xTaskGetHandle

```
#define INCLUDE_xTaskGetHandle 1
```

Definition at line 119 of file FreeRTOSConfig.h.

7.10.1.57 INCLUDE_xTaskGetSchedulerState

```
#define INCLUDE_xTaskGetSchedulerState 1
```

Definition at line 108 of file FreeRTOSConfig.h.

7.10.1.58 INCLUDE_xTimerPendFunctionCall

```
#define INCLUDE_xTimerPendFunctionCall 1
```

Definition at line 110 of file FreeRTOSConfig.h.

7.10.1.59 vPortSVCHandler

```
#define vPortSVCHandler SVC_Handler
```

Definition at line 154 of file FreeRTOSConfig.h.

7.10.1.60 xPortPendSVHandler

```
#define xPortPendSVHandler PendSV_Handler
```

Definition at line 155 of file FreeRTOSConfig.h.

7.10.1.61 xPortSysTickHandler

```
#define xPortSysTickHandler SysTick_Handler
```

Definition at line 160 of file FreeRTOSConfig.h.

7.11 Core/Inc/gpio.h File Reference

This file contains all the function prototypes for the [gpio.c](#) file.

```
#include "main.h"
```

Functions

- void [MX_GPIO_Init](#) (void)

7.11.1 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.

7.11.2 Function Documentation

7.11.2.1 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 [Blue_LED_Pin](#), [Orange_LED_Pin](#), [Red_LED_Pin](#), [Start_Button_Input_GPIO_Port](#), and [Start_Button_Input_Pin](#).

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

7.12 Core/Inc/imd.h File Reference

```
#include "main.h"
```

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_Vexi](#) = 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)

7.12.1 Enumeration Type Documentation

7.12.1.1 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 68 of file imd.h.

7.12.1.2 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 98 of file imd.h.

7.12.1.3 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	
Serial_number↵ _1	
Serial_number↵ _2	
Serial_number↵ _3	
Uptime_counter	

Definition at line 82 of file imd.h.

7.12.1.4 imd_status_bits

```
enum imd_status_bits
```

Enumerator

Isolation_status_bit0	
Isolation_status_bit1	
Low_Battery_Voltage	
High_Battery_Voltage	

Enumerator

Exc_off	
High_Uncertainty	
Touch_energy_fault	
Hardware_Error	

Definition at line 16 of file imd.h.

7.12.1.5 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 32 of file imd.h.

7.12.2 Function Documentation**7.12.2.1 IMD_Check_Battery_Voltage()**

```
void IMD_Check_Battery_Voltage (
    uint8_t Data[] )
```

Definition at line 351 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.2.2 IMD_Check_Error_Flags()

```
void IMD_Check_Error_Flags (
    uint8_t Data[] )
```

Definition at line 257 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().

7.12.2.3 IMD_Check_Isolation_Capacitances()

```
void IMD_Check_Isolation_Capacitances (
    uint8_t Data[] )
```

Definition at line 337 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.2.4 IMD_Check_Isolation_Resistances()

```
void IMD_Check_Isolation_Resistances (
    uint8_t Data[] )
```

Definition at line 312 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.12.2.5 IMD_Check_Isolation_State()

```
void IMD_Check_Isolation_State (
    uint8_t Data[] )
```

Definition at line 296 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.12.2.6 IMD_Check_Max_Battery_Working_Voltage()

```
void IMD_Check_Max_Battery_Working_Voltage (
    uint8_t Data[] )
```

Definition at line 388 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.2.7 IMD_Check_Part_Name()

```
void IMD_Check_Part_Name (
    uint8_t Data[] )
```

Definition at line 401 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().

7.12.2.8 IMD_Check_Safety_Touch_Current()

```
void IMD_Check_Safety_Touch_Current (
    uint8_t Data[] )
```

Definition at line 376 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.2.9 IMD_Check_Safety_Touch_Energy()

```
void IMD_Check_Safety_Touch_Energy (
    uint8_t Data[] )
```

Definition at line 369 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.2.10 IMD_Check_Serial_Number()

```
void IMD_Check_Serial_Number (
    uint8_t Data[] )
```

Definition at line 483 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()`.

7.12.2.11 IMD_Check_Status_Bits()

```
void IMD_Check_Status_Bits (
    uint8_t Data )
```

Definition at line 213 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()`.

7.12.2.12 IMD_Check_Temperature()

```
void IMD_Check_Temperature (
    uint8_t Data[] )
```

Definition at line 358 of file imd.c.

References `IMD_Temperature`.

Referenced by `IMD_Parse_Message()`.

7.12.2.13 IMD_Check_Uptime()

```
void IMD_Check_Uptime (
    uint8_t Data[] )
```

Definition at line 524 of file imd.c.

7.12.2.14 IMD_Check_Version()

```
void IMD_Check_Version (
    uint8_t Data[] )
```

Definition at line 443 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()`.

7.12.2.15 IMD_Check_Voltages_Vp_and_Vn()

```
void IMD_Check_Voltages_Vp_and_Vn (
    uint8_t Data[] )
```

Definition at line 344 of file imd.c.

Referenced by `IMD_Parse_Message()`.

7.12.2.16 IMD_Parse_Message()

```
void IMD_Parse_Message (
    int DLC,
    uint8_t Data[] )
```

Definition at line 68 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`.

7.12.2.17 IMD_Request_Status()

```
void IMD_Request_Status (
    uint8_t Status )
```

Definition at line 180 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()`.

7.12.2.18 IMD_Startup()

```
void IMD_Startup ( )
```

Definition at line 528 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`.

7.12.2.19 initIMD()

```
void initIMD (
    void * args )
```

Definition at line 554 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()`.

7.13 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"
```

Macros

- `#define Start_Button_Input_Pin GPIO_PIN_0`
- `#define Start_Button_Input_GPIO_Port GPIOA`
- `#define Start_Button_Input_EXTI_IRQn EXTI0_IRQn`
- `#define Orange_LED_Pin GPIO_PIN_13`
- `#define Orange_LED_GPIO_Port GPIOD`
- `#define Red_LED_Pin GPIO_PIN_14`
- `#define Red_LED_GPIO_Port GPIOD`
- `#define Blue_LED_Pin GPIO_PIN_15`
- `#define Blue_LED_GPIO_Port GPIOD`

Functions

- void `Error_Handler` (void)

This function is executed in case of error occurrence.

7.13.1 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.

7.13.2 Macro Definition Documentation

7.13.2.1 Blue_LED_GPIO_Port

```
#define Blue_LED_GPIO_Port GPIOD
```

Definition at line 72 of file main.h.

7.13.2.2 Blue_LED_Pin

```
#define Blue_LED_Pin GPIO_PIN_15
```

Definition at line 71 of file main.h.

7.13.2.3 Orange_LED_GPIO_Port

```
#define Orange_LED_GPIO_Port GPIOD
```

Definition at line 68 of file main.h.

7.13.2.4 Orange_LED_Pin

```
#define Orange_LED_Pin GPIO_PIN_13
```

Definition at line 67 of file main.h.

7.13.2.5 Red_LED_GPIO_Port

```
#define Red_LED_GPIO_Port GPIOD
```

Definition at line 70 of file main.h.

7.13.2.6 Red_LED_Pin

```
#define Red_LED_Pin GPIO_PIN_14
```

Definition at line 69 of file main.h.

7.13.2.7 Start_Button_Input_EXTI_IRQn

```
#define Start_Button_Input_EXTI_IRQn EXTI0_IRQn
```

Definition at line 66 of file main.h.

7.13.2.8 Start_Button_Input_GPIO_Port

```
#define Start_Button_Input_GPIO_Port GPIOA
```

Definition at line 65 of file main.h.

7.13.2.9 Start_Button_Input_Pin

```
#define Start_Button_Input_Pin GPIO_PIN_0
```

Definition at line 64 of file main.h.

7.13.3 Function Documentation

7.13.3.1 Error_Handler()

```
void Error_Handler (  
    void )
```

This function is executed in case of error occurrence.

Return values

None	
------	--

Definition at line 378 of file main.c.

Referenced by HAL_ADC_MspInit(), HAL_CAN_RxFifo0MsgPendingCallback(), IMD_Parse_Message(), IMD_Request_Status(), MC_Parse_Message(), MC_Request_Data(), MC_Send_Data(), MX_ADC1_Init(), MX_ADC2_Init(), MX_CAN2_Init(), MX_SPI1_Init(), MX_TIM3_Init(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), SystemClock_Config(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.14 Core/Inc/motor_controller.h File Reference

```
#include "main.h"
#include "uvfr_utils.h"
#include "uvfr_settings.h"
```

Data Structures

- struct [motor_controllor_settings](#)

Macros

- #define [DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT](#) ((uv_timespan_ms)200)

Typedefs

- typedef struct [motor_controllor_settings](#) [motor_controller_settings](#)

Enumerations

- enum [motor_controller_speed_parameters](#) { [N_actual](#) = 0x30, [N_set](#) = 0x31, [N_cmd](#) = 0x32, [N_error](#) = 0x33 }
- enum [motor_controller_current_parameters](#) { [todo1](#) = 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_magnetizing_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` {
`igbt_temperature` = 0x4A, `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, `eprom_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_Parse_Message` (int DLC, uint8_t Data[])
- void `MC_Request_Data` (uint8_t RegID)
- void `MC_Send_Data` (uint8_t RegID, uint8_t data_to_send[], uint8_t size)
- void `MC_Torque_Control` (int todo)
- void `MC_Speed_Control` (int todo)
- void `MC_Check_Error_Warning` (uint8_t Data[])
- void `MC_Check_Serial_Number` (uint8_t Data[])
- void `MC_Check_Firmware` (uint8_t Data[])
- void `MC_Startup` (void *args)

7.14.1 Macro Definition Documentation

7.14.1.1 DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT

```
#define DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT ((uv_timespan_ms)200)
```

Definition at line 15 of file `motor_controller.h`.

7.14.2 Typedef Documentation

7.14.2.1 motor_controller_settings

```
typedef struct motor_controllor_settings motor_controller_settings
```

7.14.3 Enumeration Type Documentation

7.14.3.1 motor_controller_current_parameters

```
enum motor_controller_current_parameters
```

Enumerator

todo1	
-------	--

Definition at line 27 of file motor_controller.h.

7.14.3.2 motor_controller_io

```
enum motor_controller_io
```

Enumerator

todo6969	
----------	--

Definition at line 110 of file motor_controller.h.

7.14.3.3 motor_controller_limp_mode

```
enum motor_controller_limp_mode
```

Enumerator

N_lim	
N_lim_plus	
N_lim_minus	

Definition at line 135 of file motor_controller.h.

7.14.3.4 motor_controller_measurements

```
enum motor_controller_measurements
```

Enumerator

DC_bus_voltage	
----------------	--

Definition at line 65 of file motor_controller.h.

7.14.3.5 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_magnetizing_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 31 of file motor_controller.h.

7.14.3.6 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 114 of file motor_controller.h.

7.14.3.7 motor_controller_repeating_time

```
enum motor_controller_repeating_time
```

Enumerator

none	
one_hundred_ms	

Definition at line 130 of file motor_controller.h.

7.14.3.8 motor_controller_speed_parameters

```
enum motor_controller_speed_parameters
```

Enumerator

N_actual	
N_set	
N_cmd	
N_error	

Definition at line 20 of file motor_controller.h.

7.14.3.9 motor_controller_startup

```
enum motor_controller_startup
```

Enumerator

clear_errors	
firmware_version	

Definition at line 141 of file motor_controller.h.

7.14.3.10 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 70 of file motor_controller.h.

7.14.3.11 motor_controller_temperatures

```
enum motor_controller_temperatures
```

Enumerator

igbt_temperature	
motor_temperature	
air_temperature	
current_derate_temperature	
temp_sensor_pt1	
temp_sensor_pt2	
temp_sensor_pt3	
temp_sensor_pt4	

Definition at line 54 of file motor_controller.h.

7.14.4 Function Documentation

7.14.4.1 MC_Check_Error_Warning()

```
void MC_Check_Error_Warning (
    uint8_t Data[] )
```

Definition at line 122 of file motor_controller.c.

References AC_current_offset_fault, ADC_measurement_problem, ADC_sequencer_problem, auxiliary_voltage_min_limit, bleed_resistor_overload, bleeder_resistor_warning, CAN_timeout_error, check_ecode_ID, critical_A_C_current, ecode_timeout_error, eeprom_read_error, feedback_signal_error, feedback_signal_problem, hardware_fault, IGBT_temp_max_limit, IGBT_temperature_warning, internal_hardware_voltage_problem, mains_voltage_max_limit, mains_voltage_min_limit, motor_temp_max_limit, motor_temperature_warning, parameter_conflict_detected, race_away_detected, rotate_field_enable_not_present_norun, rotate_field_enable_not_present_run, special_CPU_fault, speed_actual_resolution_limit, tripzone_glitch_detected, Vout_saturation_max_limit, warning_5, warning_9, and watchdog_reset.

Referenced by MC_Parse_Message().

7.14.4.2 MC_Check_Firmware()

```
void MC_Check_Firmware (
    uint8_t Data[] )
```

Definition at line 256 of file motor_controller.c.

7.14.4.3 MC_Check_Serial_Number()

```
void MC_Check_Serial_Number (
    uint8_t Data[] )
```

Definition at line 252 of file motor_controller.c.

7.14.4.4 MC_Parse_Message()

```
void MC_Parse_Message (
    int DLC,
    uint8_t Data[] )
```

Definition at line 26 of file motor_controller.c.

References [Error_Handler\(\)](#), [MC_Check_Error_Warning\(\)](#), and [motor_controller_errors_warnings](#).

7.14.4.5 MC_Request_Data()

```
void MC_Request_Data (
    uint8_t RegID )
```

Definition at line 47 of file motor_controller.c.

References [Error_Handler\(\)](#), [hcan2](#), [MC_CAN_ID_Tx](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

7.14.4.6 MC_Send_Data()

```
void MC_Send_Data (
    uint8_t RegID,
    uint8_t data_to_send[],
    uint8_t size )
```

Definition at line 69 of file motor_controller.c.

References [Error_Handler\(\)](#), [hcan2](#), [MC_CAN_ID_Tx](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

7.14.4.7 MC_Speed_Control()

```
void MC_Speed_Control (
    int todo )
```

7.14.4.8 MC_Startup()

```
void MC_Startup (
    void * args )
```

Definition at line 260 of file motor_controller.c.

References `uv_init_task_args::init_info_queue`, `uv_init_task_args::meta_task_handle`, `MOTOR_CONTROLLER`, `uv_init_task_args::specific_args`, and `UV_OK`.

Referenced by `uvInit()`.

7.14.4.9 MC_Torque_Control()

```
void MC_Torque_Control (
    int todo )
```

Definition at line 102 of file motor_controller.c.

7.15 Core/Inc/odometer.h File Reference

Functions

- [uv_status initOdometer](#) (void *args)
- void [odometerTask](#) (void *args)
, gotta know what the distance travelled is fam

7.15.1 Function Documentation

7.15.1.1 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()`.

7.15.1.2 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()`.

7.16 Core/Inc/oled.h File Reference

```
#include "uvfr_utils.h"
```

Functions

- void `wait` (uint32_t t)
- void `refresh_OLED` (volatile unsigned int Freq, volatile unsigned int Res)
- void `oled_Write_Cmd` (unsigned char)
- void `oled_Write_Data` (unsigned char)
- void `oled_Write` (unsigned char)
- void `oled_config` (void)
- void `amogus` (void)

7.16.1 Function Documentation

7.16.1.1 amogus()

```
void amogus (
    void )
```

7.16.1.2 oled_config()

```
void oled_config (
    void )
```

7.16.1.3 oled_Write()

```
void oled_Write (
    unsigned char )
```

7.16.1.4 oled_Write_Cmd()

```
void oled_Write_Cmd (
    unsigned char )
```

7.16.1.5 oled_Write_Data()

```
void oled_Write_Data (
    unsigned char )
```

7.16.1.6 refresh_OLED()

```
void refresh_OLED (
    volatile unsigned int Freq,
    volatile unsigned int Res )
```

7.16.1.7 wait()

```
void wait (
    uint32_t t )
```

7.17 Core/Inc/pdu.h File Reference

```
#include "main.h"
```

Enumerations

- enum `pdu_messages_5A` {
`enable_speaker_msg` = 0x1C, `disable_speaker_msg` = 0x0C, `enable_brake_light_msg` = 0x1B, `disable_brake_light_msg` = 0x0B,
`enable_motor_controller_msg` = 0x1E, `disable_motor_controller_msg` = 0x0E, `enable_shutdown_circuit_msg` = 0x1F, `disable_shutdown_circuit_msg` = 0x0F }
- enum `pdu_messages_20A` {
`enable_left_cooling_fan_msg` = 0x33, `disable_left_cooling_fan_msg` = 0x23, `enable_right_cooling_fan_msg` = 0x34, `disable_right_cooling_fan_msg` = 0x24,
`enable_coolant_pump_msg` = 0x31, `disable_coolant_pump_msg` = 0x21 }

Functions

- void `PDU_speaker_chirp` ()
- void `PDU_enable_brake_light` ()
- void `PDU_disable_brake_light` ()
- void `PDU_enable_motor_controller` ()
- void `PDU_disable_motor_controller` ()
- void `PDU_enable_shutdown_circuit` ()
- void `PDU_disable_shutdown_circuit` ()
- void `PDU_enable_cooling_fans` ()
- void `PDU_disable_cooling_fans` ()
- void `PDU_enable_coolant_pump` ()
- void `PDU_disable_coolant_pump` ()
- void `initPDU` (void *args)

7.17.1 Enumeration Type Documentation

7.17.1.1 pdu_messages_20A

enum `pdu_messages_20A`

Enumerator

<code>enable_left_cooling_fan_msg</code>	
<code>disable_left_cooling_fan_msg</code>	
<code>enable_right_cooling_fan_msg</code>	
<code>disable_right_cooling_fan_msg</code>	
<code>enable_coolant_pump_msg</code>	
<code>disable_coolant_pump_msg</code>	

Definition at line 24 of file pdu.h.

7.17.1.2 pdu_messages_5A

enum pdu_messages_5A

Enumerator

enable_speaker_msg	
disable_speaker_msg	
enable_brake_light_msg	
disable_brake_light_msg	
enable_motor_controller_msg	
disable_motor_controller_msg	
enable_shutdown_circuit_msg	
disable_shutdown_circuit_msg	

Definition at line 13 of file pdu.h.

7.17.2 Function Documentation

7.17.2.1 initPDU()

```
void initPDU (
    void * args )
```

Definition at line 183 of file pdu.c.

References uv_init_task_args::init_info_queue, uv_init_task_args::meta_task_handle, PDU, and UV_OK.

Referenced by uvInit().

7.17.2.2 PDU_disable_brake_light()

```
void PDU_disable_brake_light ( )
```

Definition at line 48 of file pdu.c.

References disable_brake_light_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and Tx↔Mailbox.

7.17.2.3 PDU_disable_coolant_pump()

```
void PDU_disable_coolant_pump ( )
```

Definition at line 170 of file pdu.c.

References `disable_coolant_pump_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `Tx↔Mailbox`.

7.17.2.4 PDU_disable_cooling_fans()

```
void PDU_disable_cooling_fans ( )
```

Definition at line 136 of file pdu.c.

References `disable_left_cooling_fan_msg`, `disable_right_cooling_fan_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_↔ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.17.2.5 PDU_disable_motor_controller()

```
void PDU_disable_motor_controller ( )
```

Definition at line 74 of file pdu.c.

References `disable_motor_controller_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.17.2.6 PDU_disable_shutdown_circuit()

```
void PDU_disable_shutdown_circuit ( )
```

Definition at line 100 of file pdu.c.

References `disable_shutdown_circuit_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.17.2.7 PDU_enable_brake_light()

```
void PDU_enable_brake_light ( )
```

Definition at line 34 of file pdu.c.

References `enable_brake_light_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `Tx↔Mailbox`.

7.17.2.8 PDU_enable_coolant_pump()

```
void PDU_enable_coolant_pump ( )
```

Definition at line 158 of file pdu.c.

References enable_coolant_pump_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

7.17.2.9 PDU_enable_cooling_fans()

```
void PDU_enable_cooling_fans ( )
```

Definition at line 115 of file pdu.c.

References enable_left_cooling_fan_msg, enable_right_cooling_fan_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

7.17.2.10 PDU_enable_motor_controller()

```
void PDU_enable_motor_controller ( )
```

Definition at line 62 of file pdu.c.

References enable_motor_controller_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

7.17.2.11 PDU_enable_shutdown_circuit()

```
void PDU_enable_shutdown_circuit ( )
```

Definition at line 87 of file pdu.c.

References enable_shutdown_circuit_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

7.17.2.12 PDU_speaker_chirp()

```
void PDU_speaker_chirp ( )
```

Definition at line 11 of file pdu.c.

References disable_speaker_msg, enable_speaker_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

7.18 Core/Inc/rb_tree.h File Reference

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) (&(rbt)->root)
- #define [RB_NIL](#)(rbt) (&(rbt)->nil)
- #define [RB_FIRST](#)(rbt) ((rbt)->root.left)
- #define [RB_MINIMAL](#)(rbt) ((rbt)->min)
- #define [RB_ISEMPY](#)(rbt) ((rbt)->root.left == &(rbt)->nil && (rbt)->root.right == &(rbt)->nil)
- #define [RB_APPLY](#)(rbt, f, c, o) rbapply_node((rbt), (rbt)->root.left, (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 *))
- [rbnode](#) * [rbInsert](#) ([rbtree](#) *rbt, void *data)
- void * [rbDelete](#) ([rbtree](#) *rbt, [rbnode](#) *node, int keep)
- int [rbCheckOrder](#) ([rbtree](#) *rbt, void *min, void *max)
- int [rbCheckBlackHeight](#) ([rbtree](#) *rbt)

7.18.1 Macro Definition Documentation

7.18.1.1 BLACK

```
#define BLACK 1
```

Definition at line 13 of file rb_tree.h.

7.18.1.2 RB_APPLY

```
#define RB_APPLY(  
    rbt,  
    f,  
    c,  
    o ) rbapply_node((rbt), (rbt)->root.left, (f), (c), (o))
```

Definition at line 63 of file rb_tree.h.

7.18.1.3 RB_DUP

```
#define RB_DUP 1
```

Definition at line 9 of file rb_tree.h.

7.18.1.4 RB_FIRST

```
#define RB_FIRST(  
    rbt ) ((rbt)->root.left)
```

Definition at line 59 of file rb_tree.h.

7.18.1.5 RB_ISEMPY

```
#define RB_ISEMPY(  
    rbt ) ((rbt)->root.left == &(rbt)->nil && (rbt)->root.right == &(rbt)->nil)
```

Definition at line 62 of file rb_tree.h.

7.18.1.6 RB_MIN

```
#define RB_MIN 1
```

Definition at line 10 of file `rb_tree.h`.

7.18.1.7 RB_MINIMAL

```
#define RB_MINIMAL(  
    rbt ) ((rbt)->min)
```

Definition at line 60 of file `rb_tree.h`.

7.18.1.8 RB_NIL

```
#define RB_NIL(  
    rbt ) (&(rbt)->nil)
```

Definition at line 58 of file `rb_tree.h`.

7.18.1.9 RB_ROOT

```
#define RB_ROOT(  
    rbt ) (&(rbt)->root)
```

Definition at line 57 of file `rb_tree.h`.

7.18.1.10 RED

```
#define RED 0
```

Definition at line 12 of file `rb_tree.h`.

7.18.2 Typedef Documentation

7.18.2.1 rbnode

```
typedef struct rbnode rbnode
```

Node of a Red-Black binary search tree.

7.18.3 Enumeration Type Documentation

7.18.3.1 rbtraversal

```
enum rbtraversal
```

Evil traversal method specifier for traversing the tree.

Enumerator

PREORDER	
INORDER	
POSTORDER	

Definition at line 18 of file rb_tree.h.

7.18.4 Function Documentation

7.18.4.1 rbApplyNode()

```
int rbApplyNode (
    rbtree * rbt,
    rbnode * node,
    int(*) (void *, void *) func,
    void * cookie,
    enum rbtraversal order )
```

7.18.4.2 rbCheckBlackHeight()

```
int rbCheckBlackHeight (
    rbtree * rbt )
```

Definition at line 551 of file rb_tree.c.

References `checkBlackHeight()`, `RB_FIRST`, `RB_NIL`, `RB_ROOT`, and `RED`.

Referenced by `rbPrint()`.

7.18.4.3 rbCheckOrder()

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

Definition at line 525 of file rb_tree.c.

References `checkOrder()`, and `RB_FIRST`.

7.18.4.4 rbCreate()

```
rbtree* rbCreate (
    int(*) (const void *, const void *) compare_func,
    void(*) (void *) destroy_func )
```

Create and initialize a binary search tree.

Definition at line 26 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.

7.18.4.5 rbDelete()

```
void* rbDelete (
    rbtree * rbt,
    rbnode * node,
    int keep )
```

Definition at line 344 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::count, rbnode::data, deleteRepair(), rbtree::destroy, rbnode::left, rbtree::min, rbnode::parent, RB_FIRST, RB_NIL, rbSuccessor(), RED, and rbnode::right.

7.18.4.6 rbDestroy()

```
void rbDestroy (
    rbtree * rbt )
```

Destroy the tree, and de-allocate it's elements.

Definition at line 59 of file rb_tree.c.

References destroyAllNodes(), and RB_FIRST.

7.18.4.7 rbFind()

```
rbnode* rbFind (
    rbtree * rbt,
    void * data )
```

Find a node of the tree based off the data you provide the tree.

Definition at line 69 of file rb_tree.c.

References rbtree::compare, rbnode::data, rbnode::left, RB_FIRST, RB_NIL, and rbnode::right.

7.18.4.8 rbInsert()

```
rbnode* rbInsert (
    rbtree * rbt,
    void * data )
```

Definition at line 191 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::compare, rbtree::count, rbnode::data, rbtree::destroy, insertRepair(), rbnode::left, rbtree::min, rbnode::parent, RB_FIRST, RB_MIN, RB_NIL, RB_ROOT, RED, and rbnode::right.

7.18.4.9 rbPrint()

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

Definition at line 587 of file rb_tree.c.

References print(), RB_FIRST, and rbCheckBlackHeight().

7.18.4.10 rbSuccessor()

```
rbnode* rbSuccessor (
    rbtree * rbt,
    rbnode * node )
```

Definition at line 90 of file rb_tree.c.

References rbnode::left, rbnode::parent, RB_NIL, RB_ROOT, and rbnode::right.

Referenced by rbDelete().

7.19 Core/Inc/spi.h File Reference

This file contains all the function prototypes for the [spi.c](#) file.

```
#include "main.h"
```

Functions

- void [MX_SPI1_Init](#) (void)

Variables

- SPI_HandleTypeDef [hspi1](#)

7.19.1 Detailed Description

This file contains all the function prototypes for the [spi.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.

7.19.2 Function Documentation

7.19.2.1 MX_SPI1_Init()

```
void MX_SPI1_Init (
    void )
```

Definition at line 30 of file spi.c.

References [Error_Handler\(\)](#), and [hspi1](#).

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

7.19.3 Variable Documentation

7.19.3.1 hspi1

```
SPI_HandleTypeDef hspi1
```

Definition at line 27 of file spi.c.

Referenced by [MX_SPI1_Init\(\)](#).

7.20 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_spi.h"
#include "stm32f4xx_hal_tim.h"
```

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_SPI_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](#) 8000000U

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 [USE_RTOS](#) 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 /* CRYPT 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_FMPMBUS_REGISTER_CALLBACKS 0U /* FMPSMBUS 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 expanse 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_RXBUFNB 4U /* 4 Rx buffers of size ETH_RX_BUF_SIZE */
• #define ETH_TXBUFNB 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) ((void)0U)

Include module's header file.

7.20.1 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.

7.20.2 Macro Definition Documentation

7.20.2.1 assert_param

```
#define assert_param(  
    expr ) ((void)0U)
```

Include module's header file.

Definition at line 488 of file stm32f4xx_hal_conf.h.

7.20.2.2 DATA_CACHE_ENABLE

```
#define DATA_CACHE_ENABLE 1U
```

Definition at line 155 of file stm32f4xx_hal_conf.h.

7.20.2.3 DP83848_PHY_ADDRESS

```
#define DP83848_PHY_ADDRESS
```

Definition at line 225 of file stm32f4xx_hal_conf.h.

7.20.2.4 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.

7.20.2.5 ETH_RXBUFNB

```
#define ETH_RXBUFNB 4U /* 4 Rx buffers of size ETH\_RX\_BUF\_SIZE */
```

Definition at line 219 of file stm32f4xx_hal_conf.h.

7.20.2.6 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.

7.20.2.7 ETH_TXBUFNB

```
#define ETH_TXBUFNB 4U /* 4 Tx buffers of size ETH_TX_BUF_SIZE */
```

Definition at line 220 of file stm32f4xx_hal_conf.h.

7.20.2.8 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.

7.20.2.9 HAL_ADC_MODULE_ENABLED

```
#define HAL_ADC_MODULE_ENABLED
```

Definition at line 41 of file stm32f4xx_hal_conf.h.

7.20.2.10 HAL_CAN_MODULE_ENABLED

```
#define HAL_CAN_MODULE_ENABLED
```

Definition at line 42 of file stm32f4xx_hal_conf.h.

7.20.2.11 HAL_CORTEX_MODULE_ENABLED

```
#define HAL_CORTEX_MODULE_ENABLED
```

Definition at line 90 of file stm32f4xx_hal_conf.h.

7.20.2.12 HAL_DMA_MODULE_ENABLED

```
#define HAL_DMA_MODULE_ENABLED
```

Definition at line 86 of file stm32f4xx_hal_conf.h.

7.20.2.13 HAL_EXTI_MODULE_ENABLED

```
#define HAL_EXTI_MODULE_ENABLED
```

Definition at line 85 of file stm32f4xx_hal_conf.h.

7.20.2.14 HAL_FLASH_MODULE_ENABLED

```
#define HAL_FLASH_MODULE_ENABLED
```

Definition at line 88 of file stm32f4xx_hal_conf.h.

7.20.2.15 HAL_GPIO_MODULE_ENABLED

```
#define HAL_GPIO_MODULE_ENABLED
```

Definition at line 84 of file stm32f4xx_hal_conf.h.

7.20.2.16 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.

7.20.2.17 HAL_PWR_MODULE_ENABLED

```
#define HAL_PWR_MODULE_ENABLED
```

Definition at line 89 of file stm32f4xx_hal_conf.h.

7.20.2.18 HAL_RCC_MODULE_ENABLED

```
#define HAL_RCC_MODULE_ENABLED
```

Definition at line 87 of file stm32f4xx_hal_conf.h.

7.20.2.19 HAL_SPI_MODULE_ENABLED

```
#define HAL_SPI_MODULE_ENABLED
```

Definition at line 65 of file stm32f4xx_hal_conf.h.

7.20.2.20 HAL_TIM_MODULE_ENABLED

```
#define HAL_TIM_MODULE_ENABLED
```

Definition at line 66 of file stm32f4xx_hal_conf.h.

7.20.2.21 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.

7.20.2.22 HSE_VALUE

```
#define HSE_VALUE 8000000U
```

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.

7.20.2.23 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.

7.20.2.24 INSTRUCTION_CACHE_ENABLE

```
#define INSTRUCTION_CACHE_ENABLE 1U
```

Definition at line 154 of file stm32f4xx_hal_conf.h.

7.20.2.25 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.

7.20.2.26 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.

7.20.2.27 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.

7.20.2.28 MAC_ADDR0

```
#define MAC_ADDR0 2U
```

Uncomment the line below to expanse the "assert_param" macro in the HAL drivers code.

Definition at line 209 of file stm32f4xx_hal_conf.h.

7.20.2.29 MAC_ADDR1

```
#define MAC_ADDR1 0U
```

Definition at line 210 of file stm32f4xx_hal_conf.h.

7.20.2.30 MAC_ADDR2

```
#define MAC_ADDR2 0U
```

Definition at line 211 of file stm32f4xx_hal_conf.h.

7.20.2.31 MAC_ADDR3

```
#define MAC_ADDR3 0U
```

Definition at line 212 of file stm32f4xx_hal_conf.h.

7.20.2.32 MAC_ADDR4

```
#define MAC_ADDR4 0U
```

Definition at line 213 of file stm32f4xx_hal_conf.h.

7.20.2.33 MAC_ADDR5

```
#define MAC_ADDR5 0U
```

Definition at line 214 of file stm32f4xx_hal_conf.h.

7.20.2.34 PHY_AUTONEGO_COMPLETE

```
#define PHY_AUTONEGO_COMPLETE ((uint16_t)0x0020U)
```

Auto-Negotiation process completed

Definition at line 250 of file stm32f4xx_hal_conf.h.

7.20.2.35 PHY_AUTONEGOTIATION

```
#define PHY_AUTONEGOTIATION ((uint16_t)0x1000U)
```

Enable auto-negotiation function

Definition at line 245 of file stm32f4xx_hal_conf.h.

7.20.2.36 PHY_BCR

```
#define PHY_BCR ((uint16_t)0x0000U)
```

Transceiver Basic Control Register

Definition at line 236 of file stm32f4xx_hal_conf.h.

7.20.2.37 PHY_BSR

```
#define PHY_BSR ((uint16_t)0x0001U)
```

Transceiver Basic Status Register

Definition at line 237 of file stm32f4xx_hal_conf.h.

7.20.2.38 PHY_CONFIG_DELAY

```
#define PHY_CONFIG_DELAY 0x00000FFFU
```

Definition at line 229 of file stm32f4xx_hal_conf.h.

7.20.2.39 PHY_DUPLEX_STATUS

```
#define PHY_DUPLEX_STATUS ((uint16_t))
```

PHY Duplex mask

Definition at line 258 of file stm32f4xx_hal_conf.h.

7.20.2.40 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.

7.20.2.41 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.

7.20.2.42 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.

7.20.2.43 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.

7.20.2.44 PHY_ISOLATE

```
#define PHY_ISOLATE ((uint16_t)0x0400U)
```

Isolate PHY from MII

Definition at line 248 of file stm32f4xx_hal_conf.h.

7.20.2.45 PHY_JABBER_DETECTION

```
#define PHY_JABBER_DETECTION ((uint16_t)0x0002U)
```

Jabber condition detected

Definition at line 252 of file stm32f4xx_hal_conf.h.

7.20.2.46 PHY_LINKED_STATUS

```
#define PHY_LINKED_STATUS ((uint16_t)0x0004U)
```

Valid link established

Definition at line 251 of file stm32f4xx_hal_conf.h.

7.20.2.47 PHY_LOOPBACK

```
#define PHY_LOOPBACK ((uint16_t)0x4000U)
```

Select loop-back mode

Definition at line 240 of file stm32f4xx_hal_conf.h.

7.20.2.48 PHY_POWERDOWN

```
#define PHY_POWERDOWN ((uint16_t)0x0800U)
```

Select the power down mode

Definition at line 247 of file stm32f4xx_hal_conf.h.

7.20.2.49 PHY_READ_TO

```
#define PHY_READ_TO 0x0000FFFFU
```

Definition at line 231 of file stm32f4xx_hal_conf.h.

7.20.2.50 PHY_RESET

```
#define PHY_RESET ((uint16_t)0x8000U)
```

PHY Reset

Definition at line 239 of file stm32f4xx_hal_conf.h.

7.20.2.51 PHY_RESET_DELAY

```
#define PHY_RESET_DELAY 0x000000FFU
```

Definition at line 227 of file stm32f4xx_hal_conf.h.

7.20.2.52 PHY_RESTART_AUTONEGOTIATION

```
#define PHY_RESTART_AUTONEGOTIATION ((uint16_t)0x0200U)
```

Restart auto-negotiation function

Definition at line 246 of file stm32f4xx_hal_conf.h.

7.20.2.53 PHY_SPEED_STATUS

```
#define PHY_SPEED_STATUS ((uint16_t))
```

PHY Speed mask

Definition at line 257 of file stm32f4xx_hal_conf.h.

7.20.2.54 PHY_SR

```
#define PHY_SR ((uint16_t))
```

PHY status register Offset

Definition at line 255 of file stm32f4xx_hal_conf.h.

7.20.2.55 PHY_WRITE_TO

```
#define PHY_WRITE_TO 0x0000FFFFU
```

Definition at line 232 of file stm32f4xx_hal_conf.h.

7.20.2.56 PREFETCH_ENABLE

```
#define PREFETCH_ENABLE 1U
```

Definition at line 153 of file stm32f4xx_hal_conf.h.

7.20.2.57 TICK_INT_PRIORITY

```
#define TICK_INT_PRIORITY 15U
```

tick interrupt priority

Definition at line 151 of file stm32f4xx_hal_conf.h.

7.20.2.58 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.

7.20.2.59 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.

7.20.2.60 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.

7.20.2.61 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.

7.20.2.62 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.

7.20.2.63 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.

7.20.2.64 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.

7.20.2.65 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.

7.20.2.66 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.

7.20.2.67 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.

7.20.2.68 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.

7.20.2.69 USE_HAL_FMPMBUS_REGISTER_CALLBACKS

```
#define USE_HAL_FMPMBUS_REGISTER_CALLBACKS 0U /* FMPMBUS register callback disabled */
```

Definition at line 171 of file stm32f4xx_hal_conf.h.

7.20.2.70 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.

7.20.2.71 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.

7.20.2.72 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.

7.20.2.73 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.

7.20.2.74 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.

7.20.2.75 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.

7.20.2.76 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.

7.20.2.77 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.

7.20.2.78 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.

7.20.2.79 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.

7.20.2.80 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.

7.20.2.81 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.

7.20.2.82 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.

7.20.2.83 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.

7.20.2.84 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.

7.20.2.85 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.

7.20.2.86 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.

7.20.2.87 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.

7.20.2.88 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.

7.20.2.89 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.

7.20.2.90 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.

7.20.2.91 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.

7.20.2.92 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.

7.20.2.93 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.

7.20.2.94 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.

7.20.2.95 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.

7.20.2.96 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.

7.20.2.97 USE_RTOS

```
#define USE_RTOS 0U
```

Definition at line 152 of file stm32f4xx_hal_conf.h.

7.20.2.98 USE_SPI_CRC

```
#define USE_SPI_CRC 0U
```

Definition at line 267 of file stm32f4xx_hal_conf.h.

7.20.2.99 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.

7.21 Core/Inc/stm32f4xx_it.h File Reference

This file contains the headers of the interrupt handlers.

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 [DebugMon_Handler](#) (void)
This function handles Debug monitor.
- void [EXTI0_IRQHandler](#) (void)
This function handles EXTI line0 interrupt.
- void [TIM1_UP_TIM10_IRQHandler](#) (void)
This function handles TIM1 update interrupt and TIM10 global interrupt.
- void [DMA2_Stream0_IRQHandler](#) (void)
This function handles DMA2 stream0 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.

7.21.1 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.

7.21.2 Function Documentation

7.21.2.1 BusFault_Handler()

```
void BusFault_Handler (  
    void )
```

This function handles Pre-fetch fault, memory access fault.

Definition at line 117 of file stm32f4xx_it.c.

7.21.2.2 CAN2_RX0_IRQHandler()

```
void CAN2_RX0_IRQHandler (  
    void )
```

This function handles CAN2 RX0 interrupts.

Definition at line 223 of file stm32f4xx_it.c.

References hcan2.

7.21.2.3 CAN2_RX1_IRQHandler()

```
void CAN2_RX1_IRQHandler (  
    void )
```

This function handles CAN2 RX1 interrupt.

Definition at line 237 of file stm32f4xx_it.c.

References hcan2.

7.21.2.4 CAN2_TX_IRQHandler()

```
void CAN2_TX_IRQHandler (
    void )
```

This function handles CAN2 TX interrupts.

Definition at line 209 of file stm32f4xx_it.c.

References hcan2.

7.21.2.5 DebugMon_Handler()

```
void DebugMon_Handler (
    void )
```

This function handles Debug monitor.

Definition at line 147 of file stm32f4xx_it.c.

7.21.2.6 DMA2_Stream0_IRQHandler()

```
void DMA2_Stream0_IRQHandler (
    void )
```

This function handles DMA2 stream0 global interrupt.

Definition at line 195 of file stm32f4xx_it.c.

References hdma_adc1.

7.21.2.7 EXTI0_IRQHandler()

```
void EXTI0_IRQHandler (
    void )
```

This function handles EXTI line0 interrupt.

Definition at line 167 of file stm32f4xx_it.c.

References Start_Button_Input_Pin.

7.21.2.8 HardFault_Handler()

```
void HardFault_Handler (
    void )
```

This function handles Hard fault interrupt.

Definition at line 87 of file stm32f4xx_it.c.

7.21.2.9 MemManage_Handler()

```
void MemManage_Handler (
    void )
```

This function handles Memory management fault.

Definition at line 102 of file stm32f4xx_it.c.

7.21.2.10 NMI_Handler()

```
void NMI_Handler (
    void )
```

This function handles Non maskable interrupt.

Definition at line 72 of file stm32f4xx_it.c.

7.21.2.11 TIM1_UP_TIM10_IRQHandler()

```
void TIM1_UP_TIM10_IRQHandler (
    void )
```

This function handles TIM1 update interrupt and TIM10 global interrupt.

Definition at line 181 of file stm32f4xx_it.c.

References htim1.

7.21.2.12 UsageFault_Handler()

```
void UsageFault_Handler (
    void )
```

This function handles Undefined instruction or illegal state.

Definition at line 132 of file stm32f4xx_it.c.

7.22 Core/Inc/temp_monitoring.h File Reference

```
#include "uvfr_utils.h"
```

Functions

- [uv_status initTempMonitor](#) (void *args)
- void [tempMonitorTask](#) (void *args)

Monitors the temperatures of various points in the tractive system, and activates various cooling systems and such accordingly.

7.22.1 Function Documentation

7.22.1.1 initTempMonitor()

```
uv_status initTempMonitor (
    void * args )
```

Definition at line 12 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\(\)](#).

7.22.1.2 tempMonitorTask()

```
void tempMonitorTask (
    void * args )
```

Monitors the temperatures of various points in the tractive system, and activates various cooling systems and such accordingly.

Atm, this is mostly serving as an example of a task 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 = 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 48 of file temp_monitoring.c.

References [uv_task_info::cmd_data](#), [handleCANbusError\(\)](#), [hcan2](#), [killSelf\(\)](#), [suspendSelf\(\)](#), [uv_task_info::task_period](#), [TxData](#), [TxHeader](#), [TxMailbox](#), [UV_KILL_CMD](#), [UV_SUSPEND_CMD](#), and [uvTaskDelayUntil](#).

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

7.23 Core/Inc/tim.h File Reference

This file contains all the function prototypes for the [tim.c](#) file.

```
#include "main.h"
```

Functions

- void [MX_TIM3_Init](#) (void)

Variables

- TIM_HandleTypeDef [htim3](#)

7.23.1 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.

7.23.2 Function Documentation

7.23.2.1 MX_TIM3_Init()

```
void MX_TIM3_Init (  
    void )
```

Definition at line 30 of file [tim.c](#).

References [Error_Handler\(\)](#), and [htim3](#).

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

7.23.3 Variable Documentation

7.23.3.1 htim3

```
TIM_HandleTypeDef htim3
```

Definition at line 27 of file tim.c.

Referenced by HAL_TIM_PeriodElapsedCallback(), and MX_TIM3_Init().

7.24 Core/Inc/uvfr_global_config.h File Reference

Macros

- `#define UV19_PDU 1`
- `#define ECUMASTER_PMU 0`
- `#define STM32_F407 1`
- `#define STM32_H7xx 0`
- `#define UV_MALLOC_LIMIT ((size_t)1024)`
- `#define USE_OS_MEM_MGMT 0`

7.24.1 Macro Definition Documentation

7.24.1.1 ECUMASTER_PMU

```
#define ECUMASTER_PMU 0
```

Definition at line 13 of file uvfr_global_config.h.

7.24.1.2 STM32_F407

```
#define STM32_F407 1
```

Definition at line 15 of file uvfr_global_config.h.

7.24.1.3 STM32_H7xx

```
#define STM32_H7xx 0
```

Definition at line 16 of file uvfr_global_config.h.

7.24.1.4 USE_OS_MEM_MGMT

```
#define USE_OS_MEM_MGMT 0
```

Definition at line 26 of file uvfr_global_config.h.

7.24.1.5 UV19_PDU

```
#define UV19_PDU 1
```

Definition at line 12 of file uvfr_global_config.h.

7.24.1.6 UV_MALLOC_LIMIT

```
#define UV_MALLOC_LIMIT ((size_t)1024)
```

Definition at line 22 of file uvfr_global_config.h.

7.25 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"
```

Data Structures

- struct [veh_gen_info](#)
- struct [uv_vehicle_settings](#)

Macros

- #define [ENABLE_FLASH_SETTINGS](#) 0

Typedefs

- typedef struct [veh_gen_info](#) [veh_gen_info](#)
- typedef struct [uv_vehicle_settings](#) [uv_vehicle_settings](#)

Functions

- void `nukeSettings` (`uv_vehicle_settings **settings_to_delete`)
- 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.

Variables

- `uv_vehicle_settings * current_vehicle_settings`

7.25.1 Macro Definition Documentation

7.25.1.1 ENABLE_FLASH_SETTINGS

```
#define ENABLE_FLASH_SETTINGS 0
```

Definition at line 21 of file `uvfr_settings.h`.

7.25.2 Typedef Documentation

7.25.2.1 uv_vehicle_settings

```
typedef struct uv_vehicle_settings uv_vehicle_settings
```

7.25.2.2 veh_gen_info

```
typedef struct veh_gen_info veh_gen_info
```

7.25.3 Function Documentation

7.25.3.1 nukeSettings()

```
void nukeSettings (  
    uv_vehicle_settings ** settings_to_delete )
```

Definition at line 51 of file `uvfr_settings.c`.

7.25.3.2 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 64 of file uvfr_settings.c.

References `setupDefaultSettings()`, `UV_ABORTED`, `UV_ERROR`, and `UV_OK`.

Referenced by `uvInit()`.

7.25.4 Variable Documentation

7.25.4.1 current_vehicle_settings

```
uv_vehicle_settings* current_vehicle_settings
```

Definition at line 15 of file uvfr_settings.c.

Referenced by `setupDefaultSettings()`, and `uvInit()`.

7.26 Core/Inc/uvfr_state_engine.h File Reference

```
#include "uvfr_utils.h"
```

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 0b0000000000000011`
- `#define UV_TASK_GENERIC_SVC 0x0001U<<(2)`
- `#define UV_TASK_MANAGER_MASK 0b0000000000000011`
- `#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) (t->task_flags|=UV_TASK_AWAITING_DELETION)`
- `#define uvTaskResetDeletionBit(t) (t->task_flags &=(~UV_TASK_AWAITING_DELETION))`
- `#define uvTaskSetDelayBit(t) (t->task_flags|=UV_TASK_DELAYING)`
- `#define uvTaskResetDelayBit(t) (t->task_flags&=(~UV_TASK_DELAYING))`
- `#define uvTaskIsDelaying(t) ((t->task_flags&UV_TASK_DELAYING)==UV_TASK_DELAYING)`
- `#define uvTaskDelay(x, t)`
State engine aware vTaskDelay wrapper.
- `#define uvTaskDelayUntil(x, lasttim, per)`
State engine aware vTaskDelayUntil wrapper.

Typedefs

- `typedef enum uv_status_t uv_status`
- `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.

Functions

- 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 gnarliness 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](#) * [uvGetTaskById](#) (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 uvDeInitStateEngine](#) ()
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)
Something bad has occurred here now we in trouble.
- 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 *task_name)

Variables

- enum [uv_vehicle_state_t](#) [vehicle_state](#)

7.26.1 Macro Definition Documentation

7.26.1.1 _LONGEST_SC_TIME

```
#define _LONGEST_SC_TIME 300
```

Definition at line 63 of file [uvfr_state_engine.h](#).

7.26.1.2 _SC_DAEMON_PERIOD

```
#define _SC_DAEMON_PERIOD 10
```

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

7.26.1.3 _UV_DEFAULT_TASK_INSTANCES

```
#define _UV_DEFAULT_TASK_INSTANCES 128
```

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

7.26.1.4 `_UV_DEFAULT_TASK_PERIOD`

```
#define _UV_DEFAULT_TASK_PERIOD 100
```

Definition at line 60 of file `uvfr_state_engine.h`.

7.26.1.5 `_UV_DEFAULT_TASK_STACK_SIZE`

```
#define _UV_DEFAULT_TASK_STACK_SIZE 128
```

Definition at line 58 of file `uvfr_state_engine.h`.

7.26.1.6 `_UV_MIN_TASK_PERIOD`

```
#define _UV_MIN_TASK_PERIOD 5
```

Definition at line 61 of file `uvfr_state_engine.h`.

7.26.1.7 `SVC_TASK_MAX_CHECKIN_PERIOD`

```
#define SVC_TASK_MAX_CHECKIN_PERIOD 500
```

Definition at line 66 of file `uvfr_state_engine.h`.

7.26.2 Typedef Documentation

7.26.2.1 `uv_status`

```
typedef enum uv_status_t uv_status
```

Definition at line 51 of file `uvfr_state_engine.h`.

7.26.2.2 `uv_task_id`

```
typedef uint8_t uv_task_id
```

Definition at line 52 of file `uvfr_state_engine.h`.

7.26.2.3 uv_timespan_ms

```
typedef uint32_t uv_timespan_ms
```

Definition at line 70 of file uvfr_state_engine.h.

7.26.3 Function Documentation

7.26.3.1 getSVCTaskID()

```
uv_task_id getSVCTaskID (
    char * tsk_name )
```

7.26.3.2 updateRunningTasks()

```
uv_status updateRunningTasks ( )
```

7.26.3.3 uvGetTaskById()

```
struct uv_task_info* uvGetTaskById (
    uint8_t id )
```

7.26.3.4 uvRegisterTask()

```
uv_status uvRegisterTask ( )
```

7.27 Core/Inc/uvfr_utils.h File Reference

```
#include "uvfr_global_config.h"
#include "main.h"
#include "cmsis_os.h"
#include "adc.h"
#include "can.h"
#include "dma.h"
#include "tim.h"
#include "gpio.h"
#include "spi.h"
#include "FreeRTOS.h"
#include "task.h"
#include "message_buffer.h"
#include "uvfr_settings.h"
#include "uvfr_state_engine.h"
#include "rb_tree.h"
#include "bms.h"
#include "motor_controller.h"
#include "dash.h"
#include "imd.h"
#include "pdu.h"
#include "daq.h"
#include "oled.h"
#include "driving_loop.h"
#include "temp_monitoring.h"
#include "odometer.h"
#include "FreeRTOSConfig.h"
#include "stdint.h"
#include <stdlib.h>
```

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) _BV_16(x)`
- `#define _BV_8(x) ((uint8_t)(0x01U >> x))`
- `#define _BV_16(x) ((uint16_t)(0x01U >> x))`
- `#define _BV_32(x) ((uint32_t)(0x01U >> x))`
- `#define endianSwap(x) endianSwap16(x)`
- `#define endianSwap8(x) x`
- `#define endianSwap16(x) (((x & 0x00FF)<<8) | ((x & 0xFF00)>>8))`
- `#define endianSwap32(x) (((x & 0x000000FF)<<16)|((x & 0x0000FF00)<<8)|((x & 0x00FF0000)>>8)|((x & 0xFF000000)>>16))`
- `#define deserializeSmallE16(x, i) ((x[i])|(x[i+1]<<8))`
- `#define deserializeSmallE32(x, i) ((x[i])|(x[i+1]<<8)|(x[i+2]<<16)|(x[i+3]<<24))`
- `#define deserializeBigE16(x, i) ((x[i]<<8)|(x[i+1]))`
- `#define deserializeBigE32(x, i) ((x[i]<<24)|(x[i+1]<<16)|(x[i+2]<<8)|(x[i+3]))`
- `#define serializeSmallE16(x, d, i) x[i]=d&0x00FF; x[i+1]=(d&0xFF00)>>8`
- `#define serializeSmallE32(x, d, i) x[i]=d&0x000000FF; x[i+1]=(d&0x0000FF00)>>8; x[i+2]=(d&0x00FF0000)>>16; x[i+3]=(d&0xFF000000)>>24`
- `#define serializeBigE16(x, d, i) x[i+1]=d&0x00FF; x[i]=(d&0xFF00)>>8`
- `#define serializeBigE32(x, d, i) x[i+3]=d&0x000000FF; x[i+2]=(d&0x0000FF00)>>8; x[i+1]=(d&0x00FF0000)>>16; x[i]=(d&0xFF000000)>>24`
- `#define setBits(x, msk, data) x=(x&(~msk)|data)`
macro to set bits of an int without touching the ones we dont want to edit
- `#define isPowerOfTwo(x) (x&&(!(x&(x-1))))`
Returns a truthy value if "x" is a power of two.
- `#define safePtrRead(x) (*((x)?x:uvPanic("nullptr_deref",0))`
lil treat to help us avoid the dreaded null pointer dereference
- `#define safePtrWrite(p, x) (*((p)?p:&x)`
- `#define false 0`
- `#define true !false`
- `#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 0x01`
- `#define UV_CAN_CHANNEL_MASK 0b00000110`
- `#define UV_CAN_DYNAMIC_MEM 0b00001000`

Typedefs

- `typedef uint8_t bool`
- `typedef uint8_t uv_task_id`
- `typedef enum uv_task_cmd_e uv_task_cmd`
- `typedef uint8_t uv_ext_device_id`
- `typedef uint32_t uv_timespan_ms`
- `typedef enum uv_status_t uv_status`
This is meant to be a return type from functions that indicates what is actually going on.
- `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_CAN_msg uv_CAN_msg`

Representative of a CAN message.

- 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](#)
- 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](#) }
- enum [uv_driving_mode_t](#) { [normal](#), [accel](#), [econ](#), [limp](#) }
- enum [uv_external_device](#) { [MOTOR_CONTROLLER](#) = 0, [BMS](#) = 1, [IMD](#) = 2, [PDU](#) = 3 }

This is meant to be a return type from functions that indicates what is actually going on.

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 [uvlInit](#) (void *arguments)
: Function that initializes all of the car's stuff.
- void [__uvlInitPanic](#) ()
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](#)

7.27.1 Detailed Description

Author

Byron Oser

7.27.2 Macro Definition Documentation

7.27.2.1 INIT_CHECK_PERIOD

```
#define INIT_CHECK_PERIOD 100
```

Definition at line 146 of file uvfr_utils.h.

7.27.2.2 MAX_INIT_TIME

```
#define MAX_INIT_TIME 2500
```

Definition at line 145 of file uvfr_utils.h.

7.27.2.3 USE_OLED_DEBUG

```
#define USE_OLED_DEBUG 1
```

Definition at line 157 of file uvfr_utils.h.

7.27.2.4 UV_CAN1

```
#define UV_CAN1
```

Definition at line 153 of file uvfr_utils.h.

7.27.2.5 UV_CAN2

```
#define UV_CAN2
```

Definition at line 154 of file uvfr_utils.h.

7.27.2.6 UV_CAN_CHANNEL_MASK

```
#define UV_CAN_CHANNEL_MASK 0b00000110
```

Definition at line 248 of file uvfr_utils.h.

7.27.2.7 UV_CAN_DYNAMIC_MEM

```
#define UV_CAN_DYNAMIC_MEM 0b00001000
```

Definition at line 249 of file uvfr_utils.h.

7.27.2.8 UV_CAN_EXTENDED_ID

```
#define UV_CAN_EXTENDED_ID 0x01
```

Definition at line 247 of file uvfr_utils.h.

7.27.3 Typedef Documentation

7.27.3.1 access_control_info

```
typedef union access_control_info access_control_info
```

7.27.3.2 access_control_type

```
typedef enum access_control_t access_control_type
```

7.27.3.3 bool

```
typedef uint8_t bool
```

Definition at line 134 of file uvfr_utils.h.

7.27.3.4 p_status

```
typedef struct p_status p_status
```

7.27.3.5 uv_CAN_msg

```
typedef struct uv_CAN_msg uv_CAN_msg
```

Representative of a CAN message.

7.27.3.6 uv_ext_device_id

```
typedef uint8_t uv_ext_device_id
```

Definition at line 138 of file uvfr_utils.h.

7.27.3.7 uv_init_struct

```
typedef struct uv_init_struct uv_init_struct
```

contains info relevant to initializing the vehicle

7.27.3.8 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.

7.27.3.9 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.

7.27.3.10 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

7.27.3.11 uv_msg_type

```
typedef enum uv_msg_type_t uv_msg_type
```

Enum dictating the meaning of a generic message.

7.27.3.12 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.

7.27.3.13 uv_task_cmd

```
typedef enum uv_task_cmd_e uv_task_cmd
```

Definition at line 136 of file uvfr_utils.h.

7.27.3.14 uv_task_id

```
typedef uint8_t uv_task_id
```

Definition at line 135 of file uvfr_utils.h.

7.27.3.15 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.

7.27.3.16 uv_timespan_ms

```
typedef uint32_t uv_timespan_ms
```

Definition at line 139 of file uvfr_utils.h.

7.27.4 Enumeration Type Documentation

7.27.4.1 access_control_t

```
enum access_control_t
```

Enumerator

UV_NONE	
UV_DUMB_FLAG	
UV_MUTEX	
UV_BINARY_SEMAPHORE	
UV_SEMAPHORE	

Definition at line 196 of file uvfr_utils.h.

7.27.4.2 uv_driving_mode_t

```
enum uv_driving_mode_t
```

Enumerator

normal	
accel	
econ	
limp	

Definition at line 179 of file uvfr_utils.h.

7.27.4.3 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 189 of file uvfr_utils.h.

7.27.4.4 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 207 of file uvfr_utils.h.

7.27.4.5 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 166 of file uvfr_utils.h.

7.27.5 Function Documentation

7.27.5.1 __uvInitPanic()

```
void __uvInitPanic ( )
```

Low Level Panic, that does not require the full UVFR utils functionality to be operational.

Attention

Calling `_uvInitPanic()` is irreversable 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 263 of file uvfr_utils.c.

Referenced by `uvInit()`, `uvInitStateEngine()`, and `uvSVCTaskManager()`.

7.27.5.2 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.

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 we have initialized the state engine, what we want to do is create the prototypes of all the tasks that will be running.

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 existence 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;
//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);
//MC_init_args->meta_task_handle = osThreadCreate(&MC_init_thread, MC_init_args);
//vTaskResume( MC_init_args->meta_task_handle );
retval =
    xTaskCreate(MC_Startup, "MC_init", 128, MC_init_args, osPriorityAboveNormal, &(MC_init_args->meta_task_handle));
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}

```

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", 128, BMS_init_args, osPriorityAboveNormal, &(BMS_init_args->meta_task_handle));
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}

```

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(imdInit, "BMS_init", 128, IMD_init_args, osPriorityAboveNormal, &(IMD_init_args->meta_task_handle));
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}
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, dum dum
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}
uint16_t ext_devices_status = 0x000F; //Tracks which devices are currently setup

```

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 37 of file `uvfr_utils.c`.

References `__uvInitPanic()`, `BMS_Init()`, `uv_vehicle_settings::bms_settings`, `changeVehicleState()`, `current_↵ vehicle_settings`, `uv_init_task_response::device`, `uv_init_task_response::errmsg`, `uv_vehicle_settings::imd_↵ settings`, `INIT_CHECK_PERIOD`, `uv_init_task_args::init_info_queue`, `init_task_handle`, `initIMD()`, `initPDU()`, `M_↵ AX_INIT_TIME`, `uv_vehicle_settings::mc_settings`, `MC_Startup()`, `uv_init_task_args::meta_task_handle`, `uv_↵ init_task_response::nchar`, `uv_init_task_args::specific_args`, `uv_init_task_response::status`, `UV_OK`, `UV_READY`, `uvInitStateEngine()`, `uvSettingsInit()`, and `uvStartStateMachine()`.

Referenced by `MX_FREERTOS_Init()`.

7.27.5.3 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 `Hard_↵ FaultHandler()`. 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 393 of file `uvfr_utils.c`.

References `UV_ERROR`, `UV_OK`, and `UV_WARNING`.

Referenced by `__uvFreeCritSection()`, `__uvFreeOS()`, and `__uvMallocOS()`.

7.27.6 Variable Documentation

7.27.6.1 global_context

```
uv_internal_params global_context
```

7.28 Core/Inc/uvfr_vehicle_commands.h File Reference

```
#include "uvfr_global_config.h"  
#include "uvfr_utils.h"
```

Macros

- #define [uvOpenSDC](#)(x) _uvOpenSDC_canBased(x)
- #define [uvOpenSDC](#)(x) _uvCloseSDC_canBased(x)
- #define [uvStartFans](#)(x) _uvStartFans_canBased(x)
- #define [uvStopFans](#)(x) _uvStopFans_canBased(x)
- #define [uvStartCoolantPump](#)(x) _uvStartCoolantPump_canBased(x)
- #define [uvStopCoolantPump](#)(x) _uvStopCoolantPump_canBased(x)
- #define [uvHonkHorn](#)(x) _uvHonkHorn_canBased(x)

Functions

- void [uvSecureVehicle](#) ()
Function to put vehicle into safe state.

7.28.1 Macro Definition Documentation

7.28.1.1 uvHonkHorn

```
#define uvHonkHorn(  
    x ) _uvHonkHorn_canBased(x)
```

Definition at line 88 of file uvfr_vehicle_commands.h.

7.28.1.2 uvOpenSDC [1/2]

```
#define uvOpenSDC(  
    x ) _uvOpenSDC_canBased(x)
```

Definition at line 38 of file uvfr_vehicle_commands.h.

7.28.1.3 uvOpenSDC [2/2]

```
#define uvOpenSDC(  
    x ) _uvCloseSDC_canBased(x)
```

Definition at line 38 of file uvfr_vehicle_commands.h.

7.28.1.4 uvStartCoolantPump

```
#define uvStartCoolantPump(  
    x ) _uvStartCoolantPump_canBased(x)
```

Definition at line 68 of file uvfr_vehicle_commands.h.

7.28.1.5 uvStartFans

```
#define uvStartFans(  
    x ) _uvStartFans_canBased(x)
```

Definition at line 48 of file uvfr_vehicle_commands.h.

7.28.1.6 uvStopCoolantPump

```
#define uvStopCoolantPump(  
    x ) _uvStopCoolantPump_canBased(x)
```

Definition at line 78 of file uvfr_vehicle_commands.h.

7.28.1.7 uvStopFans

```
#define uvStopFans(  
    x ) _uvStopFans_canBased(x)
```

Definition at line 58 of file uvfr_vehicle_commands.h.

7.29 Core/Src/adc.c File Reference

This file provides code for the configuration of the ADC instances.

```
#include "adc.h"
```

Functions

- void [MX_ADC1_Init](#) (void)
- void [MX_ADC2_Init](#) (void)
- void [HAL_ADC_MspInit](#) (ADC_HandleTypeDef *adcHandle)
- void [HAL_ADC_MspDeInit](#) (ADC_HandleTypeDef *adcHandle)

Variables

- ADC_HandleTypeDef [hadc1](#)
- ADC_HandleTypeDef [hadc2](#)
- DMA_HandleTypeDef [hdma_adc1](#)

7.29.1 Detailed Description

This file provides code for the configuration of the ADC 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.

7.29.2 Function Documentation

7.29.2.1 HAL_ADC_MspDeInit()

```
void HAL_ADC_MspDeInit (
    ADC_HandleTypeDef * adcHandle )
```

ADC1 GPIO Configuration PA1 ----> ADC1_IN1 PA2 ----> ADC1_IN2 PA3 ----> ADC1_IN3 PA4 ----> ADC1_IN4

ADC2 GPIO Configuration PA5 ----> ADC2_IN5 PA6 ----> ADC2_IN6

Definition at line 236 of file adc.c.

7.29.2.2 HAL_ADC_MspInit()

```
void HAL_ADC_MspInit (
    ADC_HandleTypeDef * adcHandle )
```

ADC1 GPIO Configuration PA1 ----> ADC1_IN1 PA2 ----> ADC1_IN2 PA3 ----> ADC1_IN3 PA4 ----> ADC1_IN4

ADC2 GPIO Configuration PA5 ----> ADC2_IN5 PA6 ----> ADC2_IN6

Definition at line 165 of file adc.c.

References Error_Handler(), and hdma_adc1.

7.29.2.3 MX_ADC1_Init()

```
void MX_ADC1_Init (
    void )
```

Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

Configure the analog watchdog

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 32 of file adc.c.

References Error_Handler(), and hadc1.

Referenced by main().

7.29.2.4 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.

Definition at line 118 of file adc.c.

References Error_Handler(), and hadc2.

Referenced by main().

7.29.3 Variable Documentation

7.29.3.1 hadc1

`ADC_HandleTypeDef hadc1`

Definition at line 27 of file `adc.c`.

Referenced by `HAL_ADC_LevelOutOfWindowCallback()`, and `MX_ADC1_Init()`.

7.29.3.2 hadc2

`ADC_HandleTypeDef hadc2`

Definition at line 28 of file `adc.c`.

Referenced by `HAL_TIM_PeriodElapsedCallback()`, and `MX_ADC2_Init()`.

7.29.3.3 hdma_adc1

`DMA_HandleTypeDef hdma_adc1`

Definition at line 29 of file `adc.c`.

Referenced by `DMA2_Stream0_IRQHandler()`, and `HAL_ADC_MspInit()`.

7.30 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"
```

Functions

- void [BMS_Init](#) (void *args)

7.30.1 Function Documentation

7.30.1.1 BMS_Init()

```
void BMS_Init (
    void * args )
```

Definition at line 11 of file bms.c.

References `BMS`, `uv_init_task_args::init_info_queue`, `uv_init_task_args::meta_task_handle`, and `UV_OK`.

Referenced by `uvInit()`.

7.31 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"
```

Macros

- `#define HAL_CAN_ERROR_INVALID_CALLBACK (0x00400000U)`

Functions

- void `handleCANbusError` (const CAN_HandleTypeDef *hcan, const uint32_t err_to_ignore)
- void `MX_CAN2_Init` (void)
- void `HAL_CAN_MspInit` (CAN_HandleTypeDef *canHandle)
- void `HAL_CAN_MspDeInit` (CAN_HandleTypeDef *canHandle)
- 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)

Variables

- static QueueHandle_t [Tx_msg_queue](#) = NULL
- CAN_HandleTypeDef [hcan2](#)

7.31.1 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.

7.31.2 Macro Definition Documentation

7.31.2.1 HAL_CAN_ERROR_INVALID_CALLBACK

```
#define HAL_CAN_ERROR_INVALID_CALLBACK (0x00400000U)
```

Definition at line 35 of file can.c.

7.31.3 Function Documentation

7.31.3.1 CANbusRxSVCDaemon()

```
void CANbusRxSVCDaemon (  
    void * args )
```

Definition at line 407 of file can.c.

7.31.3.2 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

Definition at line 358 of file `can.c`.

References `uv_CAN_msg::data`, `uv_CAN_msg::dlc`, `uv_CAN_msg::flags`, `hcan2`, `uv_CAN_msg::msg_id`, `Tx_msg←_queue`, `TxHeader`, `TxMailbox`, and `UV_CAN_EXTENDED_ID`.

Referenced by `uvSVCTaskManager()`.

7.31.3.3 HAL_CAN_MspDeInit()

```
void HAL_CAN_MspDeInit (
    CAN_HandleTypeDef * canHandle )
```

CAN2 GPIO Configuration PB5 ----> CAN2_RX PB6 ----> CAN2_TX

Definition at line 235 of file `can.c`.

7.31.3.4 HAL_CAN_MspInit()

```
void HAL_CAN_MspInit (
    CAN_HandleTypeDef * canHandle )
```

CAN2 GPIO Configuration PB5 ----> CAN2_RX PB6 ----> CAN2_TX

Definition at line 197 of file `can.c`.

7.31.3.5 HAL_CAN_RxFifo0MsgPendingCallback()

```
void HAL_CAN_RxFifo0MsgPendingCallback (
    CAN_HandleTypeDef * hcan2 )
```

Definition at line 267 of file `can.c`.

References `Error_Handler()`, `hcan2`, `RxData`, and `RxHeader`.

7.31.3.6 HAL_CAN_RxFifo1MsgPendingCallback()

```
void HAL_CAN_RxFifo1MsgPendingCallback (
    CAN_HandleTypeDef * hcan2 )
```

Definition at line 303 of file can.c.

7.31.3.7 handleCANbusError()

```
void handleCANbusError (
    const CAN_HandleTypeDef * hcan,
    const uint32_t err_to_ignore )
```

Definition at line 40 of file can.c.

References HAL_CAN_ERROR_INVALID_CALLBACK.

Referenced by main(), and tempMonitorTask().

7.31.3.8 MX_CAN2_Init()

```
void MX_CAN2_Init (
    void )
```

Definition at line 119 of file can.c.

References Error_Handler(), hcan2, and TxHeader.

Referenced by main().

7.31.3.9 uvSendCanMSG()

```
uv_status uvSendCanMSG (
    uv_CAN_msg * 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. Check that the CAN Tx daemon is actually active

Definition at line 320 of file can.c.

References CAN_TX_DAEMON_NAME, Tx_msg_queue, UV_ERROR, and UV_OK.

7.31.4 Variable Documentation

7.31.4.1 hcan2

```
CAN_HandleTypeDef hcan2
```

Definition at line 116 of file can.c.

Referenced by CAN2_RX0_IRQHandler(), CAN2_RX1_IRQHandler(), CAN2_TX_IRQHandler(), CANbusTx↵ SvcDaemon(), HAL_CAN_RxFifo0MsgPendingCallback(), IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), MX_CAN2_Init(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable↵ _cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable↵ shutdown_circuit(), PDU_speaker_chirp(), tempMonitorTask(), Update_Batt_Temp(), Update_RPM(), and Update↵ _State_Of_Charge().

7.31.4.2 Tx_msg_queue

```
QueueHandle_t Tx_msg_queue = NULL [static]
```

Definition at line 38 of file can.c.

Referenced by CANbusTxSvcDaemon(), and uvSendCanMSG().

7.32 Core/Src/constants.c File Reference

```
#include "main.h"
```

Variables

- CAN_TxHeaderTypeDef [TxHeader](#)
- CAN_RxHeaderTypeDef [RxHeader](#)
- uint8_t [TxData](#) [8]
- uint32_t [TxMailbox](#)
- uint8_t [RxData](#) [8]

7.32.1 Variable Documentation

7.32.1.1 RxData

```
uint8_t RxData[8]
```

Definition at line 9 of file constants.c.

Referenced by HAL_CAN_RxFifo0MsgPendingCallback().

7.32.1.2 RxHeader

```
CAN_RxHeaderTypeDef RxHeader
```

Definition at line 5 of file constants.c.

Referenced by HAL_CAN_RxFifo0MsgPendingCallback().

7.32.1.3 TxData

```
uint8_t TxData[8]
```

Definition at line 7 of file constants.c.

Referenced by IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), PDU_disable_brake↵_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_↵disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_↵fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), tempMonitor↵Task(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.32.1.4 TxHeader

```
CAN_TxHeaderTypeDef TxHeader
```

Definition at line 4 of file constants.c.

Referenced by CANbusTxSvcDaemon(), IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), MX_CAN2_Init(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_↵disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_↵pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_↵speaker_chirp(), tempMonitorTask(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.32.1.5 TxMailbox

```
uint32_t TxMailbox
```

Definition at line 8 of file constants.c.

Referenced by CANbusTxSvcDaemon(), IMD_Request_Status(), main(), MC_Request_Data(), MC_Send_Data(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), tempMonitorTask(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.33 Core/Src/daq.c File Reference

```
#include "uvfr_utils.h"
#include "daq.h"
```

Macros

- `#define _SRC_UVFR_DAQ`

Functions

- void `deleteParamList` ()
- void `deleteDaqSubTask` ()
- `uv_status` `startDaqSubTasks` ()
- `uv_status` `stopDaqSubTasks` ()
- `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)
- void `daqSubTask` (void *args)

Variables

- void * `param_LUT` [126]

7.33.1 Macro Definition Documentation

7.33.1.1 _SRC_UVFR_DAQ

```
#define _SRC_UVFR_DAQ
```

Definition at line 1 of file daq.c.

7.33.2 Function Documentation

7.33.2.1 daqMasterTask()

```
void daqMasterTask (
    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

```
*/
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 62 of file daq.c.

References `changeVehicleState()`, `uv_task_info::cmd_data`, `killSelf()`, `suspendSelf()`, `uv_task_info::task_period`, `UV_DRIVING`, `UV_ERROR_STATE`, `UV_KILL_CMD`, `UV_READY`, `UV_SUSPEND_CMD`, and `vehicle_state`.

Referenced by `initDaqTask()`.

7.33.2.2 daqSubTask()

```
void daqSubTask (
    void * args )
```

Definition at line 103 of file daq.c.

7.33.2.3 deleteDaqSubTask()

```
void deleteDaqSubTask ( )
```

Definition at line 13 of file daq.c.

7.33.2.4 deleteParamList()

```
void deleteParamList ( )
```

Definition at line 9 of file daq.c.

7.33.2.5 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

Definition at line 30 of file daq.c.

References `_UV_DEFAULT_TASK_STACK_SIZE`, `uv_task_info::active_states`, `daqMasterTask()`, `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_DRIVING`, `UV_ERROR`, `UV_ERROR_STATE`, `UV_LAUNCH_CONTROL`, `UV_OK`, `UV_READY`, and `uvCreateTask()`.

Referenced by `uvInitStateEngine()`.

7.33.2.6 startDaqSubTasks()

```
uv_status startDaqSubTasks ( )
```

Definition at line 17 of file daq.c.

7.33.2.7 stopDaqSubTasks()

```
uv_status stopDaqSubTasks ( )
```

Definition at line 21 of file daq.c.

7.33.3 Variable Documentation

7.33.3.1 param_LUT

```
void* param_LUT[126]
```

Definition at line 7 of file daq.c.

7.34 Core/Src/dash.c File Reference

```
#include "dash.h"
#include "can.h"
#include "main.h"
```

Functions

- void [Update_RPM](#) (int16_t value)
- void [Update_Batt_Temp](#) (uint8_t value)
- void [Update_State_Of_Charge](#) (uint8_t value)

7.34.1 Function Documentation

7.34.1.1 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](#).

7.34.1.2 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\(\)](#).

7.34.1.3 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](#).

7.35 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"
```

Functions

- void [MX_DMA_Init](#) (void)

7.35.1 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.

7.35.2 Function Documentation

7.35.2.1 MX_DMA_Init()

```
void MX_DMA_Init (  
    void )
```

Enable DMA controller clock

Definition at line 39 of file dma.c.

Referenced by main().

7.36 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 "FreeRTOS.h"  
#include "task.h"  
#include "cmsis_os.h"  
#include "driving_loop.h"
```

Functions

- enum [uv_status_t](#) [initDrivingLoop](#) (void *argument)
- void [StartDrivingLoop](#) (void *argument)

Function implementing the ledTask thread.

Variables

- uint16_t [adc1_APPS1](#)
- uint16_t [adc1_APPS2](#)
- uint16_t [adc1_BPS1](#)
- uint16_t [adc1_BPS2](#)

7.36.1 Detailed Description

File containing the meat and potatoes driving loop thread, and all supporting functions.

7.36.2 Function Documentation

7.36.2.1 [initDrivingLoop\(\)](#)

```
enum uv\_status\_t initDrivingLoop (
    void * argument )
```

Definition at line 25 of file [driving_loop.c](#).

References [uv_task_info::active_states](#), [uv_task_info::deletion_states](#), [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_LAUNCH_CONTROL](#), [UV_OK](#), [UV_READY](#), and [uvCreateTask\(\)](#).

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

7.36.2.2 [StartDrivingLoop\(\)](#)

```
void StartDrivingLoop (
    void * argument )
```

Function implementing the ledTask thread.

Parameters

<i>argument</i>	Not used for now. Will have configuration settings later.
-----------------	---

Return values

None	This function is made to be the meat and potatoes of the entire vehicle.
------	--

The first thing we do here is create some local variables here, to cache whatever variables need cached. We will be caching variables that are used very frequently in every single loop iteration, and are not

This line extracts the specific driving loop parameters as specified in the vehicle settings

```
*/
driving_loop_args* dl_params = (driving_loop_args*) params->task_args;
/**
```

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();
/**
```

Brake Plausibility Check

The way that this works is that if the brake pressure is greater than some threshold, and the accelerator pedal position is also greater than some threshold, the thing will register that a brake implausibility has occurred. This is not very cash money.

If this happens, we want to set the torque/speed output to zero. This will only reset itself once the brakes are set to less than a certain threshold. Honestly evil.

Definition at line 68 of file driving_loop.c.

References `adc1_APPS1`, `adc1_APPS2`, `adc1_BPS1`, `adc1_BPS2`, `driving_loop_args::apps_plausibility_check`, `_threshold`, `driving_loop_args::bps_plausibility_check_threshold`, `uv_task_info::cmd_data`, `Implausible`, `killSelf()`, `driving_loop_args::max_apps_offset`, `driving_loop_args::max_apps_value`, `driving_loop_args::max_BPS_value`, `Plausible`, `suspendSelf()`, `uv_task_info::task_args`, `uv_task_info::task_period`, `UV_KILL_CMD`, and `UV_SUSPEND_CMD`.

Referenced by `initDrivingLoop()`.

7.36.3 Variable Documentation

7.36.3.1 adc1_APPS1

```
uint16_t adc1_APPS1
```

Definition at line 64 of file main.c.

Referenced by `HAL_ADC_ConvCpltCallback()`, `HAL_ADC_LevelOutOfWindowCallback()`, `main()`, and `StartDrivingLoop()`.

7.36.3.2 adc1_APPS2

```
uint16_t adc1_APPS2
```

Definition at line 65 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), HAL_ADC_LevelOutOfWindowCallback(), main(), and StartDrivingLoop().

7.36.3.3 adc1_BPS1

```
uint16_t adc1_BPS1
```

Definition at line 66 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), and StartDrivingLoop().

7.36.3.4 adc1_BPS2

```
uint16_t adc1_BPS2
```

Definition at line 67 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), and StartDrivingLoop().

7.37 Core/Src/freertos.c File Reference

```
#include "FreeRTOS.h"
#include "task.h"
#include "main.h"
#include "cmsis_os.h"
#include "uvfr_utils.h"
```

Functions

- void [StartDefaultTask](#) (void const *argument)
Function implementing the defaultTask thread.
- void [MX_FREERTOS_Init](#) (void)
FreeRTOS initialization.
- void [vApplicationGetIdleTaskMemory](#) (StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize)
- void [vApplicationGetTimerTaskMemory](#) (StaticTask_t **ppxTimerTaskTCBBuffer, StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize)
- void [vApplicationTickHook](#) (void)
- void [vApplicationStackOverflowHook](#) (TaskHandle_t xTask, signed char *pcTaskName)
- void [vApplicationMallocFailedHook](#) (void)
- void [vApplicationIdleHook](#) (void)

Variables

- [uv_init_struct init_settings](#)
- [TaskHandle_t init_task_handle](#)
- [osThreadId defaultTaskHandle](#)
- [static StaticTask_t xIdleTaskTCBBuffer](#)
- [static StackType_t xIdleStack \[configMINIMAL_STACK_SIZE\]](#)
- [static StaticTask_t xTimerTaskTCBBuffer](#)
- [static StackType_t xTimerStack \[configTIMER_TASK_STACK_DEPTH\]](#)

7.37.1 Function Documentation

7.37.1.1 MX_FREERTOS_Init()

```
void MX_FREERTOS_Init (
    void )
```

FreeRTOS initialization.

Attention

DONT YOU FUCKING DARE DELETE THESE GOTO STATEMENTS, THEY ARE CRITICAL TO STOP THE OS FROM HANGING ITSELF

Definition at line 159 of file freertos.c.

References [defaultTaskHandle](#), [init_settings](#), [init_task_handle](#), [StartDefaultTask\(\)](#), [uv_init_struct::use_default_↔ settings](#), and [uvInit\(\)](#).

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

7.37.1.2 StartDefaultTask()

```
void StartDefaultTask (
    void const * argument )
```

Function implementing the defaultTask thread.

Attention

DO NOT EVER CALL THIS. IT EXISTS TO STOP A COMPILER ERROR IN THE MX_FREERTOS_INIT FUNCTION

Definition at line 208 of file freertos.c.

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

7.37.1.3 vApplicationGetIdleTaskMemory()

```
void vApplicationGetIdleTaskMemory (
    StaticTask_t ** ppxIdleTaskTCBBuffer,
    StackType_t ** ppxIdleTaskStackBuffer,
    uint32_t * pulIdleTaskStackSize )
```

Definition at line 132 of file freertos.c.

References configMINIMAL_STACK_SIZE, xIdleStack, and xIdleTaskTCBBuffer.

7.37.1.4 vApplicationGetTimerTaskMemory()

```
void vApplicationGetTimerTaskMemory (
    StaticTask_t ** ppxTimerTaskTCBBuffer,
    StackType_t ** ppxTimerTaskStackBuffer,
    uint32_t * pulTimerTaskStackSize )
```

Definition at line 146 of file freertos.c.

References configTIMER_TASK_STACK_DEPTH, xTimerStack, and xTimerTaskTCBBuffer.

7.37.1.5 vApplicationIdleHook()

```
void vApplicationIdleHook (
    void )
```

Definition at line 101 of file freertos.c.

7.37.1.6 vApplicationMallocFailedHook()

```
__weak void vApplicationMallocFailedHook (
    void )
```

Definition at line 108 of file freertos.c.

7.37.1.7 vApplicationStackOverflowHook()

```
__weak void vApplicationStackOverflowHook (
    TaskHandle_t xTask,
    signed char * pcTaskName )
```

Definition at line 89 of file freertos.c.

7.37.1.8 vApplicationTickHook()

```
__weak void vApplicationTickHook (
    void )
```

Definition at line 76 of file freertos.c.

7.37.2 Variable Documentation

7.37.2.1 defaultTaskHandle

```
osThreadId defaultTaskHandle
```

Definition at line 53 of file freertos.c.

Referenced by MX_FREERTOS_Init().

7.37.2.2 init_settings

```
uv_init_struct init_settings
```

File Name : [freertos.c](#) Description : Code for freertos applications

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 at line 48 of file freertos.c.

Referenced by MX_FREERTOS_Init().

7.37.2.3 init_task_handle

```
TaskHandle_t init_task_handle
```

Definition at line 51 of file freertos.c.

Referenced by MX_FREERTOS_Init(), and uvInit().

7.37.2.4 xIdleStack

```
StackType_t xIdleStack[configMINIMAL_STACK_SIZE] [static]
```

Definition at line 130 of file freertos.c.

Referenced by vApplicationGetIdleTaskMemory().

7.37.2.5 xIdleTaskTCBBuffer

```
StaticTask_t xIdleTaskTCBBuffer [static]
```

Definition at line 129 of file freertos.c.

Referenced by vApplicationGetIdleTaskMemory().

7.37.2.6 xTimerStack

```
StackType_t xTimerStack[configTIMER_TASK_STACK_DEPTH] [static]
```

Definition at line 144 of file freertos.c.

Referenced by vApplicationGetTimerTaskMemory().

7.37.2.7 xTimerTaskTCBBuffer

```
StaticTask_t xTimerTaskTCBBuffer [static]
```

Definition at line 143 of file freertos.c.

Referenced by vApplicationGetTimerTaskMemory().

7.38 Core/Src/gpio.c File Reference

This file provides code for the configuration of all used GPIO pins.

```
#include "gpio.h"
```

Functions

- void [MX_GPIO_Init](#) (void)

7.38.1 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.

7.38.2 Function Documentation

7.38.2.1 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 Blue_LED_Pin, Orange_LED_Pin, Red_LED_Pin, Start_Button_Input_GPIO_Port, and Start_Button↵_Input_Pin.

Referenced by main().

7.39 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"
```

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

7.39.1 Function Documentation

7.39.1.1 IMD_Check_Battery_Voltage()

```
void IMD_Check_Battery_Voltage (
    uint8_t Data[] )
```

Definition at line 351 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.2 IMD_Check_Error_Flags()

```
void IMD_Check_Error_Flags (
    uint8_t Data[] )
```

Definition at line 257 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().

7.39.1.3 IMD_Check_Isolation_Capacitances()

```
void IMD_Check_Isolation_Capacitances (
    uint8_t Data[] )
```

Definition at line 337 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.4 IMD_Check_Isolation_Resistances()

```
void IMD_Check_Isolation_Resistances (
    uint8_t Data[] )
```

Definition at line 312 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.39.1.5 IMD_Check_Isolation_State()

```
void IMD_Check_Isolation_State (
    uint8_t Data[] )
```

Definition at line 296 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.39.1.6 IMD_Check_Max_Battery_Working_Voltage()

```
void IMD_Check_Max_Battery_Working_Voltage (
    uint8_t Data[] )
```

Definition at line 388 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.7 IMD_Check_Part_Name()

```
void IMD_Check_Part_Name (
    uint8_t Data[] )
```

Definition at line 401 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().

7.39.1.8 IMD_Check_Safety_Touch_Current()

```
void IMD_Check_Safety_Touch_Current (
    uint8_t Data[] )
```

Definition at line 376 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.9 IMD_Check_Safety_Touch_Energy()

```
void IMD_Check_Safety_Touch_Energy (
    uint8_t Data[] )
```

Definition at line 369 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.10 IMD_Check_Serial_Number()

```
void IMD_Check_Serial_Number (
    uint8_t Data[] )
```

Definition at line 483 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().

7.39.1.11 IMD_Check_Status_Bits()

```
void IMD_Check_Status_Bits (
    uint8_t Data )
```

Definition at line 213 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().

7.39.1.12 IMD_Check_Temperature()

```
void IMD_Check_Temperature (
    uint8_t Data[] )
```

Definition at line 358 of file imd.c.

References IMD_Temperature.

Referenced by IMD_Parse_Message().

7.39.1.13 IMD_Check_Uptime()

```
void IMD_Check_Uptime (
    uint8_t Data[] )
```

Definition at line 524 of file imd.c.

7.39.1.14 IMD_Check_Version()

```
void IMD_Check_Version (
    uint8_t Data[] )
```

Definition at line 443 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()`.

7.39.1.15 IMD_Check_Voltages_Vp_and_Vn()

```
void IMD_Check_Voltages_Vp_and_Vn (
    uint8_t Data[] )
```

Definition at line 344 of file imd.c.

Referenced by `IMD_Parse_Message()`.

7.39.1.16 IMD_Parse_Message()

```
void IMD_Parse_Message (
    int DLC,
    uint8_t Data[] )
```

Definition at line 68 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`.

7.39.1.17 IMD_Request_Status()

```
void IMD_Request_Status (
    uint8_t Status )
```

Definition at line 180 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()`.

7.39.1.18 IMD_Startup()

```
void IMD_Startup ( )
```

Definition at line 528 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`.

7.39.1.19 initIMD()

```
void initIMD (
    void * args )
```

Definition at line 554 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()`.

7.39.2 Variable Documentation

7.39.2.1 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()`.

7.39.2.2 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().

7.39.2.3 IMD_Expected_Serial_Number

```
const uint32_t IMD_Expected_Serial_Number[4]
```

Initial value:

```
= {0xB8DD9AF9,                                0x6094F48B,  
                                0x1F1C3794,  
                                0xFCF9A95B}
```

Definition at line 46 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.4 IMD_Expected_Version

```
const uint32_t IMD_Expected_Version[3]
```

Definition at line 36 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.5 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().

7.39.2.6 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().

7.39.2.7 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().

7.39.2.8 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().

7.39.2.9 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().

7.39.2.10 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().

7.39.2.11 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().

7.39.2.12 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()`.

7.39.2.13 IMD_Read_Version

```
uint32_t IMD_Read_Version[3]
```

Definition at line 35 of file imd.c.

Referenced by `IMD_Check_Version()`.

7.39.2.14 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()`.

7.39.2.15 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()`.

7.39.2.16 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()`.

7.39.2.17 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()`.

7.39.2.18 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()`.

7.39.2.19 IMD_status_bits

```
uint8_t IMD_status_bits = 0
```

Definition at line 19 of file imd.c.

7.39.2.20 IMD_Temperature

```
int32_t IMD_Temperature
```

Definition at line 57 of file imd.c.

Referenced by `IMD_Check_Temperature()`.

7.39.2.21 IMD_Version_0_Set

```
uint8_t IMD_Version_0_Set = 0
```

Definition at line 38 of file imd.c.

Referenced by `IMD_Check_Version()`.

7.39.2.22 IMD_Version_1_Set

```
uint8_t IMD_Version_1_Set = 0
```

Definition at line 39 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.23 IMD_Version_2_Set

```
uint8_t IMD_Version_2_Set = 0
```

Definition at line 40 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.24 IMD_Version_Set

```
uint8_t IMD_Version_Set = 0
```

Definition at line 41 of file imd.c.

Referenced by IMD_Check_Version().

7.40 Core/Src/main.c File Reference

: Main program body

```
#include "main.h"
#include "cmsis_os.h"
#include "adc.h"
#include "can.h"
#include "dma.h"
#include "spi.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"
```

Macros

- #define [DEBUG_CAN_IN_MAIN](#) 0

Functions

- void [SystemClock_Config](#) (void)
System Clock Configuration.
- void [MX_FREERTOS_Init](#) (void)
FreeRTOS initialization.
- 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](#) [[ADC1_BUF_LEN](#)]
- uint16_t [adc1_APPPS1](#)
- uint16_t [adc1_APPPS2](#)
- uint16_t [adc1_BPS1](#)
- uint16_t [adc1_BPS2](#)
- volatile uint32_t [adc_buf2](#) [[ADC2_BUF_LEN](#)]
- uint16_t [adc2_CoolantTemp](#)
- uint16_t [adc2_CoolantFlow](#)

7.40.1 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.

7.40.2 Macro Definition Documentation

7.40.2.1 DEBUG_CAN_IN_MAIN

```
#define DEBUG_CAN_IN_MAIN 0
```

Definition at line 51 of file main.c.

7.40.3 Function Documentation

7.40.3.1 Error_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

None	
------	--

Definition at line 378 of file main.c.

Referenced by HAL_ADC_MspInit(), HAL_CAN_RxFifo0MsgPendingCallback(), IMD_Parse_Message(), IMD_Request_Status(), MC_Parse_Message(), MC_Request_Data(), MC_Send_Data(), MX_ADC1_Init(), MX_ADC2_Init(), MX_CAN2_Init(), MX_SPI1_Init(), MX_TIM3_Init(), PDU_disable_brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PDU_disable_shutdown_circuit(), PDU_enable_brake_light(), PDU_enable_coolant_pump(), PDU_enable_cooling_fans(), PDU_enable_motor_controller(), PDU_enable_shutdown_circuit(), PDU_speaker_chirp(), SystemClock_Config(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

7.40.3.2 HAL_ADC_ConvCpltCallback()

```
void HAL_ADC_ConvCpltCallback (
    ADC_HandleTypeDef * hadc )
```

Definition at line 275 of file main.c.

References adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, ADC1_SAMPLES, adc2_CoolantFlow, adc2_CoolantTemp, adc_buf1, and adc_buf2.

7.40.3.3 HAL_ADC_LevelOutOfWindowCallback()

```
void HAL_ADC_LevelOutOfWindowCallback (
    ADC_HandleTypeDef * hadc )
```

Definition at line 330 of file main.c.

References adc1_APPS1, adc1_APPS2, hadc1, Red_LED_GPIO_Port, and Red_LED_Pin.

7.40.3.4 HAL_GPIO_EXTI_Callback()

```
void HAL_GPIO_EXTI_Callback (
    uint16_t GPIO_Pin )
```

Definition at line 321 of file main.c.

7.40.3.5 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 354 of file main.c.

References ADC2_BUF_LEN, adc_buf2, hadc2, and htim3.

7.40.3.6 main()

```
int main (
    void )
```

The application entry point.

Return values

<i>int</i>	
------------	--

Definition at line 97 of file main.c.

References adc1_APP1, adc1_APP2, handleCANbusError(), hcan2, MX_ADC1_Init(), MX_ADC2_Init(), MX_↵

CAN2_Init(), MX_DMA_Init(), MX_FREERTOS_Init(), MX_GPIO_Init(), MX_SPI1_Init(), MX_TIM3_Init(), SystemClock_Config(), TxData, TxHeader, TxMailbox, and Update_RPM().

7.40.3.7 MX_FREERTOS_Init()

```
void MX_FREERTOS_Init (
    void )
```

FreeRTOS initialization.

Attention

DONT YOU FUCKING DARE DELETE THESE GOTO STATEMENTS, THEY ARE CRITICAL TO STOP THE OS FROM HANGING ITSELF

Definition at line 159 of file freertos.c.

References defaultTaskHandle, init_settings, init_task_handle, StartDefaultTask(), uv_init_struct::use_default_settings, and uvInit().

Referenced by main().

7.40.3.8 SystemClock_Config()

```
void SystemClock_Config (
    void )
```

System Clock Configuration.

Return values

None	
------	--

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 217 of file main.c.

References Error_Handler().

Referenced by main().

7.40.4 Variable Documentation

7.40.4.1 adc1_APPS1

```
uint16_t adc1_APPS1
```

Definition at line 64 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), HAL_ADC_LevelOutOfWindowCallback(), main(), and StartDrivingLoop().

7.40.4.2 adc1_APPS2

```
uint16_t adc1_APPS2
```

Definition at line 65 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), HAL_ADC_LevelOutOfWindowCallback(), main(), and StartDrivingLoop().

7.40.4.3 adc1_BPS1

```
uint16_t adc1_BPS1
```

Definition at line 66 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), and StartDrivingLoop().

7.40.4.4 adc1_BPS2

```
uint16_t adc1_BPS2
```

Definition at line 67 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback(), and StartDrivingLoop().

7.40.4.5 adc2_CoolantFlow

```
uint16_t adc2_CoolantFlow
```

Definition at line 71 of file main.c.

Referenced by HAL_ADC_ConvCpltCallback().

7.40.4.6 `adc2_CoolantTemp`

```
uint16_t adc2_CoolantTemp
```

Definition at line 70 of file `main.c`.

Referenced by `HAL_ADC_ConvCpltCallback()`.

7.40.4.7 `adc_buf1`

```
volatile uint32_t adc_buf1[ADC1_BUF_LEN]
```

Definition at line 62 of file `main.c`.

Referenced by `HAL_ADC_ConvCpltCallback()`.

7.40.4.8 `adc_buf2`

```
volatile uint32_t adc_buf2[ADC2_BUF_LEN]
```

Definition at line 69 of file `main.c`.

Referenced by `HAL_ADC_ConvCpltCallback()`, and `HAL_TIM_PeriodElapsedCallback()`.

7.41 `Core/Src/motor_controller.c` File Reference

```
#include "motor_controller.h"
#include "can.h"
#include "main.h"
#include "constants.h"
#include "pdu.h"
```

Functions

- void [MC_Parse_Message](#) (int DLC, uint8_t Data[])
- void [MC_Request_Data](#) (uint8_t RegID)
- void [MC_Send_Data](#) (uint8_t RegID, uint8_t data_to_send[], uint8_t size)
- void [MC_Torque_Control](#) (int todo)
- void [MC_Check_Error_Warning](#) (uint8_t Data[])
- void [MC_Validate](#) ()
- void [MC_Check_Serial_Number](#) (uint8_t Data[])
- void [MC_Check_Firmware](#) (uint8_t Data[])
- void [MC_Startup](#) (void *args)

Variables

- const uint32_t [MC_Expected_Serial_Number](#) = 0x627E7A01
- const uint16_t [MC_Expected_FW_Version](#) = 0xDC01
- const uint32_t [max_motor_speed](#) = 3277
- uint8_t [desired_motor_speed](#) [2]

7.41.1 Function Documentation

7.41.1.1 MC_Check_Error_Warning()

```
void MC_Check_Error_Warning (
    uint8_t Data[] )
```

Definition at line 122 of file motor_controller.c.

References [AC_current_offset_fault](#), [ADC_measurement_problem](#), [ADC_sequencer_problem](#), [auxiliary_voltage_min_limit](#), [bleed_resistor_overload](#), [bleeder_resistor_warning](#), [CAN_timeout_error](#), [check_ecode_ID](#), [critical_A_C_current](#), [ecode_timeout_error](#), [eprom_read_error](#), [feedback_signal_error](#), [feedback_signal_problem](#), [hardware_fault](#), [IGBT_temp_max_limit](#), [IGBT_temperature_warning](#), [internal_hardware_voltage_problem](#), [mains_voltage_max_limit](#), [mains_voltage_min_limit](#), [motor_temp_max_limit](#), [motor_temperature_warning](#), [parameter_conflict_detected](#), [race_away_detected](#), [rotate_field_enable_not_present_norun](#), [rotate_field_enable_not_present_run](#), [special_CPU_fault](#), [speed_actual_resolution_limit](#), [tripzone_glitch_detected](#), [Vout_saturation_max_limit](#), [warning_5](#), [warning_9](#), and [watchdog_reset](#).

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

7.41.1.2 MC_Check_Firmware()

```
void MC_Check_Firmware (
    uint8_t Data[] )
```

Definition at line 256 of file motor_controller.c.

7.41.1.3 MC_Check_Serial_Number()

```
void MC_Check_Serial_Number (
    uint8_t Data[] )
```

Definition at line 252 of file motor_controller.c.

7.41.1.4 MC_Parse_Message()

```
void MC_Parse_Message (
    int DLC,
    uint8_t Data[] )
```

Definition at line 26 of file motor_controller.c.

References [Error_Handler\(\)](#), [MC_Check_Error_Warning\(\)](#), and [motor_controller_errors_warnings](#).

7.41.1.5 MC_Request_Data()

```
void MC_Request_Data (
    uint8_t RegID )
```

Definition at line 47 of file motor_controller.c.

References [Error_Handler\(\)](#), [hcan2](#), [MC_CAN_ID_Tx](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

7.41.1.6 MC_Send_Data()

```
void MC_Send_Data (
    uint8_t RegID,
    uint8_t data_to_send[],
    uint8_t size )
```

Definition at line 69 of file motor_controller.c.

References [Error_Handler\(\)](#), [hcan2](#), [MC_CAN_ID_Tx](#), [TxData](#), [TxHeader](#), and [TxMailbox](#).

7.41.1.7 MC_Startup()

```
void MC_Startup (
    void * args )
```

Definition at line 260 of file motor_controller.c.

References [uv_init_task_args::init_info_queue](#), [uv_init_task_args::meta_task_handle](#), [MOTOR_CONTROLLER](#), [uv_init_task_args::specific_args](#), and [UV_OK](#).

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

7.41.1.8 MC_Torque_Control()

```
void MC_Torque_Control (
    int todo )
```

Definition at line 102 of file motor_controller.c.

7.41.1.9 MC_Validate()

```
void MC_Validate ( )
```

Definition at line 248 of file motor_controller.c.

7.41.2 Variable Documentation

7.41.2.1 desired_motor_speed

```
uint8_t desired_motor_speed[2]
```

Definition at line 22 of file motor_controller.c.

7.41.2.2 max_motor_speed

```
const uint32_t max_motor_speed = 3277
```

Definition at line 21 of file motor_controller.c.

7.41.2.3 MC_Expected_FW_Version

```
const uint16_t MC_Expected_FW_Version = 0xDC01
```

Definition at line 18 of file motor_controller.c.

7.41.2.4 MC_Expected_Serial_Number

```
const uint32_t MC_Expected_Serial_Number = 0x627E7A01
```

Definition at line 17 of file motor_controller.c.

7.42 Core/Src/odometer.c File Reference

```
#include "uvfr_utils.h"
```

Functions

- [uv_status initOdometer](#) (void *args)
- void [odometerTask](#) (void *args)
, gotta know what the distance travelled is fam

7.42.1 Function Documentation

7.42.1.1 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\(\)](#).

7.42.1.2 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\(\)](#).

7.43 Core/Src/oled.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "oled.h"
#include "main.h"
#include "uvfr_utils.h"
```

7.44 Core/Src/pdu.c File Reference

```
#include "pdu.h"
#include "can.h"
#include "main.h"
#include "constants.h"
```

Functions

- void [PDU_speaker_chirp](#) ()
- void [PDU_enable_brake_light](#) ()
- void [PDU_disable_brake_light](#) ()
- void [PDU_enable_motor_controller](#) ()
- void [PDU_disable_motor_controller](#) ()
- void [PDU_enable_shutdown_circuit](#) ()
- void [PDU_disable_shutdown_circuit](#) ()
- void [PDU_enable_cooling_fans](#) ()
- void [PDU_disable_cooling_fans](#) ()
- void [PDU_enable_coolant_pump](#) ()
- void [PDU_disable_coolant_pump](#) ()
- void [initPDU](#) (void *args)

7.44.1 Function Documentation

7.44.1.1 [initPDU\(\)](#)

```
void initPDU (
    void * args )
```

Definition at line 183 of file pdu.c.

References [uv_init_task_args::init_info_queue](#), [uv_init_task_args::meta_task_handle](#), [PDU](#), and [UV_OK](#).

Referenced by [uvInit](#)().

7.44.1.2 PDU_disable_brake_light()

```
void PDU_disable_brake_light ( )
```

Definition at line 48 of file pdu.c.

References `disable_brake_light_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.3 PDU_disable_coolant_pump()

```
void PDU_disable_coolant_pump ( )
```

Definition at line 170 of file pdu.c.

References `disable_coolant_pump_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.4 PDU_disable_cooling_fans()

```
void PDU_disable_cooling_fans ( )
```

Definition at line 136 of file pdu.c.

References `disable_left_cooling_fan_msg`, `disable_right_cooling_fan_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.5 PDU_disable_motor_controller()

```
void PDU_disable_motor_controller ( )
```

Definition at line 74 of file pdu.c.

References `disable_motor_controller_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.6 PDU_disable_shutdown_circuit()

```
void PDU_disable_shutdown_circuit ( )
```

Definition at line 100 of file pdu.c.

References `disable_shutdown_circuit_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.7 PDU_enable_brake_light()

```
void PDU_enable_brake_light ( )
```

Definition at line 34 of file pdu.c.

References `enable_brake_light_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.8 PDU_enable_coolant_pump()

```
void PDU_enable_coolant_pump ( )
```

Definition at line 158 of file pdu.c.

References `enable_coolant_pump_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.9 PDU_enable_cooling_fans()

```
void PDU_enable_cooling_fans ( )
```

Definition at line 115 of file pdu.c.

References `enable_left_cooling_fan_msg`, `enable_right_cooling_fan_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.10 PDU_enable_motor_controller()

```
void PDU_enable_motor_controller ( )
```

Definition at line 62 of file pdu.c.

References `enable_motor_controller_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.11 PDU_enable_shutdown_circuit()

```
void PDU_enable_shutdown_circuit ( )
```

Definition at line 87 of file pdu.c.

References `enable_shutdown_circuit_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.44.1.12 PDU_speaker_chirp()

```
void PDU_speaker_chirp ( )
```

Definition at line 11 of file pdu.c.

References `disable_speaker_msg`, `enable_speaker_msg`, `Error_Handler()`, `hcan2`, `PDU_CAN_ID_Tx`, `TxData`, `TxHeader`, and `TxMailbox`.

7.45 Core/Src/rb_tree.c File Reference

```
#include "rb_tree.h"
#include <stdio.h>
#include <stdlib.h>
#include "uvfr_utils.h"
```

Functions

- static void `insertRepair` (`rbtree` *rbt, `rbnode` *current)
- static void `deleteRepair` (`rbtree` *rbt, `rbnode` *current)
- static void `rotateLeft` (`rbtree` *, `rbnode` *)
- static void `rotateRight` (`rbtree` *, `rbnode` *)
- static int `checkOrder` (`rbtree` *rbt, `rbnode` *n, void *min, void *max)
- static int `checkBlackHeight` (`rbtree` *rbt, `rbnode` *node)
- static void `print` (`rbtree` *rbt, `rbnode` *node, void(*print_func)(void *), int depth, char *label)
- static void `destroyAllNodes` (`rbtree` *rbt, `rbnode` *node)
- `rbtree` * `rbCreate` (int(*compare)(const void *, const void *), void(*destroy)(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 `rb_apply` (`rbtree` *rbt, `rbnode` *node, int(*func)(void *, void *), void *cookie, enum `rbtraversal` order)
- `rbnode` * `rbInsert` (`rbtree` *rbt, void *data)
- void * `rbDelete` (`rbtree` *rbt, `rbnode` *node, int keep)
- int `rbCheckOrder` (`rbtree` *rbt, void *min, void *max)
- int `rbCheckBlackHeight` (`rbtree` *rbt)
- void `rbPrint` (`rbtree` *rbt, void(*print_func)(void *))

7.45.1 Function Documentation

7.45.1.1 checkBlackHeight()

```
int checkBlackHeight (
    rbtree * rbt,
    rbnode * node ) [static]
```

Definition at line 562 of file rb_tree.c.

References BLACK, rbnode::color, rbnode::left, rbnode::parent, RB_NIL, RED, and rbnode::right.

Referenced by rbCheckBlackHeight().

7.45.1.2 checkOrder()

```
int checkOrder (
    rbtree * rbt,
    rbnode * n,
    void * min,
    void * max ) [static]
```

Definition at line 533 of file rb_tree.c.

References rbtree::compare, rbnode::data, rbnode::left, RB_NIL, and rbnode::right.

Referenced by rbCheckOrder().

7.45.1.3 deleteRepair()

```
void deleteRepair (
    rbtree * rbt,
    rbnode * current ) [static]
```

Definition at line 434 of file rb_tree.c.

References BLACK, rbnode::color, rbnode::left, rbnode::parent, RB_FIRST, RED, rbnode::right, rotateLeft(), and rotateRight().

Referenced by rbDelete().

7.45.1.4 destroyAllNodes()

```
void destroyAllNodes (
    rbtree * rbt,
    rbnode * node ) [static]
```

Definition at line 629 of file rb_tree.c.

References rbtree::count, rbnode::data, rbtree::destroy, rbnode::left, rbnode::parent, RB_NIL, and rbnode::right.

Referenced by rbDestroy().

7.45.1.5 insertRepair()

```
void insertRepair (
    rbtree * rbt,
    rbnode * current ) [static]
```

Definition at line 277 of file `rb_tree.c`.

References `BLACK`, `rbnode::color`, `rbnode::left`, `rbnode::parent`, `RED`, `rbnode::right`, `rotateLeft()`, and `rotateRight()`.

Referenced by `rbInsert()`.

7.45.1.6 print()

```
void print (
    rbtree * rbt,
    rbnode * node,
    void(*) (void *) print_func,
    int depth,
    char * label ) [static]
```

Definition at line 597 of file `rb_tree.c`.

References `rbnode::color`, `rbnode::data`, `rbnode::left`, `RB_NIL`, `RED`, and `rbnode::right`.

Referenced by `rbPrint()`.

7.45.1.7 rb_apply()

```
int rb_apply (
    rbtree * rbt,
    rbnode * node,
    int(*) (void *, void *) func,
    void * cookie,
    enum rbtraversal order )
```

Definition at line 114 of file `rb_tree.c`.

References `rbnode::data`, `INORDER`, `rbnode::left`, `POSTORDER`, `PREORDER`, `RB_NIL`, and `rbnode::right`.

7.45.1.8 rbCheckBlackHeight()

```
int rbCheckBlackHeight (
    rbtree * rbt )
```

Definition at line 551 of file `rb_tree.c`.

References `checkBlackHeight()`, `RB_FIRST`, `RB_NIL`, `RB_ROOT`, and `RED`.

Referenced by `rbPrint()`.

7.45.1.9 rbCheckOrder()

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

Definition at line 525 of file rb_tree.c.

References `checkOrder()`, and `RB_FIRST`.

7.45.1.10 rbCreate()

```
rbtree* rbCreate (
    int(*) (const void *, const void *) compare_func,
    void(*) (void *) destroy_func )
```

Create and initialize a binary search tree.

Definition at line 26 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`.

7.45.1.11 rbDelete()

```
void* rbDelete (
    rbtree * rbt,
    rbnode * node,
    int keep )
```

Definition at line 344 of file rb_tree.c.

References `BLACK`, `rbnode::color`, `rbtree::count`, `rbnode::data`, `deleteRepair()`, `rbtree::destroy`, `rbnode::left`, `rbtree::min`, `rbnode::parent`, `RB_FIRST`, `RB_NIL`, `rbSuccessor()`, `RED`, and `rbnode::right`.

7.45.1.12 rbDestroy()

```
void rbDestroy (
    rbtree * rbt )
```

Destroy the tree, and de-allocate it's elements.

Definition at line 59 of file rb_tree.c.

References `destroyAllNodes()`, and `RB_FIRST`.

7.45.1.13 rbFind()

```
rbnode* rbFind (
    rbtree * rbt,
    void * data )
```

Find a node of the tree based off the data you provide the tree.

Definition at line 69 of file rb_tree.c.

References `rbtree::compare`, `rbnode::data`, `rbnode::left`, `RB_FIRST`, `RB_NIL`, and `rbnode::right`.

7.45.1.14 rbInsert()

```
rbnode* rbInsert (
    rbtree * rbt,
    void * data )
```

Definition at line 191 of file rb_tree.c.

References `BLACK`, `rbnode::color`, `rbtree::compare`, `rbtree::count`, `rbnode::data`, `rbtree::destroy`, `insertRepair()`, `rbnode::left`, `rbtree::min`, `rbnode::parent`, `RB_FIRST`, `RB_MIN`, `RB_NIL`, `RB_ROOT`, `RED`, and `rbnode::right`.

7.45.1.15 rbPrint()

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

Definition at line 587 of file rb_tree.c.

References `print()`, `RB_FIRST`, and `rbCheckBlackHeight()`.

7.45.1.16 rbSuccessor()

```
rbnode* rbSuccessor (
    rbtree * rbt,
    rbnode * node )
```

Definition at line 90 of file rb_tree.c.

References `rbnode::left`, `rbnode::parent`, `RB_NIL`, `RB_ROOT`, and `rbnode::right`.

Referenced by `rbDelete()`.

7.45.1.17 rotateLeft()

```
void rotateLeft (
    rbtree * rbt,
    rbnode * x ) [static]
```

Definition at line 137 of file rb_tree.c.

References `rbnode::left`, `rbnode::parent`, `RB_NIL`, and `rbnode::right`.

Referenced by `deleteRepair()`, and `insertRepair()`.

7.45.1.18 rotateRight()

```
void rotateRight (
    rbtree * rbt,
    rbnode * x ) [static]
```

Definition at line 163 of file rb_tree.c.

References `rbnode::left`, `rbnode::parent`, `RB_NIL`, and `rbnode::right`.

Referenced by `deleteRepair()`, and `insertRepair()`.

7.46 Core/Src/spi.c File Reference

This file provides code for the configuration of the SPI instances.

```
#include "spi.h"
```

Functions

- void `MX_SPI1_Init` (void)
- void `HAL_SPI_MspInit` (SPI_HandleTypeDef *spiHandle)
- void `HAL_SPI_MspDeInit` (SPI_HandleTypeDef *spiHandle)

Variables

- SPI_HandleTypeDef `hspi1`

7.46.1 Detailed Description

This file provides code for the configuration of the SPI instances.

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.

7.46.2 Function Documentation

7.46.2.1 HAL_SPI_MspDeInit()

```
void HAL_SPI_MspDeInit (
    SPI_HandleTypeDef * spiHandle )
```

SPI1 GPIO Configuration PA7 ----> SPI1_MOSI PB3 ----> SPI1_SCK PB4 ----> SPI1_MISO

Definition at line 101 of file spi.c.

7.46.2.2 HAL_SPI_MspInit()

```
void HAL_SPI_MspInit (
    SPI_HandleTypeDef * spiHandle )
```

SPI1 GPIO Configuration PA7 ----> SPI1_MOSI PB3 ----> SPI1_SCK PB4 ----> SPI1_MISO

Definition at line 62 of file spi.c.

7.46.2.3 MX_SPI1_Init()

```
void MX_SPI1_Init (
    void )
```

Definition at line 30 of file spi.c.

References `Error_Handler()`, and `hspi1`.

Referenced by `main()`.

7.46.3 Variable Documentation

7.46.3.1 hspi1

`SPI_HandleTypeDef hspi1`

Definition at line 27 of file spi.c.

Referenced by `MX_SPI1_Init()`.

7.47 Core/Src/stm32f4xx_hal_msp.c File Reference

This file provides code for the MSP Initialization and de-Initialization codes.

```
#include "main.h"
```

Functions

- void [HAL_MspInit](#) (void)

7.47.1 Detailed Description

This file provides code for the MSP Initialization and de-Initialization codes.

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.

7.47.2 Function Documentation

7.47.2.1 HAL_MspInit()

```
void HAL_MspInit (  
    void )
```

Initializes the Global MSP.

Definition at line 64 of file stm32f4xx_hal_msp.c.

7.48 Core/Src/stm32f4xx_hal_timebase_tim.c File Reference

HAL time base based on the hardware TIM.

```
#include "stm32f4xx_hal.h"
#include "stm32f4xx_hal_tim.h"
```

Functions

- HAL_StatusTypeDef [HAL_InitTick](#) (uint32_t TickPriority)
This function configures the TIM1 as a time base source. The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.
- void [HAL_SuspendTick](#) (void)
Suspend Tick increment.
- void [HAL_ResumeTick](#) (void)
Resume Tick increment.

Variables

- TIM_HandleTypeDef [htim1](#)

7.48.1 Detailed Description

HAL time base based on the hardware TIM.

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.

7.48.2 Function Documentation

7.48.2.1 HAL_InitTick()

```
HAL_StatusTypeDef HAL_InitTick (
    uint32_t TickPriority )
```

This function configures the TIM1 as a time base source. The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.

Note

This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().

Parameters

<i>TickPriority</i>	Tick interrupt priority.
---------------------	--------------------------

Return values

<i>HAL</i>	status
------------	--------

Definition at line 41 of file stm32f4xx_hal_timebase_tim.c.

References htim1.

7.48.2.2 HAL_ResumeTick()

```
void HAL_ResumeTick (
    void )
```

Resume Tick increment.

Note

Enable the tick increment by Enabling TIM1 update interrupt.

Definition at line 117 of file stm32f4xx_hal_timebase_tim.c.

References htim1.

7.48.2.3 HAL_SuspendTick()

```
void HAL_SuspendTick (
    void )
```

Suspend Tick increment.

Note

Disable the tick increment by disabling TIM1 update interrupt.

Definition at line 107 of file stm32f4xx_hal_timebase_tim.c.

References htim1.

7.48.3 Variable Documentation

7.48.3.1 htim1

TIM_HandleTypeDef htim1

Definition at line 28 of file stm32f4xx_hal_timebase_tim.c.

Referenced by HAL_InitTick(), HAL_ResumeTick(), HAL_SuspendTick(), and TIM1_UP_TIM10_IRQHandler().

7.49 Core/Src/stm32f4xx_it.c File Reference

Interrupt Service Routines.

```
#include "main.h"
#include "stm32f4xx_it.h"
```

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 [DebugMon_Handler](#) (void)
This function handles Debug monitor.
- void [EXTI0_IRQHandler](#) (void)
This function handles EXTI line0 interrupt.
- void [TIM1_UP_TIM10_IRQHandler](#) (void)
This function handles TIM1 update interrupt and TIM10 global interrupt.
- void [DMA2_Stream0_IRQHandler](#) (void)
This function handles DMA2 stream0 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.

Variables

- DMA_HandleTypeDef [hdma_adc1](#)
- CAN_HandleTypeDef [hcan2](#)
- TIM_HandleTypeDef [htim1](#)

7.49.1 Detailed Description

Interrupt Service Routines.

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.

7.49.2 Function Documentation

7.49.2.1 BusFault_Handler()

```
void BusFault_Handler (  
    void )
```

This function handles Pre-fetch fault, memory access fault.

Definition at line 117 of file stm32f4xx_it.c.

7.49.2.2 CAN2_RX0_IRQHandler()

```
void CAN2_RX0_IRQHandler (  
    void )
```

This function handles CAN2 RX0 interrupts.

Definition at line 223 of file stm32f4xx_it.c.

References hcan2.

7.49.2.3 CAN2_RX1_IRQHandler()

```
void CAN2_RX1_IRQHandler (  
    void )
```

This function handles CAN2 RX1 interrupt.

Definition at line 237 of file stm32f4xx_it.c.

References hcan2.

7.49.2.4 CAN2_TX_IRQHandler()

```
void CAN2_TX_IRQHandler (  
    void )
```

This function handles CAN2 TX interrupts.

Definition at line 209 of file stm32f4xx_it.c.

References hcan2.

7.49.2.5 DebugMon_Handler()

```
void DebugMon_Handler (  
    void )
```

This function handles Debug monitor.

Definition at line 147 of file stm32f4xx_it.c.

7.49.2.6 DMA2_Stream0_IRQHandler()

```
void DMA2_Stream0_IRQHandler (  
    void )
```

This function handles DMA2 stream0 global interrupt.

Definition at line 195 of file stm32f4xx_it.c.

References hdma_adc1.

7.49.2.7 EXTI0_IRQHandler()

```
void EXTI0_IRQHandler (  
    void )
```

This function handles EXTI line0 interrupt.

Definition at line 167 of file stm32f4xx_it.c.

References Start_Button_Input_Pin.

7.49.2.8 HardFault_Handler()

```
void HardFault_Handler (
    void )
```

This function handles Hard fault interrupt.

Definition at line 87 of file stm32f4xx_it.c.

7.49.2.9 MemManage_Handler()

```
void MemManage_Handler (
    void )
```

This function handles Memory management fault.

Definition at line 102 of file stm32f4xx_it.c.

7.49.2.10 NMI_Handler()

```
void NMI_Handler (
    void )
```

This function handles Non maskable interrupt.

Definition at line 72 of file stm32f4xx_it.c.

7.49.2.11 TIM1_UP_TIM10_IRQHandler()

```
void TIM1_UP_TIM10_IRQHandler (
    void )
```

This function handles TIM1 update interrupt and TIM10 global interrupt.

Definition at line 181 of file stm32f4xx_it.c.

References htim1.

7.49.2.12 UsageFault_Handler()

```
void UsageFault_Handler (
    void )
```

This function handles Undefined instruction or illegal state.

Definition at line 132 of file stm32f4xx_it.c.

7.49.3 Variable Documentation

7.49.3.1 hcan2

```
CAN_HandleTypeDef hcan2
```

Definition at line 116 of file can.c.

Referenced by CAN2_RX0_IRQHandler(), CAN2_RX1_IRQHandler(), CAN2_TX_IRQHandler(), CANbusTxSvc← Daemon(), HAL_CAN_RxFifo0MsgPendingCallback(), and MX_CAN2_Init().

7.49.3.2 hdma_adc1

```
DMA_HandleTypeDef hdma_adc1
```

Definition at line 29 of file adc.c.

Referenced by DMA2_Stream0_IRQHandler(), and HAL_ADC_MspInit().

7.49.3.3 htim1

```
TIM_HandleTypeDef htim1
```

Definition at line 28 of file stm32f4xx_hal_timebase_tim.c.

Referenced by HAL_InitTick(), HAL_ResumeTick(), HAL_SuspendTick(), and TIM1_UP_TIM10_IRQHandler().

7.50 Core/Src/syscalls.c File Reference

STM32CubeIDE Minimal System calls file.

```
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
```

Functions

- int `__io_putchar` (int ch) `__attribute__((weak))`
- int `__io_getchar` (void)
- void `initialise_monitor_handles` ()
- int `_getpid` (void)
- int `_kill` (int pid, int sig)
- void `_exit` (int status)
- `__attribute__((weak))`
- int `_close` (int file)
- int `_fstat` (int file, struct stat *st)
- int `_isatty` (int file)
- int `_lseek` (int file, int ptr, int dir)
- int `_open` (char *path, int flags,...)
- int `_wait` (int *status)
- int `_unlink` (char *name)
- int `_times` (struct tms *buf)
- int `_stat` (char *file, struct stat *st)
- int `_link` (char *old, char *new)
- int `_fork` (void)
- int `_execve` (char *name, char **argv, char **env)

Variables

- char ** `environ` = `__env`

7.50.1 Detailed Description

STM32CubeIDE Minimal System calls file.

Author

Auto-generated by STM32CubeIDE

```
For more information about which c-functions
need which of these lowlevel functions
please consult the Newlib libc-manual
```

Attention

Copyright (c) 2020-2023 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.

7.50.2 Function Documentation

7.50.2.1 `__attribute__()`

```
__attribute__ (  
    (weak) )
```

Definition at line 67 of file syscalls.c.

References `__io_getchar()`.

7.50.2.2 `__io_getchar()`

```
int __io_getchar (  
    void )
```

Definition at line 36 of file syscalls.c.

Referenced by `__attribute__()`.

7.50.2.3 `__io_putchar()`

```
int __io_putchar (  
    int ch )
```

7.50.2.4 `_close()`

```
int _close (  
    int file )
```

Definition at line 92 of file syscalls.c.

7.50.2.5 `_execve()`

```
int _execve (  
    char * name,  
    char ** argv,  
    char ** env )
```

Definition at line 169 of file syscalls.c.

7.50.2.6 `_exit()`

```
void _exit (
    int status )
```

Definition at line 61 of file syscalls.c.

References `_kill()`.

7.50.2.7 `_fork()`

```
int _fork (
    void )
```

Definition at line 163 of file syscalls.c.

7.50.2.8 `_fstat()`

```
int _fstat (
    int file,
    struct stat * st )
```

Definition at line 99 of file syscalls.c.

7.50.2.9 `_getpid()`

```
int _getpid (
    void )
```

Definition at line 48 of file syscalls.c.

7.50.2.10 `_isatty()`

```
int _isatty (
    int file )
```

Definition at line 106 of file syscalls.c.

7.50.2.11 `_kill()`

```
int _kill (
    int pid,
    int sig )
```

Definition at line 53 of file syscalls.c.

Referenced by `_exit()`.

7.50.2.12 `_link()`

```
int _link (
    char * old,
    char * new )
```

Definition at line 155 of file syscalls.c.

7.50.2.13 `_lseek()`

```
int _lseek (
    int file,
    int ptr,
    int dir )
```

Definition at line 112 of file syscalls.c.

7.50.2.14 `_open()`

```
int _open (
    char * path,
    int flags,
    ... )
```

Definition at line 120 of file syscalls.c.

7.50.2.15 `_stat()`

```
int _stat (
    char * file,
    struct stat * st )
```

Definition at line 148 of file syscalls.c.

7.50.2.16 `_times()`

```
int _times (
    struct tms * buf )
```

Definition at line 142 of file syscalls.c.

7.50.2.17 `_unlink()`

```
int _unlink (
    char * name )
```

Definition at line 135 of file syscalls.c.

7.50.2.18 `_wait()`

```
int _wait (
    int * status )
```

Definition at line 128 of file syscalls.c.

7.50.2.19 `initialise_monitor_handles()`

```
void initialise_monitor_handles ( )
```

Definition at line 44 of file syscalls.c.

7.50.3 Variable Documentation

7.50.3.1 `environ`

```
char** environ = __env
```

Definition at line 40 of file syscalls.c.

7.51 Core/Src/sysmem.c File Reference

STM32CubeIDE System Memory calls file.

```
#include <errno.h>
#include <stdint.h>
```

Functions

- void * [_sbrk](#) (ptrdiff_t incr)
[_sbrk\(\)](#) allocates memory to the newlib heap and is used by malloc and others from the C library

Variables

- static uint8_t * [__sbrk_heap_end](#) = NULL

7.51.1 Detailed Description

STM32CubeIDE System Memory calls file.

Author

Generated by STM32CubeIDE

```
For more information about which C functions
need which of these lowlevel functions
please consult the newlib libc manual
```

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.

7.51.2 Function Documentation

7.51.2.1 [_sbrk\(\)](#)

```
void* _sbrk (
    ptrdiff_t incr )
```

[_sbrk\(\)](#) allocates memory to the newlib heap and is used by malloc and others from the C library

```
* #####
* # .data # .bss #          newlib heap          #          MSP stack          #
* #          #          #          #          # Reserved by _Min_Stack_Size #
* #####
* ^-- RAM start          ^-- _end                      _estack, RAM end --^
*
```

This implementation starts allocating at the '_end' linker symbol The '_Min_Stack_Size' linker symbol reserves a memory for the MSP stack The implementation considers '_estack' linker symbol to be RAM end NOTE: If the MSP stack, at any point during execution, grows larger than the reserved size, please increase the '_Min_Stack_Size'.

Parameters

<i>incr</i>	Memory size
-------------	-------------

Returns

Pointer to allocated memory

Definition at line 53 of file sysmem.c.

References `__sbrk_heap_end`.

7.51.3 Variable Documentation

7.51.3.1 `__sbrk_heap_end`

```
uint8_t* __sbrk_heap_end = NULL [static]
```

Pointer to the current high watermark of the heap usage

Definition at line 30 of file sysmem.c.

Referenced by `_sbrk()`.

7.52 Core/Src/system_stm32f4xx.c File Reference

CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.

```
#include "stm32f4xx.h"
```

Macros

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

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.

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}

7.52.1 Detailed Description

CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.

Author

MCD Application Team This file provides two functions and one global variable to be called from user application:

- [SystemInit\(\)](#): This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f4xx.s" file.
- [SystemCoreClock](#) variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
- [SystemCoreClockUpdate\(\)](#): Updates the variable [SystemCoreClock](#) and must be called whenever the core clock is changed during program execution.

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.

7.53 Core/Src/temp_monitoring.c File Reference

```
#include "uvfr_utils.h"
#include "gpio.h"
```

Functions

- [uv_status initTempMonitor](#) (void *arguments)
- void [tempMonitorTask](#) (void *args)

Monitors the temperatures of various points in the tractive system, and activates various cooling systems and such accordingly.

7.53.1 Function Documentation

7.53.1.1 initTempMonitor()

```
uv_status initTempMonitor (
    void * arguments )
```

Definition at line 12 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()`.

7.53.1.2 tempMonitorTask()

```
void tempMonitorTask (
    void * args )
```

Monitors the temperatures of various points in the tractive system, and activates various cooling systems and such accordingly.

Atm, this is mostly serving as an example of a task 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 = 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 48 of file temp_monitoring.c.

References `uv_task_info::cmd_data`, `handleCANbusError()`, `hcan2`, `killSelf()`, `suspendSelf()`, `uv_task_info::task_period`, `TxData`, `TxHeader`, `TxMailbox`, `UV_KILL_CMD`, `UV_SUSPEND_CMD`, and `uvTaskDelayUntil`.

Referenced by `initTempMonitor()`.

7.54 Core/Src/tim.c File Reference

This file provides code for the configuration of the TIM instances.

```
#include "tim.h"
```

Functions

- void `MX_TIM3_Init` (void)
- void `HAL_TIM_Base_MspInit` (TIM_HandleTypeDef *tim_baseHandle)
- void `HAL_TIM_Base_MspDeInit` (TIM_HandleTypeDef *tim_baseHandle)

Variables

- TIM_HandleTypeDef [htim3](#)

7.54.1 Detailed Description

This file provides code for the configuration of the TIM instances.

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.

7.54.2 Function Documentation

7.54.2.1 HAL_TIM_Base_MspDeInit()

```
void HAL_TIM_Base_MspDeInit (
    TIM_HandleTypeDef * tim_baseHandle )
```

Definition at line 86 of file tim.c.

7.54.2.2 HAL_TIM_Base_MspInit()

```
void HAL_TIM_Base_MspInit (
    TIM_HandleTypeDef * tim_baseHandle )
```

Definition at line 70 of file tim.c.

7.54.2.3 MX_TIM3_Init()

```
void MX_TIM3_Init (
    void )
```

Definition at line 30 of file tim.c.

References [Error_Handler\(\)](#), and [htim3](#).

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

7.54.3 Variable Documentation

7.54.3.1 htim3

```
TIM_HandleTypeDef htim3
```

Definition at line 27 of file tim.c.

Referenced by HAL_TIM_PeriodElapsedCallback(), and MX_TIM3_Init().

7.55 Core/Src/uvfr_settings.c File Reference

```
#include "uvfr_utils.h"  
#include "main.h"  
#include "stdlib.h"
```

Macros

- `#define SRC_UVFR_SETTINGS_C_`

Functions

- void `setupDefaultSettings` ()
Function that allocates the necessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.
- void `nukeSettings` (`uv_vehicle_settings **settings_to_delete`)
- 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.
- void `uvSettingsProgrammerTask` (void *args)

Variables

- `uv_vehicle_settings * current_vehicle_settings = NULL`
- struct `uv_os_settings default_os_settings`

7.55.1 Macro Definition Documentation

7.55.1.1 SRC_UVFR_SETTINGS_C_

```
#define SRC_UVFR_SETTINGS_C_
```

Definition at line 7 of file uvfr_settings.c.

7.55.2 Function Documentation

7.55.2.1 nukeSettings()

```
void nukeSettings (
    uv_vehicle_settings ** settings_to_delete )
```

Definition at line 51 of file uvfr_settings.c.

7.55.2.2 setupDefaultSettings()

```
void setupDefaultSettings ( )
```

Function that allocates the necessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.

Definition at line 42 of file uvfr_settings.c.

References `current_vehicle_settings`, `default_os_settings`, and `uv_vehicle_settings::os_settings`.

Referenced by `uvSettingsInit()`.

7.55.2.3 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 64 of file uvfr_settings.c.

References `setupDefaultSettings()`, `UV_ABORTED`, `UV_ERROR`, and `UV_OK`.

Referenced by `uvInit()`.

7.55.2.4 uvSettingsProgrammerTask()

```
void uvSettingsProgrammerTask (  
    void * args )
```

Definition at line 88 of file uvfr_settings.c.

7.55.3 Variable Documentation

7.55.3.1 current_vehicle_settings

```
uv_vehicle_settings* current_vehicle_settings = NULL
```

Definition at line 15 of file uvfr_settings.c.

Referenced by setupDefaultSettings(), and uvInit().

7.56 Core/Src/uvfr_state_engine.c File Reference

File containing the implementation of the vehicle's state engine and error handling infrastructure.

```
#include "uvfr_utils.h"
```

Data Structures

- struct [state_change_daemon_args](#)

Macros

- #define [UVFR_STATE_MACHINE_IMPLIMENTATION](#)
- #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.
- [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 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 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 gnarliness here, but not unacceptable levels. Pray this thing doesn't hang itself.
- [uv_status addTaskToTaskRegister](#) ([uv_task_id](#) id, uint8_t assign_to_whom)
- [uv_status _uvValidateSpecificTask](#) ([uv_task_id](#) 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, [uv_task_info](#) *t)
: This is a function that starts tasks which are already registered in the system
- static [uv_status uvKillTaskViolently](#) ([uv_task_info](#) *t)
if a task refuses to comply with the SCD, then it has no choice but to be deleted. There is nothing that can be done.
- [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 [uvSecureVehicle](#) ()
Function to put vehicle into safe state.
- void [__uvPanic](#) (char *msg, uint8_t msg_len, 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.
- static [uv_status proccessSCDMsg](#) ([uv_scd_response](#) *msg)
Helper function for the SCD, that proccesses a message, and double checks to make sure the task that sent the message isn't straight up lying to us.
- void [_stateChangeDaemon](#) (void *args) PRIVILEGED_FUNCTION
This collects all the data changing from different tasks, and makes sure that everything works properly.

- [uv_status uvInvokeSCD](#) (void *scd_params)
used to wake up the SCD
- [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

- static [uv_task_id _next_task_id](#) = 0
- static [uv_task_info * _task_register](#) = NULL
- static [uv_task_id _next_svc_task_id](#) = 0
- static [uv_task_info * _svc_task_register](#) = NULL
- TaskHandle_t * [scd_handle_ptr](#)
- static volatile [bool SCD_active](#) = false
- static QueueHandle_t [state_change_queue](#) = NULL
- [rbtree * task_name_lut](#) = NULL
- 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
- [uv_os_settings default_os_settings](#)

7.56.1 Detailed Description

File containing the implementation of the vehicle's state engine and error handling infrastructure.

Author

Byron Oser

7.56.2 Macro Definition Documentation

7.56.2.1 UVFR_STATE_MACHINE_IMPLIMENTATION

```
#define UVFR_STATE_MACHINE_IMPLIMENTATION
```

Definition at line 10 of file uvfr_state_engine.c.

7.57 Core/Src/uvfr_utils.c File Reference

```
#include "uvfr_utils.h"
```

Macros

- `#define UV_UTILS_SRC_IMPLIMENTATION`

Functions

- void `uvInit` (void *arguments)
: Function that initializes all of the car's stuff.
- enum `uv_status_t uvUtilsReset` ()
This function is a soft-reboot of the `uv_utils_backend` and OS abstraction.
- void `setup_extern_devices` (void *argument)
- void `__uvInitPanic` ()
Low Level Panic, that does not require the full UVFR utils functionality to be operational.
- void * `__uvMallocCriticalSection` (size_t memrequest)
Wrapper function for `malloc()` that makes it thread safe.
- `uv_status __uvFreeCriticalSection` (void *ptr)
Thread-safe wrapper for `free`.
- void * `__uvMallocOS` (size_t memrequest)
`malloc()` wrapper that calls `pvPortMalloc()` rather than `malloc()`
- `uv_status __uvFreeOS` (void *ptr)
OS-based free wrapper that calls `pvPortFree`.
- `uv_status uvIsPTRValid` (void *ptr)
function that checks to make sure a pointer points to a place it is allowed to point to

Variables

- TaskHandle_t `init_task_handle`
- uint8_t `TxData` [8]

7.57.1 Macro Definition Documentation

7.57.1.1 UV_UTILS_SRC_IMPLIMENTATION

```
#define UV_UTILS_SRC_IMPLIMENTATION
```

Definition at line 9 of file `uvfr_utils.c`.

7.57.2 Function Documentation

7.57.2.1 __uvFreeCritSection()

```
uv_status __uvFreeCritSection (
    void * ptr )
```

Thread-safe wrapper for `free`.

This is typically called from the macro expansion of `uvFree(x)`

Definition at line 320 of file `uvfr_utils.c`.

References `UV_ERROR`, `UV_OK`, and `uvIsPTRValid()`.

7.57.2.2 __uvFreeOS()

```
uv_status __uvFreeOS (
    void * ptr )
```

OS-based free wrapper that calls `pvPortFree`.

Definition at line 371 of file `uvfr_utils.c`.

References `UV_ERROR`, `UV_OK`, and `uvIsPTRValid()`.

7.57.2.3 __uvInitPanic()

```
void __uvInitPanic ( )
```

Low Level Panic, that does not require the full UVFR utils functionality to be operational.

Attention

Calling `_uvInitPanic()` is irreversable 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 263 of file `uvfr_utils.c`.

Referenced by `uvInit()`, `uvInitStateEngine()`, and `uvSVCTaskManager()`.

7.57.2.4 `__uvMallocCriticalSection()`

```
void* __uvMallocCriticalSection (
    size_t memrequest )
```

Wrapper function for `malloc()` that makes it thread safe.

This typically appears in a macro expansion from `uvMalloc(x)`

Definition at line 284 of file `uvfr_utils.c`.

7.57.2.5 `__uvMallocOS()`

```
void* __uvMallocOS (
    size_t memrequest )
```

`malloc()` wrapper that calls `pvPortMalloc()` rather than `malloc()`

The reason we might want to be using `pvPortMalloc()` rather than regular `stdlib malloc()` is to consolidate the heap between RTOS and non-RTOS functions.

Definition at line 345 of file `uvfr_utils.c`.

References `UV_MALLOC_LIMIT`, `UV_OK`, and `uvIsPTRValid()`.

7.57.2.6 `setup_extern_devices()`

```
void setup_extern_devices (
    void * argument )
```

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

Definition at line 251 of file `uvfr_utils.c`.

7.57.2.7 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.

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 we have initialized the state engine, what we want to do is create the prototypes of all the tasks that will be running.

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 existence 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;
//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);
//MC_init_args->meta_task_handle = osThreadCreate(&MC_init_thread,MC_init_args);
//vTaskResume( MC_init_args->meta_task_handle );
retval =
    xTaskCreate(MC_Startup, "MC_init", 128, MC_init_args, osPriorityAboveNormal, &(MC_init_args->meta_task_handle));
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}
```

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", 128, BMS_init_args, osPriorityAboveNormal, &(BMS_init_args->meta_task_handle));
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}
```

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, "BMS_init", 128, IMD_init_args, osPriorityAboveNormal, &(IMD_init_args->meta_task_handle));
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}
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, dum dum
if (retval != pdPASS) {
    //FUCK
    error_msg = "bruh";
}
uint16_t ext_devices_status = 0x000F; //Tracks which devices are currently setup
```

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 37 of file `uvfr_utils.c`.

References `__uvInitPanic()`, `BMS_Init()`, `uv_vehicle_settings::bms_settings`, `changeVehicleState()`, `current_vehicle_settings`, `uv_init_task_response::device`, `uv_init_task_response::errmsg`, `uv_vehicle_settings::imd_settings`, `INIT_CHECK_PERIOD`, `uv_init_task_args::init_info_queue`, `init_task_handle`, `initIMD()`, `initPDU()`, `MAX_INIT_TIME`, `uv_vehicle_settings::mc_settings`, `MC_Startup()`, `uv_init_task_args::meta_task_handle`, `uv_init_task_response::nchar`, `uv_init_task_args::specific_args`, `uv_init_task_response::status`, `UV_OK`, `UV_READY`, `uvInitStateEngine()`, `uvSettingsInit()`, and `uvStartStateMachine()`.

Referenced by `MX_FREERTOS_Init()`.

7.57.2.8 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 393 of file `uvfr_utils.c`.

References `UV_ERROR`, `UV_OK`, and `UV_WARNING`.

Referenced by `__uvFreeCriticalSection()`, `__uvFreeOS()`, and `__uvMallocOS()`.

7.57.2.9 uvUtilsReset()

```
enum uv_status_t uvUtilsReset ( )
```

This function is a soft-reboot of the `uv_utils_backend` and OS abstraction.

The idea here is to basically start from a blank slate and boot up everything. So therefore we must:

- Halt state machine.
- Nuke vehicle operation related tasks.
- Nuke the state machine
- Nuke old settings

reinitialize `uv_utils`

Definition at line 243 of file `uvfr_utils.c`.

References `UV_OK`.

7.57.3 Variable Documentation

7.57.3.1 init_task_handle

```
TaskHandle_t init_task_handle
```

Definition at line 51 of file `freertos.c`.

Referenced by `MX_FREERTOS_Init()`, and `uvInit()`.

7.57.3.2 TxData

```
uint8_t TxData[8]
```

Definition at line 7 of file `constants.c`.

7.58 Core/Src/uvfr_vehicle_commands.c File Reference

Index

- `_BV`
 - Utility Macros, 41
- `_BV_16`
 - Utility Macros, 41
- `_BV_32`
 - Utility Macros, 42
- `_BV_8`
 - Utility Macros, 42
- `_LONGEST_SC_TIME`
 - uvfr_state_engine.h, 201
- `_NUM_ERRORS_`
 - errorLUT.h, 123
- `_NUM_LOGGABLE_PARAMS`
 - daq.h, 113
- `_SC_DAEMON_PERIOD`
 - uvfr_state_engine.h, 201
- `_SRC_UVFR_DAQ`
 - daq.c, 227
- `_UV_DEFAULT_TASK_INSTANCES`
 - uvfr_state_engine.h, 201
- `_UV_DEFAULT_TASK_PERIOD`
 - uvfr_state_engine.h, 201
- `_UV_DEFAULT_TASK_STACK_SIZE`
 - uvfr_state_engine.h, 202
- `_UV_MIN_TASK_PERIOD`
 - uvfr_state_engine.h, 202
- `__attribute__`
 - syscalls.c, 279
- `__io_getchar`
 - syscalls.c, 280
- `__io_putchar`
 - syscalls.c, 280
- `__sbrk_heap_end`
 - sysmem.c, 285
- `__uvFreeCriticalSection`
 - uvfr_utils.c, 295
- `__uvFreeOS`
 - uvfr_utils.c, 295
- `__uvInitPanic`
 - uvfr_utils.c, 295
 - uvfr_utils.h, 213
- `__uvMallocCriticalSection`
 - uvfr_utils.c, 295
- `__uvMallocOS`
 - uvfr_utils.c, 296
- `__uvPanic`
 - State Engine Internals, 29
- `_close`
 - syscalls.c, 280
- `_execve`
 - syscalls.c, 280
- `_exit`
 - syscalls.c, 280
- `_fork`
 - syscalls.c, 281
- `_fstat`
 - syscalls.c, 281
- `_getpid`
 - syscalls.c, 281
- `_isatty`
 - syscalls.c, 281
- `_kill`
 - syscalls.c, 281
- `_link`
 - syscalls.c, 282
- `_lseek`
 - syscalls.c, 282
- `_next_svc_task_id`
 - State Engine, 11
- `_next_task_id`
 - State Engine, 11
- `_open`
 - syscalls.c, 282
- `_sbrk`
 - sysmem.c, 284
- `_stat`
 - syscalls.c, 282
- `_stateChangeDaemon`
 - State Engine Internals, 29
- `_svc_task_register`
 - State Engine, 12
- `_task_register`
 - State Engine, 12
- `_times`
 - syscalls.c, 282
- `_unlink`
 - syscalls.c, 283
- `_uvValidateSpecificTask`
 - State Engine Internals, 31
- `_wait`
 - syscalls.c, 283
- `a`
 - s_curve_torque_map_args, 78
- `ABOVE_NORMAL`
 - State Engine API, 23
- `absolute_max_acc_pwr`
 - driving_loop_args, 63
- `absolute_max_accum_current`

- driving_loop_args, 64
- absolute_max_motor_rpm
 - driving_loop_args, 64
- absolute_max_motor_torque
 - driving_loop_args, 64
- AC_current_offset_fault
 - motor_controller.h, 152
- ACC_POWER
 - daq.h, 114
- ACC_POWER_LIMIT
 - daq.h, 114
- accel
 - uvfr_utils.h, 211
- accelerate_ramp
 - motor_controller.h, 151
- ACCELERATOR_PEDAL_RATIO
 - daq.h, 114
- access_control_info, 57
 - bin_semaphore, 57
 - mutex, 57
 - semaphore, 57
 - uvfr_utils.h, 208
- access_control_t
 - uvfr_utils.h, 211
- access_control_type
 - uvfr_utils.h, 208
- accum_regen_soc_threshold
 - driving_loop_args, 64
- activation_time
 - p_status, 73
- active_states
 - uv_task_info, 92
- adc.c
 - hadc1, 220
 - hadc2, 220
 - HAL_ADC_MspDeInit, 218
 - HAL_ADC_MspInit, 218
 - hdma_adc1, 220
 - MX_ADC1_Init, 219
 - MX_ADC2_Init, 219
- adc.h
 - ADC1_BUF_LEN, 102
 - ADC1_CHNL_CNT, 102
 - ADC1_MAX_VOLT, 102
 - ADC1_MIN_VOLT, 102
 - ADC1_SAMPLES, 102
 - ADC2_BUF_LEN, 103
 - ADC2_CHNL_CNT, 103
 - ADC2_MAX_VOLT, 103
 - ADC2_MIN_VOLT, 103
 - ADC2_SAMPLES, 103
 - hadc1, 104
 - hadc2, 105
 - MX_ADC1_Init, 104
 - MX_ADC2_Init, 104
- adc1_APPS1
 - driving_loop.c, 233
 - main.c, 254
- adc1_APPS2
 - driving_loop.c, 233
 - main.c, 255
- adc1_BPS1
 - driving_loop.c, 234
 - main.c, 255
- adc1_BPS2
 - driving_loop.c, 234
 - main.c, 255
- ADC1_BUF_LEN
 - adc.h, 102
- ADC1_CHNL_CNT
 - adc.h, 102
- ADC1_MAX_VOLT
 - adc.h, 102
- ADC1_MIN_VOLT
 - adc.h, 102
- ADC1_SAMPLES
 - adc.h, 102
- ADC2_BUF_LEN
 - adc.h, 103
- ADC2_CHNL_CNT
 - adc.h, 103
- adc2_CoolantFlow
 - main.c, 255
- adc2_CoolantTemp
 - main.c, 255
- ADC2_MAX_VOLT
 - adc.h, 103
- ADC2_MIN_VOLT
 - adc.h, 103
- ADC2_SAMPLES
 - adc.h, 103
- adc_buf1
 - main.c, 256
- adc_buf2
 - main.c, 256
- ADC_measurement_problem
 - motor_controller.h, 152
- ADC_sequencer_problem
 - motor_controller.h, 152
- addTaskToTaskRegister
 - State Engine Internals, 31
- AHBPrescTable
 - STM32F4xx_System_Private_Variables, 54
- air_temperature
 - motor_controller.h, 153
- amogus
 - oled.h, 156
- APBPrescTable
 - STM32F4xx_System_Private_Variables, 54
- APPS1_ADC_VAL
 - daq.h, 114
- APPS2_ADC_VAL
 - daq.h, 114
- apps_bottom
 - driving_loop_args, 64
- apps_implausibility_recovery_threshold

- driving_loop_args, 65
- apps_plausibility_check_threshold
 - driving_loop_args, 65
- apps_top
 - driving_loop_args, 65
- assert_param
 - stm32f4xx_hal_conf.h, 171
- auxiliary_voltage_min_limit
 - motor_controller.h, 152
- AVG_CELL_TEMP
 - daq.h, 114
- b
 - s_curve_torque_map_args, 78
- battery_voltage
 - imd.h, 139
- BELOW_NORMAL
 - State Engine API, 23
- bin_semaphore
 - access_control_info, 57
- BLACK
 - rb_tree.h, 163
- bleed_resistor_overload
 - motor_controller.h, 152
- bleeder_resistor_warning
 - motor_controller.h, 152
- Blue_LED_GPIO_Port
 - main.h, 145
- Blue_LED_Pin
 - main.h, 145
- BMS
 - uvfr_utils.h, 212
- bms.c
 - BMS_Init, 221
- bms.h
 - BMS_Init, 106
 - bms_settings_t, 106
 - DEFAULT_BMS_CAN_TIMEOUT, 105
- BMS_CURRENT
 - daq.h, 114
- BMS_ERRORS
 - daq.h, 114
- BMS_Init
 - bms.c, 221
 - bms.h, 106
- bms_settings
 - uv_vehicle_settings, 99
- bms_settings_t, 58
 - bms.h, 106
 - mc_CAN_timeout, 58
- BMS_VOLTAGE
 - daq.h, 114
- bool
 - uvfr_utils.h, 208
- BPS1_ADC_VAL
 - daq.h, 114
- BPS2_ADC_VAL
 - daq.h, 114
- bps_implausibility_recovery_threshold
 - driving_loop_args, 65
 - bps_plausibility_check_threshold
 - driving_loop_args, 65
- BRAKE_PRESSURE_PA
 - daq.h, 114
- BusFault_Handler
 - stm32f4xx_it.c, 275
 - stm32f4xx_it.h, 190
- c
 - s_curve_torque_map_args, 79
- can.c
 - CANbusRxSVCDaemon, 222
 - CANbusTxSvcDaemon, 222
 - HAL_CAN_ERROR_INVALID_CALLBACK, 222
 - HAL_CAN_MspDeInit, 223
 - HAL_CAN_MspInit, 223
 - HAL_CAN_RxFifo0MsgPendingCallback, 223
 - HAL_CAN_RxFifo1MsgPendingCallback, 223
 - handleCANbusError, 224
 - hcan2, 225
 - MX_CAN2_Init, 224
 - Tx_msg_queue, 225
 - uvSendCanMSG, 224
- can.h
 - CAN_RX_DAEMON_NAME, 107
 - CAN_TX_DAEMON_NAME, 107
 - CANbusTxSvcDaemon, 108
 - HAL_CAN_RxFifo0MsgPendingCallback, 108
 - HAL_CAN_RxFifo1MsgPendingCallback, 108
 - hcan2, 109
 - MX_CAN2_Init, 109
 - uv_CAN_msg, 108
 - uv_status, 108
 - uvSendCanMSG, 109
- CAN2_RX0_IRQHandler
 - stm32f4xx_it.c, 275
 - stm32f4xx_it.h, 190
- CAN2_RX1_IRQHandler
 - stm32f4xx_it.c, 275
 - stm32f4xx_it.h, 190
- CAN2_TX_IRQHandler
 - stm32f4xx_it.c, 275
 - stm32f4xx_it.h, 190
- can_id
 - daq_datapoint, 60
- CAN_IDs
 - constants.h, 110
- CAN_RX_DAEMON_NAME
 - can.h, 107
- CAN_timeout_error
 - motor_controller.h, 152
- CAN_TX_DAEMON_NAME
 - can.h, 107
- CANbusRxSVCDaemon
 - can.c, 222
- CANbusTxSvcDaemon
 - can.c, 222
 - can.h, 108

- changeVehicleState
 - State Engine API, [25](#)
- check_ecode_ID
 - motor_controller.h, [152](#)
- checkBlackHeight
 - rb_tree.c, [264](#)
- checkOrder
 - rb_tree.c, [265](#)
- clear_errors
 - motor_controller.h, [152](#)
- cmd_data
 - uv_task_info, [92](#)
- CMSIS, [48](#)
- color
 - rbnode, [74](#)
- compare
 - rbtree, [76](#)
- configASSERT
 - FreeRTOSConfig.h, [125](#)
- configCHECK_FOR_STACK_OVERFLOW
 - FreeRTOSConfig.h, [125](#)
- configCPU_CLOCK_HZ
 - FreeRTOSConfig.h, [125](#)
- configENABLE_BACKWARD_COMPATIBILITY
 - FreeRTOSConfig.h, [126](#)
- configENABLE_FPU
 - FreeRTOSConfig.h, [126](#)
- configENABLE_MPU
 - FreeRTOSConfig.h, [126](#)
- configKERNEL_INTERRUPT_PRIORITY
 - FreeRTOSConfig.h, [126](#)
- configLIBRARY_LOWEST_INTERRUPT_PRIORITY
 - FreeRTOSConfig.h, [126](#)
- configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY
 - FreeRTOSConfig.h, [126](#)
- configMAX_CO_ROUTINE_PRIORITIES
 - FreeRTOSConfig.h, [127](#)
- configMAX_PRIORITIES
 - FreeRTOSConfig.h, [127](#)
- configMAX_SYSCALL_INTERRUPT_PRIORITY
 - FreeRTOSConfig.h, [127](#)
- configMAX_TASK_NAME_LEN
 - FreeRTOSConfig.h, [127](#)
- configMESSAGE_BUFFER_LENGTH_TYPE
 - FreeRTOSConfig.h, [127](#)
- configMINIMAL_STACK_SIZE
 - FreeRTOSConfig.h, [127](#)
- configPRIO_BITS
 - FreeRTOSConfig.h, [128](#)
- configQUEUE_REGISTRY_SIZE
 - FreeRTOSConfig.h, [128](#)
- configRECORD_STACK_HIGH_ADDRESS
 - FreeRTOSConfig.h, [128](#)
- configSUPPORT_DYNAMIC_ALLOCATION
 - FreeRTOSConfig.h, [128](#)
- configSUPPORT_STATIC_ALLOCATION
 - FreeRTOSConfig.h, [128](#)
- configTICK_RATE_HZ
 - FreeRTOSConfig.h, [128](#)
- configTIMER_QUEUE_LENGTH
 - FreeRTOSConfig.h, [129](#)
- configTIMER_TASK_PRIORITY
 - FreeRTOSConfig.h, [129](#)
- configTIMER_TASK_STACK_DEPTH
 - FreeRTOSConfig.h, [129](#)
- configTOTAL_HEAP_SIZE
 - FreeRTOSConfig.h, [129](#)
- configUSE_16_BIT_TICKS
 - FreeRTOSConfig.h, [129](#)
- configUSE_APPLICATION_TASK_TAG
 - FreeRTOSConfig.h, [129](#)
- configUSE_CO_ROUTINES
 - FreeRTOSConfig.h, [130](#)
- configUSE_COUNTING_SEMAPHORES
 - FreeRTOSConfig.h, [130](#)
- configUSE_IDLE_HOOK
 - FreeRTOSConfig.h, [130](#)
- configUSE_MALLOC_FAILED_HOOK
 - FreeRTOSConfig.h, [130](#)
- configUSE_MUTEXES
 - FreeRTOSConfig.h, [130](#)
- configUSE_PORT_OPTIMISED_TASK_SELECTION
 - FreeRTOSConfig.h, [131](#)
- configUSE_PREEMPTION
 - FreeRTOSConfig.h, [131](#)
- configUSE_TICK_HOOK
 - FreeRTOSConfig.h, [131](#)
- configUSE_TIMERS
 - FreeRTOSConfig.h, [131](#)
- constants.c
 - RxData, [225](#)
 - RxHeader, [226](#)
 - TxData, [226](#)
 - TxHeader, [226](#)
 - TxMailbox, [226](#)
- constants.h
 - CAN_IDs, [110](#)
 - IMD_CAN_ID_Rx, [110](#)
 - IMD_CAN_ID_Tx, [110](#)
 - MC_CAN_ID_Rx, [110](#)
 - MC_CAN_ID_Tx, [110](#)
 - PDU_CAN_ID_Tx, [110](#)
 - RxData, [110](#)
 - RxHeader, [110](#)
 - TxData, [111](#)
 - TxHeader, [111](#)
 - TxMailbox, [111](#)
- control_map_fn
 - drivingMode, [69](#)
- Core/Inc/adc.h, [101](#)
- Core/Inc/bms.h, [105](#)
- Core/Inc/can.h, [106](#)
- Core/Inc/constants.h, [110](#)
- Core/Inc/daq.h, [112](#)
- Core/Inc/dash.h, [116](#)
- Core/Inc/dma.h, [117](#)

- Core/Inc/driving_loop.h, 118
- Core/Inc/errorLUT.h, 123
- Core/Inc/FreeRTOSConfig.h, 124
- Core/Inc/gpio.h, 135
- Core/Inc/imd.h, 136
- Core/Inc/main.h, 144
- Core/Inc/motor_controller.h, 147
- Core/Inc/odometer.h, 155
- Core/Inc/oled.h, 156
- Core/Inc/pdu.h, 157
- Core/Inc/rb_tree.h, 162
- Core/Inc/spi.h, 167
- Core/Inc/stm32f4xx_hal_conf.h, 169
- Core/Inc/stm32f4xx_it.h, 189
- Core/Inc/temp_monitoring.h, 193
- Core/Inc/tim.h, 194
- Core/Inc/uvfr_global_config.h, 195
- Core/Inc/uvfr_settings.h, 196
- Core/Inc/uvfr_state_engine.h, 198
- Core/Inc/uvfr_utils.h, 204
- Core/Inc/uvfr_vehicle_commands.h, 216
- Core/Src/adc.c, 217
- Core/Src/bms.c, 220
- Core/Src/can.c, 221
- Core/Src/constants.c, 225
- Core/Src/daq.c, 227
- Core/Src/dash.c, 229
- Core/Src/dma.c, 230
- Core/Src/driving_loop.c, 231
- Core/Src/freertos.c, 234
- Core/Src/gpio.c, 238
- Core/Src/imd.c, 239
- Core/Src/main.c, 250
- Core/Src/motor_controller.c, 256
- Core/Src/odometer.c, 260
- Core/Src/oled.c, 261
- Core/Src/pdu.c, 261
- Core/Src/rb_tree.c, 264
- Core/Src/spi.c, 269
- Core/Src/stm32f4xx_hal_msp.c, 271
- Core/Src/stm32f4xx_hal_timebase_tim.c, 272
- Core/Src/stm32f4xx_it.c, 274
- Core/Src/syscalls.c, 278
- Core/Src/system.c, 284
- Core/Src/system_stm32f4xx.c, 285
- Core/Src/temp_monitoring.c, 286
- Core/Src/tim.c, 287
- Core/Src/uvfr_settings.c, 289
- Core/Src/uvfr_state_engine.c, 291
- Core/Src/uvfr_utils.c, 294
- Core/Src/uvfr_vehicle_commands.c, 299
- count
 - rbtree, 76
- critical_AC_current
 - motor_controller.h, 152
- current_derate_temperature
 - motor_controller.h, 153
- CURRENT_DRIVING_MODE
 - daq.h, 114
- current_feed_forward
 - motor_controller.h, 151
- current_vehicle_settings
 - uvfr_settings.c, 291
 - uvfr_settings.h, 198
- daq.c
 - _SRC_UVFR_DAQ, 227
 - daqMasterTask, 228
 - daqSubTask, 228
 - deleteDaqSubTask, 228
 - deleteParamList, 228
 - initDaqTask, 228
 - param_LUT, 229
 - startDaqSubTasks, 229
 - stopDaqSubTasks, 229
- daq.h
 - _NUM_LOGGABLE_PARAMS, 113
 - ACC_POWER, 114
 - ACC_POWER_LIMIT, 114
 - ACCELERATOR_PEDAL_RATIO, 114
 - APPS1_ADC_VAL, 114
 - APPS2_ADC_VAL, 114
 - AVG_CELL_TEMP, 114
 - BMS_CURRENT, 114
 - BMS_ERRORS, 114
 - BMS_VOLTAGE, 114
 - BPS1_ADC_VAL, 114
 - BPS2_ADC_VAL, 114
 - BRAKE_PRESSURE_PA, 114
 - CURRENT_DRIVING_MODE, 114
 - daq_child_task, 113
 - daq_datapoint, 113
 - daq_loop_args, 113
 - daq_param_list_node, 113
 - daqMasterTask, 115
 - data_type, 113
 - initDaqTask, 115
 - loggable_params, 114
 - MAX_CELL_TEMP, 114
 - MAX_LOGGABLE_PARAMS, 114
 - MC_CURRENT, 114
 - MC_ERRORS, 114
 - MC_TEMP, 114
 - MC_VOLTAGE, 114
 - MIN_CELL_TEMP, 114
 - MOTOR_CURRENT, 114
 - MOTOR_RPM, 114
 - MOTOR_TEMP, 114
 - param_LUT, 115
 - POWER_DERATE_FACTOR, 114
 - UV_DOUBLE, 114
 - UV_FLOAT, 114
 - UV_INT16, 114
 - UV_INT32, 114
 - UV_INT8, 114
 - UV_STRING, 114
 - UV_UINT16, 114

- UV_UINT32, 114
- UV_UINT8, 114
- daq_child_task, 58
 - daq.h, 113
 - meta_task_handle, 59
 - param_list, 59
 - period, 59
 - treenode, 59
- daq_datapoint, 60
 - can_id, 60
 - daq.h, 113
 - period, 60
 - type, 60
- daq_loop_args, 61
 - daq.h, 113
 - datapoints, 61
 - minimum_daq_period, 61
 - padding, 61
 - padding2, 61
 - throttle_daq_to_preserve_performance, 62
- daq_param_list_node, 62
 - daq.h, 113
 - next, 62
 - param_idx, 62
- daq_settings
 - uv_vehicle_settings, 99
- daqMasterTask
 - daq.c, 228
 - daq.h, 115
- daqSubTask
 - daq.c, 228
- dash.c
 - Update_Batt_Temp, 230
 - Update_RPM, 230
 - Update_State_Of_Charge, 230
- dash.h
 - Dash_Battery_Temperature, 116
 - dash_can_ids, 116
 - Dash_Motor_Temperature, 116
 - Dash_RPM, 116
 - Dash_State_of_Charge, 116
 - Update_Batt_Temp, 116
 - Update_RPM, 116
 - Update_State_Of_Charge, 117
- Dash_Battery_Temperature
 - dash.h, 116
- dash_can_ids
 - dash.h, 116
- Dash_Motor_Temperature
 - dash.h, 116
- Dash_RPM
 - dash.h, 116
- Dash_State_of_Charge
 - dash.h, 116
- data
 - rbnode, 74
 - uv_CAN_msg, 82
- DATA_CACHE_ENABLE
 - stm32f4xx_hal_conf.h, 172
- data_type
 - daq.h, 113
- datapoints
 - daq_loop_args, 61
- DC_bus_voltage
 - motor_controller.h, 150
- DEBUG_CAN_IN_MAIN
 - main.c, 251
- DebugMon_Handler
 - stm32f4xx_it.c, 276
 - stm32f4xx_it.h, 191
- DEFAULT_BMS_CAN_TIMEOUT
 - bms.h, 105
- DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT
 - motor_controller.h, 148
- default_os_settings
 - State Engine, 12
- DEFAULT_PERIOD
 - driving_loop.h, 119
- defaultTaskHandle
 - freertos.c, 237
- deleteDaqSubTask
 - daq.c, 228
- deleteParamList
 - daq.c, 228
- deleteRepair
 - rb_tree.c, 265
- deletion_delay
 - uv_task_info, 92
- deletion_states
 - uv_task_info, 92
- deserializeBigE16
 - Utility Macros, 42
- deserializeBigE32
 - Utility Macros, 42
- deserializeSmallE16
 - Utility Macros, 42
- deserializeSmallE32
 - Utility Macros, 43
- desired_motor_speed
 - motor_controller.c, 259
- destroy
 - rbtree, 76
- destroyAllNodes
 - rb_tree.c, 265
- device
 - uv_init_task_response, 86
- disable_brake_light_msg
 - pdu.h, 159
- disable_coolant_pump_msg
 - pdu.h, 158
- disable_left_cooling_fan_msg
 - pdu.h, 158
- disable_motor_controller_msg
 - pdu.h, 159
- disable_right_cooling_fan_msg
 - pdu.h, 158

- disable_shutdown_circuit_msg
 - pdu.h, 159
- disable_speaker_msg
 - pdu.h, 159
- dismantling_ramp
 - motor_controller.h, 151
- DL_internal_state
 - driving_loop.h, 120
- dlc
 - uv_CAN_msg, 83
- dm_name
 - drivingMode, 69
- dma.c
 - MX_DMA_Init, 231
- dma.h
 - MX_DMA_Init, 118
- DMA2_Stream0_IRQHandler
 - stm32f4xx_it.c, 276
 - stm32f4xx_it.h, 191
- dmodes
 - driving_loop_args, 66
- DP83848_PHY_ADDRESS
 - stm32f4xx_hal_conf.h, 172
- driving_loop.c
 - adc1_APPS1, 233
 - adc1_APPS2, 233
 - adc1_BPS1, 234
 - adc1_BPS2, 234
 - initDrivingLoop, 232
 - StartDrivingLoop, 232
- driving_loop.h
 - DEFAULT_PERIOD, 119
 - DL_internal_state, 120
 - driving_loop_args, 119
 - drivingMode, 119
 - drivingModeParams, 119
 - Erroneous, 121
 - exp_speed_map, 121
 - exp_torque_map, 121
 - exp_torque_map_args, 120
 - Implausible, 121
 - initDrivingLoop, 121
 - linear_speed_map, 121
 - linear_torque_map, 121
 - linear_torque_map_args, 120
 - map_mode, 121
 - MC_POWER, 120
 - MC_RPM, 120
 - MC_Torque, 120
 - Plausible, 121
 - s_curve_speed_map, 121
 - s_curve_torque_map, 121
 - s_curve_torque_map_args, 120
 - StartDrivingLoop, 121
- driving_loop_args, 63
 - absolute_max_acc_pwr, 63
 - absolute_max_accum_current, 64
 - absolute_max_motor_rpm, 64
 - absolute_max_motor_torque, 64
 - accum_regen_soc_threshold, 64
 - apps_bottom, 64
 - apps_implausibility_recovery_threshold, 65
 - apps_plausibility_check_threshold, 65
 - apps_top, 65
 - bps_implausibility_recovery_threshold, 65
 - bps_plausibility_check_threshold, 65
 - dmodes, 66
 - driving_loop.h, 119
 - max_accum_current_5s, 66
 - max_apps_offset, 66
 - max_apps_value, 66
 - max_BPS_value, 66
 - min_apps_offset, 67
 - min_apps_value, 67
 - min_BPS_value, 67
 - num_driving_modes, 67
 - period, 67
 - regen_rpm_cutoff, 68
- driving_loop_settings
 - uv_vehicle_settings, 99
- drivingLoopArgs, 68
- drivingMode, 68
 - control_map_fn, 69
 - dm_name, 69
 - driving_loop.h, 119
 - flags, 69
 - map_fn_params, 69
 - max_acc_pwr, 70
 - max_current, 70
 - max_motor_torque, 70
- drivingModeParams, 70
 - driving_loop.h, 119
- e_code
 - uv_internal_params, 87
- ecode_timeout_error
 - motor_controller.h, 152
- econ
 - uvfr_utils.h, 211
- ECUMASTER_PMU
 - uvfr_global_config.h, 195
- enable_brake_light_msg
 - pdu.h, 159
- enable_coolant_pump_msg
 - pdu.h, 158
- ENABLE_FLASH_SETTINGS
 - uvfr_settings.h, 197
- enable_left_cooling_fan_msg
 - pdu.h, 158
- enable_motor_controller_msg
 - pdu.h, 159
- enable_right_cooling_fan_msg
 - pdu.h, 158
- enable_shutdown_circuit_msg
 - pdu.h, 159
- enable_speaker_msg
 - pdu.h, 159

- endianSwap
 - Utility Macros, 43
- endianSwap16
 - Utility Macros, 43
- endianSwap32
 - Utility Macros, 43
- endianSwap8
 - Utility Macros, 43
- environ
 - syscalls.c, 283
- eprom_read_error
 - motor_controller.h, 152
- Err_CH
 - imd.h, 137
- Err_clock
 - imd.h, 137
- Err_temp
 - imd.h, 137
- Err_Vexi
 - imd.h, 137
- Err_Vpwr
 - imd.h, 137
- Err_Vx1
 - imd.h, 137
- Err_Vx2
 - imd.h, 137
- Err_VxR
 - imd.h, 137
- Err_Watchdog
 - imd.h, 137
- errmsg
 - uv_init_task_response, 86
- Erroneous
 - driving_loop.h, 121
- Error_flags
 - imd.h, 139
- Error_Handler
 - main.c, 252
 - main.h, 146
- errorLUT.h
 - _NUM_ERRORS_, 123
- ETH_RX_BUF_SIZE
 - stm32f4xx_hal_conf.h, 172
- ETH_RXBUFNB
 - stm32f4xx_hal_conf.h, 172
- ETH_TX_BUF_SIZE
 - stm32f4xx_hal_conf.h, 172
- ETH_TXBUFNB
 - stm32f4xx_hal_conf.h, 172
- Exc_off
 - imd.h, 139
- exp_speed_map
 - driving_loop.h, 121
- exp_torque_map
 - driving_loop.h, 121
- exp_torque_map_args, 71
 - driving_loop.h, 120
 - gamma, 71
 - offset, 71
- EXTERNAL_CLOCK_VALUE
 - stm32f4xx_hal_conf.h, 173
- EXTIO_IRQHandler
 - stm32f4xx_it.c, 276
 - stm32f4xx_it.h, 191
- false
 - Utility Macros, 44
- feedback_signal_error
 - motor_controller.h, 152
- feedback_signal_problem
 - motor_controller.h, 152
- firmware_version
 - motor_controller.h, 152
- flags
 - drivingMode, 69
 - uv_CAN_msg, 83
- freertos.c
 - defaultTaskHandle, 237
 - init_settings, 237
 - init_task_handle, 237
 - MX_FREERTOS_Init, 235
 - StartDefaultTask, 235
 - vApplicationGetIdleTaskMemory, 235
 - vApplicationGetTimerTaskMemory, 236
 - vApplicationIdleHook, 236
 - vApplicationMallocFailedHook, 236
 - vApplicationStackOverflowHook, 236
 - vApplicationTickHook, 236
 - xIdleStack, 237
 - xIdleTaskTCBBuffer, 238
 - xTimerStack, 238
 - xTimerTaskTCBBuffer, 238
- FreeRTOSConfig.h
 - configASSERT, 125
 - configCHECK_FOR_STACK_OVERFLOW, 125
 - configCPU_CLOCK_HZ, 125
 - configENABLE_BACKWARD_COMPATIBILITY, 126
 - configENABLE_FPU, 126
 - configENABLE_MPU, 126
 - configKERNEL_INTERRUPT_PRIORITY, 126
 - configLIBRARY_LOWEST_INTERRUPT_PRIORITY, 126
 - configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY, 126
 - configMAX_CO_ROUTINE_PRIORITIES, 127
 - configMAX_PRIORITIES, 127
 - configMAX_SYSCALL_INTERRUPT_PRIORITY, 127
 - configMAX_TASK_NAME_LEN, 127
 - configMESSAGE_BUFFER_LENGTH_TYPE, 127
 - configMINIMAL_STACK_SIZE, 127
 - configPRIO_BITS, 128
 - configQUEUE_REGISTRY_SIZE, 128
 - configRECORD_STACK_HIGH_ADDRESS, 128
 - configSUPPORT_DYNAMIC_ALLOCATION, 128
 - configSUPPORT_STATIC_ALLOCATION, 128

- configTICK_RATE_HZ, [128](#)
- configTIMER_QUEUE_LENGTH, [129](#)
- configTIMER_TASK_PRIORITY, [129](#)
- configTIMER_TASK_STACK_DEPTH, [129](#)
- configTOTAL_HEAP_SIZE, [129](#)
- configUSE_16_BIT_TICKS, [129](#)
- configUSE_APPLICATION_TASK_TAG, [129](#)
- configUSE_CO_ROUTINES, [130](#)
- configUSE_COUNTING_SEMAPHORES, [130](#)
- configUSE_IDLE_HOOK, [130](#)
- configUSE_MALLOC_FAILED_HOOK, [130](#)
- configUSE_MUTEXES, [130](#)
- configUSE_PORT_OPTIMISED_TASK_SELECTION, [131](#)
- configUSE_PREEMPTION, [131](#)
- configUSE_TICK_HOOK, [131](#)
- configUSE_TIMERS, [131](#)
- INCLUDE_eTaskGetState, [131](#)
- INCLUDE_pcTaskGetTaskName, [131](#)
- INCLUDE_uxTaskGetStackHighWaterMark, [132](#)
- INCLUDE_uxTaskGetStackHighWaterMark2, [132](#)
- INCLUDE_uxTaskPriorityGet, [132](#)
- INCLUDE_vTaskCleanUpResources, [132](#)
- INCLUDE_vTaskDelay, [132](#)
- INCLUDE_vTaskDelayUntil, [132](#)
- INCLUDE_vTaskDelete, [133](#)
- INCLUDE_vTaskPrioritySet, [133](#)
- INCLUDE_vTaskSuspend, [133](#)
- INCLUDE_xEventGroupSetBitFromISR, [133](#)
- INCLUDE_xQueueGetMutexHolder, [133](#)
- INCLUDE_xSemaphoreGetMutexHolder, [133](#)
- INCLUDE_xTaskAbortDelay, [134](#)
- INCLUDE_xTaskDelayUntil, [134](#)
- INCLUDE_xTaskGetCurrentTaskHandle, [134](#)
- INCLUDE_xTaskGetHandle, [134](#)
- INCLUDE_xTaskGetSchedulerState, [134](#)
- INCLUDE_xTimerPendFunctionCall, [134](#)
- vPortSVCHandler, [135](#)
- xPortPendSVHandler, [135](#)
- xPortSysTickHandler, [135](#)
- main.c, [252](#)
- HAL_ADC_LevelOutOfWindowCallback
 - main.c, [252](#)
- HAL_ADC_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [173](#)
- HAL_ADC_MspDeInit
 - adc.c, [218](#)
- HAL_ADC_MspInit
 - adc.c, [218](#)
- HAL_CAN_ERROR_INVALID_CALLBACK
 - can.c, [222](#)
- HAL_CAN_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [173](#)
- HAL_CAN_MspDeInit
 - can.c, [223](#)
- HAL_CAN_MspInit
 - can.c, [223](#)
- HAL_CAN_RxFifo0MsgPendingCallback
 - can.c, [223](#)
 - can.h, [108](#)
- HAL_CAN_RxFifo1MsgPendingCallback
 - can.c, [223](#)
 - can.h, [108](#)
- HAL_CORTEX_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [173](#)
- HAL_DMA_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [173](#)
- HAL_EXTI_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [173](#)
- HAL_FLASH_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [174](#)
- HAL_GPIO_EXTI_Callback
 - main.c, [252](#)
- HAL_GPIO_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [174](#)
- HAL_InitTick
 - stm32f4xx_hal_timebase_tim.c, [272](#)
- HAL_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [174](#)
- HAL_MspInit
 - stm32f4xx_hal_msp.c, [271](#)
- HAL_PWR_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [174](#)
- HAL_RCC_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [174](#)
- HAL_ResumeTick
 - stm32f4xx_hal_timebase_tim.c, [273](#)
- HAL_SPI_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [174](#)
- HAL_SPI_MspDeInit
 - spi.c, [270](#)
- HAL_SPI_MspInit
 - spi.c, [270](#)
- HAL_SuspendTick
 - stm32f4xx_hal_timebase_tim.c, [273](#)
- HAL_TIM_Base_MspDeInit
 - tim.c, [288](#)
- HAL_TIM_Base_MspInit
- gamma
 - exp_torque_map_args, [71](#)
- getSVCTaskID
 - uvfr_state_engine.h, [203](#)
- global_context
 - uvfr_utils.h, [215](#)
- gpio.c
 - MX_GPIO_Init, [239](#)
- gpio.h
 - MX_GPIO_Init, [136](#)
- hadc1
 - adc.c, [220](#)
 - adc.h, [104](#)
- hadc2
 - adc.c, [220](#)
 - adc.h, [105](#)
- HAL_ADC_ConvCpltCallback

- tim.c, [288](#)
- HAL_TIM_MODULE_ENABLED
 - stm32f4xx_hal_conf.h, [175](#)
- HAL_TIM_PeriodElapsedCallback
 - main.c, [253](#)
- handle
 - uv_binary_semaphore_info, [82](#)
 - uv_mutex_info, [88](#)
 - uv_semaphore_info, [91](#)
- handleCANbusError
 - can.c, [224](#)
- HardFault_Handler
 - stm32f4xx_it.c, [276](#)
 - stm32f4xx_it.h, [191](#)
- Hardware_Error
 - imd.h, [139](#)
- hardware_fault
 - motor_controller.h, [152](#)
- hcan2
 - can.c, [225](#)
 - can.h, [109](#)
 - stm32f4xx_it.c, [278](#)
- hdma_adc1
 - adc.c, [220](#)
 - stm32f4xx_it.c, [278](#)
- High_Battery_Voltage
 - imd.h, [138](#)
- HIGH_PRIORITY
 - State Engine API, [23](#)
- High_Uncertainty
 - imd.h, [139](#)
- HSE_STARTUP_TIMEOUT
 - stm32f4xx_hal_conf.h, [175](#)
- HSE_VALUE
 - stm32f4xx_hal_conf.h, [175](#)
 - STM32F4xx_System_Private_Includes, [50](#)
- HSI_VALUE
 - stm32f4xx_hal_conf.h, [175](#)
 - STM32F4xx_System_Private_Includes, [50](#)
- hspl1
 - spl.c, [271](#)
 - spl.h, [168](#)
- htim1
 - stm32f4xx_hal_timebase_tim.c, [273](#)
 - stm32f4xx_it.c, [278](#)
- htim3
 - tim.c, [289](#)
 - tim.h, [194](#)
- IDLE_TASK_PRIORITY
 - State Engine API, [23](#)
- IGBT_temp_max_limit
 - motor_controller.h, [152](#)
- igbt_temperature
 - motor_controller.h, [153](#)
- IGBT_temperature_warning
 - motor_controller.h, [152](#)
- IMD
 - uvfr_utils.h, [212](#)

- imd.c
 - IMD_Check_Battery_Voltage, [240](#)
 - IMD_Check_Error_Flags, [241](#)
 - IMD_Check_Isolation_Capacitances, [241](#)
 - IMD_Check_Isolation_Resistances, [241](#)
 - IMD_Check_Isolation_State, [241](#)
 - IMD_Check_Max_Battery_Working_Voltage, [242](#)
 - IMD_Check_Part_Name, [242](#)
 - IMD_Check_Safety_Touch_Current, [242](#)
 - IMD_Check_Safety_Touch_Energy, [242](#)
 - IMD_Check_Serial_Number, [243](#)
 - IMD_Check_Status_Bits, [243](#)
 - IMD_Check_Temperature, [243](#)
 - IMD_Check_Uptime, [243](#)
 - IMD_Check_Version, [244](#)
 - IMD_Check_Voltages_Vp_and_Vn, [244](#)
 - IMD_error_flags_requested, [245](#)
 - IMD_Expected_Part_Name, [245](#)
 - IMD_Expected_Serial_Number, [246](#)
 - IMD_Expected_Version, [246](#)
 - IMD_High_Uncertainty, [246](#)
 - IMD_Parse_Message, [244](#)
 - IMD_Part_Name_0_Set, [246](#)
 - IMD_Part_Name_1_Set, [246](#)
 - IMD_Part_Name_2_Set, [247](#)
 - IMD_Part_Name_3_Set, [247](#)
 - IMD_Part_Name_Set, [247](#)
 - IMD_Read_Part_Name, [247](#)
 - IMD_Read_Serial_Number, [247](#)
 - IMD_Read_Version, [248](#)
 - IMD_Request_Status, [244](#)
 - IMD_Serial_Number_0_Set, [248](#)
 - IMD_Serial_Number_1_Set, [248](#)
 - IMD_Serial_Number_2_Set, [248](#)
 - IMD_Serial_Number_3_Set, [248](#)
 - IMD_Serial_Number_Set, [249](#)
 - IMD_Startup, [245](#)
 - IMD_status_bits, [249](#)
 - IMD_Temperature, [249](#)
 - IMD_Version_0_Set, [249](#)
 - IMD_Version_1_Set, [249](#)
 - IMD_Version_2_Set, [250](#)
 - IMD_Version_Set, [250](#)
 - initIMD, [245](#)
- imd.h
 - battery_voltage, [139](#)
 - Err_CH, [137](#)
 - Err_clock, [137](#)
 - Err_temp, [137](#)
 - Err_Vexi, [137](#)
 - Err_Vpwr, [137](#)
 - Err_Vx1, [137](#)
 - Err_Vx2, [137](#)
 - Err_VxR, [137](#)
 - Err_Watchdog, [137](#)
 - Error_flags, [139](#)
 - Exc_off, [139](#)
 - Hardware_Error, [139](#)

- High_Battery_Voltage, [138](#)
- High_Uncertainty, [139](#)
- IMD_Check_Battery_Voltage, [139](#)
- IMD_Check_Error_Flags, [139](#)
- IMD_Check_Isolation_Capacitances, [140](#)
- IMD_Check_Isolation_Resistances, [140](#)
- IMD_Check_Isolation_State, [140](#)
- IMD_Check_Max_Battery_Working_Voltage, [140](#)
- IMD_Check_Part_Name, [141](#)
- IMD_Check_Safety_Touch_Current, [141](#)
- IMD_Check_Safety_Touch_Energy, [141](#)
- IMD_Check_Serial_Number, [141](#)
- IMD_Check_Status_Bits, [142](#)
- IMD_Check_Temperature, [142](#)
- IMD_Check_Uptime, [142](#)
- IMD_Check_Version, [142](#)
- IMD_Check_Voltages_Vp_and_Vn, [143](#)
- imd_error_flags, [137](#)
- imd_high_resolution_measurements, [137](#)
- imd_manufacturer_requests, [138](#)
- IMD_Parse_Message, [143](#)
- IMD_Request_Status, [143](#)
- IMD_Startup, [143](#)
- imd_status_bits, [138](#)
- imd_status_requests, [139](#)
- initIMD, [144](#)
- isolation_capacitances, [139](#)
- isolation_resistances, [139](#)
- isolation_state, [139](#)
- Isolation_status_bit0, [138](#)
- Isolation_status_bit1, [138](#)
- Low_Battery_Voltage, [138](#)
- Max_battery_working_voltage, [139](#)
- Part_name_0, [138](#)
- Part_name_1, [138](#)
- Part_name_2, [138](#)
- Part_name_3, [138](#)
- safety_touch_current, [139](#)
- safety_touch_energy, [139](#)
- Serial_number_0, [138](#)
- Serial_number_1, [138](#)
- Serial_number_2, [138](#)
- Serial_number_3, [138](#)
- Temperature, [139](#)
- Touch_energy_fault, [139](#)
- Uptime_counter, [138](#)
- Vb_hi_res, [138](#)
- Version_0, [138](#)
- Version_1, [138](#)
- Version_2, [138](#)
- Vexc_hi_res, [138](#)
- Vn_hi_res, [138](#)
- voltages_Vp_and_Vn, [139](#)
- Vp_hi_res, [138](#)
- Vpwr_hi_res, [138](#)
- IMD_CAN_ID_Rx
 - constants.h, [110](#)
- IMD_CAN_ID_Tx
 - constants.h, [110](#)
- IMD_Check_Battery_Voltage
 - imd.c, [240](#)
 - imd.h, [139](#)
- IMD_Check_Error_Flags
 - imd.c, [241](#)
 - imd.h, [139](#)
- IMD_Check_Isolation_Capacitances
 - imd.c, [241](#)
 - imd.h, [140](#)
- IMD_Check_Isolation_Resistances
 - imd.c, [241](#)
 - imd.h, [140](#)
- IMD_Check_Isolation_State
 - imd.c, [241](#)
 - imd.h, [140](#)
- IMD_Check_Max_Battery_Working_Voltage
 - imd.c, [242](#)
 - imd.h, [140](#)
- IMD_Check_Part_Name
 - imd.c, [242](#)
 - imd.h, [141](#)
- IMD_Check_Safety_Touch_Current
 - imd.c, [242](#)
 - imd.h, [141](#)
- IMD_Check_Safety_Touch_Energy
 - imd.c, [242](#)
 - imd.h, [141](#)
- IMD_Check_Serial_Number
 - imd.c, [243](#)
 - imd.h, [141](#)
- IMD_Check_Status_Bits
 - imd.c, [243](#)
 - imd.h, [142](#)
- IMD_Check_Temperature
 - imd.c, [243](#)
 - imd.h, [142](#)
- IMD_Check_Uptime
 - imd.c, [243](#)
 - imd.h, [142](#)
- IMD_Check_Version
 - imd.c, [244](#)
 - imd.h, [142](#)
- IMD_Check_Voltages_Vp_and_Vn
 - imd.c, [244](#)
 - imd.h, [143](#)
- imd_error_flags
 - imd.h, [137](#)
- IMD_error_flags_requested
 - imd.c, [245](#)
- IMD_Expected_Part_Name
 - imd.c, [245](#)
- IMD_Expected_Serial_Number
 - imd.c, [246](#)
- IMD_Expected_Version
 - imd.c, [246](#)
- imd_high_resolution_measurements
 - imd.h, [137](#)

IMD_High_Uncertainty
 imd.c, [246](#)
 imd_manufacturer_requests
 imd.h, [138](#)
 IMD_Parse_Message
 imd.c, [244](#)
 imd.h, [143](#)
 IMD_Part_Name_0_Set
 imd.c, [246](#)
 IMD_Part_Name_1_Set
 imd.c, [246](#)
 IMD_Part_Name_2_Set
 imd.c, [247](#)
 IMD_Part_Name_3_Set
 imd.c, [247](#)
 IMD_Part_Name_Set
 imd.c, [247](#)
 IMD_Read_Part_Name
 imd.c, [247](#)
 IMD_Read_Serial_Number
 imd.c, [247](#)
 IMD_Read_Version
 imd.c, [248](#)
 IMD_Request_Status
 imd.c, [244](#)
 imd.h, [143](#)
 IMD_Serial_Number_0_Set
 imd.c, [248](#)
 IMD_Serial_Number_1_Set
 imd.c, [248](#)
 IMD_Serial_Number_2_Set
 imd.c, [248](#)
 IMD_Serial_Number_3_Set
 imd.c, [248](#)
 IMD_Serial_Number_Set
 imd.c, [249](#)
 imd_settings
 uv_vehicle_settings, [99](#)
 IMD_Startup
 imd.c, [245](#)
 imd.h, [143](#)
 IMD_status_bits
 imd.c, [249](#)
 imd_status_bits
 imd.h, [138](#)
 imd_status_requests
 imd.h, [139](#)
 IMD_Temperature
 imd.c, [249](#)
 IMD_Version_0_Set
 imd.c, [249](#)
 IMD_Version_1_Set
 imd.c, [249](#)
 IMD_Version_2_Set
 imd.c, [250](#)
 IMD_Version_Set
 imd.c, [250](#)
 Implausible
 driving_loop.h, [121](#)
 INCLUDE_eTaskGetState
 FreeRTOSConfig.h, [131](#)
 INCLUDE_pcTaskGetTaskName
 FreeRTOSConfig.h, [131](#)
 INCLUDE_uxTaskGetStackHighWaterMark
 FreeRTOSConfig.h, [132](#)
 INCLUDE_uxTaskGetStackHighWaterMark2
 FreeRTOSConfig.h, [132](#)
 INCLUDE_uxTaskPriorityGet
 FreeRTOSConfig.h, [132](#)
 INCLUDE_vTaskCleanUpResources
 FreeRTOSConfig.h, [132](#)
 INCLUDE_vTaskDelay
 FreeRTOSConfig.h, [132](#)
 INCLUDE_vTaskDelayUntil
 FreeRTOSConfig.h, [132](#)
 INCLUDE_vTaskDelete
 FreeRTOSConfig.h, [133](#)
 INCLUDE_vTaskPrioritySet
 FreeRTOSConfig.h, [133](#)
 INCLUDE_vTaskSuspend
 FreeRTOSConfig.h, [133](#)
 INCLUDE_xEventGroupSetBitFromISR
 FreeRTOSConfig.h, [133](#)
 INCLUDE_xQueueGetMutexHolder
 FreeRTOSConfig.h, [133](#)
 INCLUDE_xSemaphoreGetMutexHolder
 FreeRTOSConfig.h, [133](#)
 INCLUDE_xTaskAbortDelay
 FreeRTOSConfig.h, [134](#)
 INCLUDE_xTaskDelayUntil
 FreeRTOSConfig.h, [134](#)
 INCLUDE_xTaskGetCurrentTaskHandle
 FreeRTOSConfig.h, [134](#)
 INCLUDE_xTaskGetHandle
 FreeRTOSConfig.h, [134](#)
 INCLUDE_xTaskGetSchedulerState
 FreeRTOSConfig.h, [134](#)
 INCLUDE_xTimerPendFunctionCall
 FreeRTOSConfig.h, [134](#)
 INIT_CHECK_PERIOD
 uvfr_utils.h, [207](#)
 init_info_queue
 uv_init_task_args, [85](#)
 init_params
 uv_internal_params, [87](#)
 init_settings
 freertos.c, [237](#)
 init_task_handle
 freertos.c, [237](#)
 uvfr_utils.c, [299](#)
 initDaqTask
 daq.c, [228](#)
 daq.h, [115](#)
 initDrivingLoop
 driving_loop.c, [232](#)
 driving_loop.h, [121](#)

- initialise_monitor_handles
 - syscalls.c, [283](#)
- initIMD
 - imd.c, [245](#)
 - imd.h, [144](#)
- initOdometer
 - odometer.c, [260](#)
 - odometer.h, [155](#)
- initPDU
 - pdu.c, [261](#)
 - pdu.h, [159](#)
- initTempMonitor
 - temp_monitoring.c, [286](#)
 - temp_monitoring.h, [193](#)
- INORDER
 - rb_tree.h, [165](#)
- insertRepair
 - rb_tree.c, [265](#)
- INSTRUCTION_CACHE_ENABLE
 - stm32f4xx_hal_conf.h, [175](#)
- integral_memory_max
 - motor_controller.h, [151](#)
- integral_time_constant
 - motor_controller.h, [151](#)
- intended_recipient
 - uv_task_msg_t, [97](#)
- internal_hardware_voltage_problem
 - motor_controller.h, [152](#)
- is_default
 - uv_vehicle_settings, [99](#)
- isolation_capacitances
 - imd.h, [139](#)
- isolation_resistances
 - imd.h, [139](#)
- isolation_state
 - imd.h, [139](#)
- Isolation_status_bit0
 - imd.h, [138](#)
- Isolation_status_bit1
 - imd.h, [138](#)
- isPowerOfTwo
 - Utility Macros, [44](#)
- killEmAll
 - State Engine Internals, [31](#)
- killSelf
 - State Engine Internals, [32](#)
- leakage_inductance_ph_ph
 - motor_controller.h, [150](#)
- left
 - rbnode, [75](#)
- limp
 - uvfr_utils.h, [211](#)
- linear_speed_map
 - driving_loop.h, [121](#)
- linear_torque_map
 - driving_loop.h, [121](#)
- linear_torque_map_args, [71](#)
- driving_loop.h, [120](#)
- offset, [72](#)
- slope, [72](#)
- loggable_params
 - daq.h, [114](#)
- Low_Battery_Voltage
 - imd.h, [138](#)
- LOW_PRIORITY
 - State Engine API, [23](#)
- LSE_STARTUP_TIMEOUT
 - stm32f4xx_hal_conf.h, [176](#)
- LSE_VALUE
 - stm32f4xx_hal_conf.h, [176](#)
- LSI_VALUE
 - stm32f4xx_hal_conf.h, [176](#)
- MAC_ADDR0
 - stm32f4xx_hal_conf.h, [176](#)
- MAC_ADDR1
 - stm32f4xx_hal_conf.h, [176](#)
- MAC_ADDR2
 - stm32f4xx_hal_conf.h, [177](#)
- MAC_ADDR3
 - stm32f4xx_hal_conf.h, [177](#)
- MAC_ADDR4
 - stm32f4xx_hal_conf.h, [177](#)
- MAC_ADDR5
 - stm32f4xx_hal_conf.h, [177](#)
- main
 - main.c, [253](#)
- main.c
 - adc1_APPS1, [254](#)
 - adc1_APPS2, [255](#)
 - adc1_BPS1, [255](#)
 - adc1_BPS2, [255](#)
 - adc2_CoolantFlow, [255](#)
 - adc2_CoolantTemp, [255](#)
 - adc_buf1, [256](#)
 - adc_buf2, [256](#)
 - DEBUG_CAN_IN_MAIN, [251](#)
 - Error_Handler, [252](#)
 - HAL_ADC_ConvCpltCallback, [252](#)
 - HAL_ADC_LevelOutOfWindowCallback, [252](#)
 - HAL_GPIO_EXTI_Callback, [252](#)
 - HAL_TIM_PeriodElapsedCallback, [253](#)
 - main, [253](#)
 - MX_FREERTOS_Init, [254](#)
 - SystemClock_Config, [254](#)
- main.h
 - Blue_LED_GPIO_Port, [145](#)
 - Blue_LED_Pin, [145](#)
 - Error_Handler, [146](#)
 - Orange_LED_GPIO_Port, [145](#)
 - Orange_LED_Pin, [145](#)
 - Red_LED_GPIO_Port, [145](#)
 - Red_LED_Pin, [146](#)
 - Start_Button_Input_EXTI_IRQn, [146](#)
 - Start_Button_Input_GPIO_Port, [146](#)
 - Start_Button_Input_Pin, [146](#)

- mains_voltage_max_limit
 - motor_controller.h, [152](#)
- mains_voltage_min_limit
 - motor_controller.h, [152](#)
- map_fn_params
 - drivingMode, [69](#)
- map_mode
 - driving_loop.h, [121](#)
- max_acc_pwr
 - drivingMode, [70](#)
- max_accum_current_5s
 - driving_loop_args, [66](#)
- max_apps_offset
 - driving_loop_args, [66](#)
- max_apps_value
 - driving_loop_args, [66](#)
- Max_battery_working_voltage
 - imd.h, [139](#)
- max_BPS_value
 - driving_loop_args, [66](#)
- MAX_CELL_TEMP
 - daq.h, [114](#)
- max_current
 - drivingMode, [70](#)
- MAX_INIT_TIME
 - uvfr_utils.h, [207](#)
- MAX_LOGGABLE_PARAMS
 - daq.h, [114](#)
- max_motor_speed
 - motor_controller.c, [259](#)
- max_motor_torque
 - drivingMode, [70](#)
- MAX_NUM_MANAGED_TASKS
 - State Engine, [10](#)
- max_svc_task_period
 - uv_os_settings, [89](#)
- max_task_period
 - uv_os_settings, [89](#)
- MC_CAN_ID_Rx
 - constants.h, [110](#)
- MC_CAN_ID_Tx
 - constants.h, [110](#)
- mc_CAN_timeout
 - bms_settings_t, [58](#)
 - motor_controllor_settings, [73](#)
- MC_Check_Error_Warning
 - motor_controller.c, [257](#)
 - motor_controller.h, [153](#)
- MC_Check_Firmware
 - motor_controller.c, [257](#)
 - motor_controller.h, [153](#)
- MC_Check_Serial_Number
 - motor_controller.c, [257](#)
 - motor_controller.h, [153](#)
- MC_CURRENT
 - daq.h, [114](#)
- MC_ERRORS
 - daq.h, [114](#)
- MC_Expected_FW_Version
 - motor_controller.c, [259](#)
- MC_Expected_Serial_Number
 - motor_controller.c, [259](#)
- MC_Parse_Message
 - motor_controller.c, [257](#)
 - motor_controller.h, [154](#)
- MC_POWER
 - driving_loop.h, [120](#)
- MC_Request_Data
 - motor_controller.c, [258](#)
 - motor_controller.h, [154](#)
- MC_RPM
 - driving_loop.h, [120](#)
- MC_Send_Data
 - motor_controller.c, [258](#)
 - motor_controller.h, [154](#)
- mc_settings
 - uv_vehicle_settings, [100](#)
- MC_Speed_Control
 - motor_controller.h, [154](#)
- MC_Startup
 - motor_controller.c, [258](#)
 - motor_controller.h, [154](#)
- MC_TEMP
 - daq.h, [114](#)
- MC_Torque
 - driving_loop.h, [120](#)
- MC_Torque_Control
 - motor_controller.c, [258](#)
 - motor_controller.h, [155](#)
- MC_Validate
 - motor_controller.c, [259](#)
- MC_VOLTAGE
 - daq.h, [114](#)
- MEDIUM_PRIORITY
 - State Engine API, [23](#)
- MemManage_Handler
 - stm32f4xx_it.c, [277](#)
 - stm32f4xx_it.h, [192](#)
- message_size
 - uv_task_msg_t, [97](#)
- message_type
 - uv_task_msg_t, [97](#)
- meta_id
 - uv_scd_response, [90](#)
- meta_task_handle
 - daq_child_task, [59](#)
 - state_change_daemon_args, [79](#)
 - uv_init_task_args, [85](#)
- min
 - rbtree, [77](#)
- min_apps_offset
 - driving_loop_args, [67](#)
- min_apps_value
 - driving_loop_args, [67](#)
- min_BPS_value
 - driving_loop_args, [67](#)

- MIN_CELL_TEMP
 - daq.h, 114
- min_task_period
 - uv_os_settings, 89
- minimum_daq_period
 - daq_loop_args, 61
- minimum_magnetising_current
 - motor_controller.h, 150
- motor_continuous_current
 - motor_controller.h, 150
- MOTOR_CONTROLLER
 - uvfr_utils.h, 212
- motor_controller.c
 - desired_motor_speed, 259
 - max_motor_speed, 259
 - MC_Check_Error_Warning, 257
 - MC_Check_Firmware, 257
 - MC_Check_Serial_Number, 257
 - MC_Expected_FW_Version, 259
 - MC_Expected_Serial_Number, 259
 - MC_Parse_Message, 257
 - MC_Request_Data, 258
 - MC_Send_Data, 258
 - MC_Startup, 258
 - MC_Torque_Control, 258
 - MC_Validate, 259
- motor_controller.h
 - AC_current_offset_fault, 152
 - accelerate_ramp, 151
 - ADC_measurement_problem, 152
 - ADC_sequencer_problem, 152
 - air_temperature, 153
 - auxiliary_voltage_min_limit, 152
 - bleed_resistor_overload, 152
 - bleeder_resistor_warning, 152
 - CAN_timeout_error, 152
 - check_ecode_ID, 152
 - clear_errors, 152
 - critical_AC_current, 152
 - current_derate_temperature, 153
 - current_feed_forward, 151
 - DC_bus_voltage, 150
 - DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT, 148
 - dismantling_ramp, 151
 - ecode_timeout_error, 152
 - eprom_read_error, 152
 - feedback_signal_error, 152
 - feedback_signal_problem, 152
 - firmware_version, 152
 - hardware_fault, 152
 - IGBT_temp_max_limit, 152
 - igbt_temperature, 153
 - IGBT_temperature_warning, 152
 - integral_memory_max, 151
 - integral_time_constant, 151
 - internal_hardware_voltage_problem, 152
 - leakage_inductance_ph_ph, 150
 - mains_voltage_max_limit, 152
 - mains_voltage_min_limit, 152
 - MC_Check_Error_Warning, 153
 - MC_Check_Firmware, 153
 - MC_Check_Serial_Number, 153
 - MC_Parse_Message, 154
 - MC_Request_Data, 154
 - MC_Send_Data, 154
 - MC_Speed_Control, 154
 - MC_Startup, 154
 - MC_Torque_Control, 155
 - minimum_magnetising_current, 150
 - motor_continuous_current, 150
 - motor_controller_current_parameters, 149
 - motor_controller_errors_warnings, 152
 - motor_controller_io, 149
 - motor_controller_limp_mode, 149
 - motor_controller_measurements, 150
 - motor_controller_motor_constants, 150
 - motor_controller_PI_values, 150
 - motor_controller_repeating_time, 151
 - motor_controller_settings, 149
 - motor_controller_speed_parameters, 151
 - motor_controller_startup, 151
 - motor_controller_status_information_errors_warnings, 152
 - motor_controller_temperatures, 153
 - motor_ke_constant, 150
 - motor_kt_constant, 150
 - motor_magnetising_inductance, 150
 - motor_max_current, 150
 - motor_pole_number, 150
 - motor_temp_max_limit, 152
 - motor_temperature, 153
 - motor_temperature_switch_off_point, 150
 - motor_temperature_warning, 152
 - N_actual, 151
 - N_cmd, 151
 - N_error, 151
 - N_lim, 149
 - N_lim_minus, 149
 - N_lim_plus, 149
 - N_set, 151
 - nominal_magnetizing_current, 150
 - nominal_motor_frequency, 150
 - nominal_motor_voltage, 150
 - none, 151
 - one_hundred_ms, 151
 - parameter_conflict_detected, 152
 - power_factor, 150
 - proportional_gain, 151
 - proportional_gain_2, 151
 - race_away_detected, 152
 - ramp_set_current, 151
 - rated_motor_speed, 150
 - recuperation_ramp, 151
 - rotate_field_enable_not_present_norun, 152
 - rotate_field_enable_not_present_run, 152

- rotor_resistance, 150
- special_CPU_fault, 152
- speed_actual_resolution_limit, 152
- stator_leakage_inductance, 150
- stator_resistance_ph_ph, 150
- temp_sensor_pt1, 153
- temp_sensor_pt2, 153
- temp_sensor_pt3, 153
- temp_sensor_pt4, 153
- time_constant_rotor, 150
- time_constant_stator, 150
- todo1, 149
- todo6969, 149
- tripzone_glitch_detected, 152
- Vout_saturation_max_limit, 152
- warning_5, 152
- warning_9, 152
- watchdog_reset, 152
- motor_controller_current_parameters
 - motor_controller.h, 149
- motor_controller_errors_warnings
 - motor_controller.h, 152
- motor_controller_io
 - motor_controller.h, 149
- motor_controller_limp_mode
 - motor_controller.h, 149
- motor_controller_measurements
 - motor_controller.h, 150
- motor_controller_motor_constants
 - motor_controller.h, 150
- motor_controller_PI_values
 - motor_controller.h, 150
- motor_controller_repeating_time
 - motor_controller.h, 151
- motor_controller_settings
 - motor_controller.h, 149
- motor_controller_speed_parameters
 - motor_controller.h, 151
- motor_controller_startup
 - motor_controller.h, 151
- motor_controller_status_information_errors_warnings
 - motor_controller.h, 152
- motor_controller_temperatures
 - motor_controller.h, 153
- motor_controller_settings, 72
 - mc_CAN_timeout, 73
- MOTOR_CURRENT
 - daq.h, 114
- motor_ke_constant
 - motor_controller.h, 150
- motor_kt_constant
 - motor_controller.h, 150
- motor_magnetising_inductance
 - motor_controller.h, 150
- motor_max_current
 - motor_controller.h, 150
- motor_pole_number
 - motor_controller.h, 150
- MOTOR_RPM
 - daq.h, 114
- MOTOR_TEMP
 - daq.h, 114
- motor_temp_max_limit
 - motor_controller.h, 152
- motor_temperature
 - motor_controller.h, 153
- motor_temperature_switch_off_point
 - motor_controller.h, 150
- motor_temperature_warning
 - motor_controller.h, 152
- msg_contents
 - uv_task_msg_t, 98
- msg_id
 - uv_CAN_msg, 83
- mutex
 - access_control_info, 57
- MX_ADC1_Init
 - adc.c, 219
 - adc.h, 104
- MX_ADC2_Init
 - adc.c, 219
 - adc.h, 104
- MX_CAN2_Init
 - can.c, 224
 - can.h, 109
- MX_DMA_Init
 - dma.c, 231
 - dma.h, 118
- MX_FREERTOS_Init
 - freertos.c, 235
 - main.c, 254
- MX_GPIO_Init
 - gpio.c, 239
 - gpio.h, 136
- MX_SPI1_Init
 - spi.c, 270
 - spi.h, 168
- MX_TIM3_Init
 - tim.c, 288
 - tim.h, 194
- N_actual
 - motor_controller.h, 151
- N_cmd
 - motor_controller.h, 151
- N_error
 - motor_controller.h, 151
- N_lim
 - motor_controller.h, 149
- N_lim_minus
 - motor_controller.h, 149
- N_lim_plus
 - motor_controller.h, 149
- N_set
 - motor_controller.h, 151
- nchar
 - uv_init_task_response, 86

- next
 - daq_param_list_node, 62
- nil
 - rbtree, 77
- NMI_Handler
 - stm32f4xx_it.c, 277
 - stm32f4xx_it.h, 192
- nominal_magnetizing_current
 - motor_controller.h, 150
- nominal_motor_frequency
 - motor_controller.h, 150
- nominal_motor_voltage
 - motor_controller.h, 150
- none
 - motor_controller.h, 151
- normal
 - uvfr_utils.h, 211
- nukeSettings
 - uvfr_settings.c, 290
 - uvfr_settings.h, 197
- num_driving_modes
 - driving_loop_args, 67
- odometer.c
 - initOdometer, 260
 - odometerTask, 260
- odometer.h
 - initOdometer, 155
 - odometerTask, 155
- odometerTask
 - odometer.c, 260
 - odometer.h, 155
- offset
 - exp_torque_map_args, 71
 - linear_torque_map_args, 72
- oled.h
 - amogus, 156
 - oled_config, 156
 - oled_Write, 157
 - oled_Write_Cmd, 157
 - oled_Write_Data, 157
 - refresh_OLED, 157
 - wait, 157
- oled_config
 - oled.h, 156
- oled_Write
 - oled.h, 157
- oled_Write_Cmd
 - oled.h, 157
- oled_Write_Data
 - oled.h, 157
- one_hundred_ms
 - motor_controller.h, 151
- Orange_LED_GPIO_Port
 - main.h, 145
- Orange_LED_Pin
 - main.h, 145
- os_settings
 - uv_vehicle_settings, 100
- p_status, 73
 - activation_time, 73
 - peripheral_status, 73
 - uvfr_utils.h, 208
- padding
 - daq_loop_args, 61
- padding2
 - daq_loop_args, 61
- param_idx
 - daq_param_list_node, 62
- param_list
 - daq_child_task, 59
- param_LUT
 - daq.c, 229
 - daq.h, 115
- parameter_conflict_detected
 - motor_controller.h, 152
- parent
 - rbnode, 75
 - uv_task_info, 93
- parent_msg_queue
 - task_management_info, 80
- Part_name_0
 - imd.h, 138
- Part_name_1
 - imd.h, 138
- Part_name_2
 - imd.h, 138
- Part_name_3
 - imd.h, 138
- PDU
 - uvfr_utils.h, 212
- pdu.c
 - initPDU, 261
 - PDU_disable_brake_light, 261
 - PDU_disable_coolant_pump, 262
 - PDU_disable_cooling_fans, 262
 - PDU_disable_motor_controller, 262
 - PDU_disable_shutdown_circuit, 262
 - PDU_enable_brake_light, 262
 - PDU_enable_coolant_pump, 263
 - PDU_enable_cooling_fans, 263
 - PDU_enable_motor_controller, 263
 - PDU_enable_shutdown_circuit, 263
 - PDU_speaker_chirp, 263
- pdu.h
 - disable_brake_light_msg, 159
 - disable_coolant_pump_msg, 158
 - disable_left_cooling_fan_msg, 158
 - disable_motor_controller_msg, 159
 - disable_right_cooling_fan_msg, 158
 - disable_shutdown_circuit_msg, 159
 - disable_speaker_msg, 159
 - enable_brake_light_msg, 159
 - enable_coolant_pump_msg, 158
 - enable_left_cooling_fan_msg, 158
 - enable_motor_controller_msg, 159
 - enable_right_cooling_fan_msg, 158

- enable_shutdown_circuit_msg, 159
- enable_speaker_msg, 159
- initPDU, 159
- PDU_disable_brake_light, 159
- PDU_disable_coolant_pump, 159
- PDU_disable_cooling_fans, 160
- PDU_disable_motor_controller, 160
- PDU_disable_shutdown_circuit, 160
- PDU_enable_brake_light, 160
- PDU_enable_coolant_pump, 160
- PDU_enable_cooling_fans, 161
- PDU_enable_motor_controller, 161
- PDU_enable_shutdown_circuit, 161
- pdu_messages_20A, 158
- pdu_messages_5A, 158
- PDU_speaker_chirp, 161
- PDU_CAN_ID_Tx
 - constants.h, 110
- PDU_disable_brake_light
 - pdu.c, 261
 - pdu.h, 159
- PDU_disable_coolant_pump
 - pdu.c, 262
 - pdu.h, 159
- PDU_disable_cooling_fans
 - pdu.c, 262
 - pdu.h, 160
- PDU_disable_motor_controller
 - pdu.c, 262
 - pdu.h, 160
- PDU_disable_shutdown_circuit
 - pdu.c, 262
 - pdu.h, 160
- PDU_enable_brake_light
 - pdu.c, 262
 - pdu.h, 160
- PDU_enable_coolant_pump
 - pdu.c, 263
 - pdu.h, 160
- PDU_enable_cooling_fans
 - pdu.c, 263
 - pdu.h, 161
- PDU_enable_motor_controller
 - pdu.c, 263
 - pdu.h, 161
- PDU_enable_shutdown_circuit
 - pdu.c, 263
 - pdu.h, 161
- pdu_messages_20A
 - pdu.h, 158
- pdu_messages_5A
 - pdu.h, 158
- pdu_settings
 - uv_vehicle_settings, 100
- PDU_speaker_chirp
 - pdu.c, 263
 - pdu.h, 161
- period
 - daq_child_task, 59
 - daq_datapoint, 60
 - driving_loop_args, 67
- peripheral_status
 - p_status, 73
 - uv_internal_params, 87
- PHY_AUTONEGO_COMPLETE
 - stm32f4xx_hal_conf.h, 177
- PHY_AUTONEGOTIATION
 - stm32f4xx_hal_conf.h, 177
- PHY_BCR
 - stm32f4xx_hal_conf.h, 178
- PHY_BSR
 - stm32f4xx_hal_conf.h, 178
- PHY_CONFIG_DELAY
 - stm32f4xx_hal_conf.h, 178
- PHY_DUPLEX_STATUS
 - stm32f4xx_hal_conf.h, 178
- PHY_FULLDUPLEX_100M
 - stm32f4xx_hal_conf.h, 178
- PHY_FULLDUPLEX_10M
 - stm32f4xx_hal_conf.h, 179
- PHY_HALFDUPLEX_100M
 - stm32f4xx_hal_conf.h, 179
- PHY_HALFDUPLEX_10M
 - stm32f4xx_hal_conf.h, 179
- PHY_ISOLATE
 - stm32f4xx_hal_conf.h, 179
- PHY_JABBER_DETECTION
 - stm32f4xx_hal_conf.h, 179
- PHY_LINKED_STATUS
 - stm32f4xx_hal_conf.h, 180
- PHY_LOOPBACK
 - stm32f4xx_hal_conf.h, 180
- PHY_POWERDOWN
 - stm32f4xx_hal_conf.h, 180
- PHY_READ_TO
 - stm32f4xx_hal_conf.h, 180
- PHY_RESET
 - stm32f4xx_hal_conf.h, 180
- PHY_RESET_DELAY
 - stm32f4xx_hal_conf.h, 181
- PHY_RESTART_AUTONEGOTIATION
 - stm32f4xx_hal_conf.h, 181
- PHY_SPEED_STATUS
 - stm32f4xx_hal_conf.h, 181
- PHY_SR
 - stm32f4xx_hal_conf.h, 181
- PHY_WRITE_TO
 - stm32f4xx_hal_conf.h, 181
- Plausible
 - driving_loop.h, 121
- POSTORDER
 - rb_tree.h, 165
- POWER_DERATE_FACTOR
 - daq.h, 114
- power_factor
 - motor_controller.h, 150

PREFETCH_ENABLE
 stm32f4xx_hal_conf.h, 182
 PREORDER
 rb_tree.h, 165
 previous_state
 State Engine, 12
 print
 rb_tree.c, 266
 rbtree, 77
 proccessSCDMsg
 State Engine Internals, 32
 PROGRAMMING
 State Engine API, 25
 proportional_gain
 motor_controller.h, 151
 proportional_gain_2
 motor_controller.h, 151

 race_away_detected
 motor_controller.h, 152
 ramp_set_current
 motor_controller.h, 151
 rated_motor_speed
 motor_controller.h, 150
 RB_APPLY
 rb_tree.h, 163
 rb_apply
 rb_tree.c, 266
 RB_DUP
 rb_tree.h, 163
 RB_FIRST
 rb_tree.h, 163
 RB_ISEMPTY
 rb_tree.h, 163
 RB_MIN
 rb_tree.h, 163
 RB_MINIMAL
 rb_tree.h, 164
 RB_NIL
 rb_tree.h, 164
 RB_ROOT
 rb_tree.h, 164
 rb_tree.c
 checkBlackHeight, 264
 checkOrder, 265
 deleteRepair, 265
 destroyAllNodes, 265
 insertRepair, 265
 print, 266
 rb_apply, 266
 rbCheckBlackHeight, 266
 rbCheckOrder, 266
 rbCreate, 267
 rbDelete, 267
 rbDestroy, 267
 rbFind, 267
 rbInsert, 268
 rbPrint, 268
 rbSuccessor, 268
 rotateLeft, 268
 rotateRight, 269
 rb_tree.h
 BLACK, 163
 INORDER, 165
 POSTORDER, 165
 PREORDER, 165
 RB_APPLY, 163
 RB_DUP, 163
 RB_FIRST, 163
 RB_ISEMPTY, 163
 RB_MIN, 163
 RB_MINIMAL, 164
 RB_NIL, 164
 RB_ROOT, 164
 rbApplyNode, 165
 rbCheckBlackHeight, 165
 rbCheckOrder, 165
 rbCreate, 165
 rbDelete, 166
 rbDestroy, 166
 rbFind, 166
 rbInsert, 166
 rbnode, 164
 rbPrint, 167
 rbSuccessor, 167
 rbtraversal, 164
 RED, 164
 rbApplyNode
 rb_tree.h, 165
 rbCheckBlackHeight
 rb_tree.c, 266
 rb_tree.h, 165
 rbCheckOrder
 rb_tree.c, 266
 rb_tree.h, 165
 rbCreate
 rb_tree.c, 267
 rb_tree.h, 165
 rbDelete
 rb_tree.c, 267
 rb_tree.h, 166
 rbDestroy
 rb_tree.c, 267
 rb_tree.h, 166
 rbFind
 rb_tree.c, 267
 rb_tree.h, 166
 rbInsert
 rb_tree.c, 268
 rb_tree.h, 166
 rbnode, 74
 color, 74
 data, 74
 left, 75
 parent, 75
 rb_tree.h, 164
 right, 75

- rbPrint
 - rb_tree.c, 268
 - rb_tree.h, 167
- rbSuccessor
 - rb_tree.c, 268
 - rb_tree.h, 167
- rbtraversal
 - rb_tree.h, 164
- rbtree, 76
 - compare, 76
 - count, 76
 - destroy, 76
 - min, 77
 - nil, 77
 - print, 77
 - root, 77
- REALTIME_PRIORITY
 - State Engine API, 23
- recuperation_ramp
 - motor_controller.h, 151
- RED
 - rb_tree.h, 164
- Red_LED_GPIO_Port
 - main.h, 145
- Red_LED_Pin
 - main.h, 146
- refresh_OLED
 - oled.h, 157
- regen_rpm_cutoff
 - driving_loop_args, 68
- response_val
 - uv_scd_response, 90
- right
 - rbnode, 75
- root
 - rbtree, 77
- rotate_field_enable_not_present_norun
 - motor_controller.h, 152
- rotate_field_enable_not_present_run
 - motor_controller.h, 152
- rotateLeft
 - rb_tree.c, 268
- rotateRight
 - rb_tree.c, 269
- rotor_resistance
 - motor_controller.h, 150
- RxData
 - constants.c, 225
 - constants.h, 110
- RxHeader
 - constants.c, 226
 - constants.h, 110
- s_curve_speed_map
 - driving_loop.h, 121
- s_curve_torque_map
 - driving_loop.h, 121
- s_curve_torque_map_args, 78
 - a, 78
 - b, 78
 - c, 79
- driving_loop.h, 120
- safePtrRead
 - Utility Macros, 44
- safePtrWrite
 - Utility Macros, 44
- safety_touch_current
 - imd.h, 139
- safety_touch_energy
 - imd.h, 139
- SCD_active
 - State Engine, 12
- scd_handle_ptr
 - State Engine, 13
- semaphore
 - access_control_info, 57
- sender
 - uv_task_msg_t, 98
- Serial_number_0
 - imd.h, 138
- Serial_number_1
 - imd.h, 138
- Serial_number_2
 - imd.h, 138
- Serial_number_3
 - imd.h, 138
- serializeBigE16
 - Utility Macros, 44
- serializeBigE32
 - Utility Macros, 45
- serializeSmallE16
 - Utility Macros, 45
- serializeSmallE32
 - Utility Macros, 45
- setBits
 - Utility Macros, 45
- setup_extern_devices
 - uvfr_utils.c, 296
- setupDefaultSettings
 - uvfr_settings.c, 290
- slope
 - linear_torque_map_args, 72
- special_CPU_fault
 - motor_controller.h, 152
- specific_args
 - uv_init_task_args, 85
- speed_actual_resolution_limit
 - motor_controller.h, 152
- spi.c
 - HAL_SPI_MspDeInit, 270
 - HAL_SPI_MspInit, 270
 - hspi1, 271
 - MX_SPI1_Init, 270
- spi.h
 - hspi1, 168
 - MX_SPI1_Init, 168
- SRC_UVFR_SETTINGS_C_

- uvfr_settings.c, 289
- stack_size
 - uv_task_info, 93
- Start_Button_Input_EXTI_IRQn
 - main.h, 146
- Start_Button_Input_GPIO_Port
 - main.h, 146
- Start_Button_Input_Pin
 - main.h, 146
- startDaqSubTasks
 - daq.c, 229
- StartDefaultTask
 - freertos.c, 235
- StartDrivingLoop
 - driving_loop.c, 232
 - driving_loop.h, 121
- State Engine, 9
 - _next_svc_task_id, 11
 - _next_task_id, 11
 - _svc_task_register, 12
 - _task_register, 12
 - default_os_settings, 12
 - MAX_NUM_MANAGED_TASKS, 10
 - previous_state, 12
 - SCD_active, 12
 - scd_handle_ptr, 13
 - state_change_daemon_args, 10
 - state_change_queue, 13
 - svc_task_manager, 13
 - task_manager, 13
 - task_name_lut, 13
 - uvSVCTaskManager, 11
 - vehicle_state, 13
- State Engine API, 14
 - ABOVE_NORMAL, 23
 - BELOW_NORMAL, 23
 - changeVehicleState, 25
 - HIGH_PRIORITY, 23
 - IDLE_TASK_PRIORITY, 23
 - LOW_PRIORITY, 23
 - MEDIUM_PRIORITY, 23
 - PROGRAMMING, 25
 - REALTIME_PRIORITY, 23
 - task_management_info, 21
 - task_priority, 21, 23
 - task_status_block, 21
 - UV_BOOT, 25
 - UV_COULDNT_DELETE, 23
 - UV_COULDNT_SUSPEND, 23
 - UV_DRIVING, 25
 - UV_ERROR_STATE, 25
 - UV_HALT, 25
 - UV_INIT, 25
 - UV_KILL_CMD, 24
 - UV_LAUNCH_CONTROL, 25
 - UV_NO_CMD, 24
 - uv_os_settings, 22
 - UV_READY, 25
 - uv_scd_response, 22
 - uv_scd_response_e, 23
 - UV_SUCCESSFUL_DELETION, 23
 - UV_SUCCESSFUL_SUSPENSION, 23
 - UV_SUSPEND_CMD, 24
 - UV_SUSPENDED, 25
 - UV_TASK_AWAITING_DELETION, 16
 - uv_task_cmd, 22
 - uv_task_cmd_e, 24
 - UV_TASK_DEADLINE_FIRM, 16
 - UV_TASK_DEADLINE_HARD, 16
 - UV_TASK_DEADLINE_MASK, 16
 - UV_TASK_DEADLINE_NOT_ENFORCED, 16
 - UV_TASK_DEFER_DELETION, 17
 - UV_TASK_DELAYING, 17
 - UV_TASK_DELETED, 24
 - UV_TASK_DORMANT_SVC, 17
 - UV_TASK_ERR_IN_CHILD, 17
 - UV_TASK_GENERIC_SVC, 17
 - uv_task_info, 22
 - UV_TASK_IS_CHILD, 17
 - UV_TASK_IS_ORPHAN, 18
 - UV_TASK_IS_PARENT, 18
 - UV_TASK_LOG_MEM_USAGE, 18
 - UV_TASK_LOG_START_STOP_TIME, 18
 - UV_TASK_MANAGER_MASK, 18
 - UV_TASK_MISSION_CRITICAL, 18
 - UV_TASK_NOT_STARTED, 24
 - UV_TASK_PERIODIC_SVC, 19
 - UV_TASK_PRIO_INCREMENTATION, 19
 - UV_TASK_RUNNING, 24
 - UV_TASK_SCD_IGNORE, 19
 - UV_TASK_START_CMD, 24
 - uv_task_state_t, 24
 - uv_task_status, 22
 - UV_TASK_SUSPENDED, 24
 - UV_TASK_VEHICLE_APPLICATION, 19
 - UV_UNSAFE_STATE, 23
 - uv_vehicle_state, 22
 - uv_vehicle_state_t, 24
 - uvCreateTask, 26
 - uvDelInitStateEngine, 26
 - uvInitStateEngine, 26
 - uvStartStateMachine, 26
 - uvTaskDelay, 19
 - uvTaskDelayUntil, 20
 - uvTaskIsDelaying, 20
 - uvTaskResetDelayBit, 20
 - uvTaskResetDeletionBit, 20
 - uvTaskSetDelayBit, 21
 - uvTaskSetDeletionBit, 21
- State Engine Internals, 28
 - __uvPanic, 29
 - _stateChangeDaemon, 29
 - _uvValidateSpecificTask, 31
 - addTaskToTaskRegister, 31
 - killEmAll, 31
 - killSelf, 32

- processSCDMsg, [32](#)
- suspendSelf, [32](#)
- uvAbortTaskDeletion, [33](#)
- uvCreateServiceTask, [33](#)
- uvDeleteSVCTask, [33](#)
- uvDeleteTask, [34](#)
- uvGetTaskFromName, [34](#)
- uvGetTaskFromRTOSHandle, [34](#)
- uvInvokeSCD, [35](#)
- uvKillTaskViolently, [35](#)
- uvRestartSVCTask, [35](#)
- uvScheduleTaskDeletion, [35](#)
- uvSecureVehicle, [36](#)
- uvStartSVCTask, [36](#)
- uvStartTask, [36](#)
- uvSuspendSVCTask, [37](#)
- uvSuspendTask, [37](#)
- uvTaskCrashHandler, [38](#)
- uvTaskManager, [38](#)
- uvValidateManagedTasks, [38](#)
- state_change_daemon_args, [79](#)
 - meta_task_handle, [79](#)
 - State Engine, [10](#)
- state_change_queue
 - State Engine, [13](#)
- stator_leakage_inductance
 - motor_controller.h, [150](#)
- stator_resistance_ph_ph
 - motor_controller.h, [150](#)
- status
 - uv_init_task_response, [86](#)
- STM32_F407
 - uvfr_global_config.h, [195](#)
- STM32_H7xx
 - uvfr_global_config.h, [195](#)
- stm32f4xx_hal_conf.h
 - assert_param, [171](#)
 - DATA_CACHE_ENABLE, [172](#)
 - DP83848_PHY_ADDRESS, [172](#)
 - ETH_RX_BUF_SIZE, [172](#)
 - ETH_RXBUFNB, [172](#)
 - ETH_TX_BUF_SIZE, [172](#)
 - ETH_TXBUFNB, [172](#)
 - EXTERNAL_CLOCK_VALUE, [173](#)
 - HAL_ADC_MODULE_ENABLED, [173](#)
 - HAL_CAN_MODULE_ENABLED, [173](#)
 - HAL_CORTEX_MODULE_ENABLED, [173](#)
 - HAL_DMA_MODULE_ENABLED, [173](#)
 - HAL_EXTI_MODULE_ENABLED, [173](#)
 - HAL_FLASH_MODULE_ENABLED, [174](#)
 - HAL_GPIO_MODULE_ENABLED, [174](#)
 - HAL_MODULE_ENABLED, [174](#)
 - HAL_PWR_MODULE_ENABLED, [174](#)
 - HAL_RCC_MODULE_ENABLED, [174](#)
 - HAL_SPI_MODULE_ENABLED, [174](#)
 - HAL_TIM_MODULE_ENABLED, [175](#)
 - HSE_STARTUP_TIMEOUT, [175](#)
 - HSE_VALUE, [175](#)
 - HSI_VALUE, [175](#)
 - INSTRUCTION_CACHE_ENABLE, [175](#)
 - LSE_STARTUP_TIMEOUT, [176](#)
 - LSE_VALUE, [176](#)
 - LSI_VALUE, [176](#)
 - MAC_ADDR0, [176](#)
 - MAC_ADDR1, [176](#)
 - MAC_ADDR2, [177](#)
 - MAC_ADDR3, [177](#)
 - MAC_ADDR4, [177](#)
 - MAC_ADDR5, [177](#)
 - PHY_AUTONEGO_COMPLETE, [177](#)
 - PHY_AUTONEGOTIATION, [177](#)
 - PHY_BCR, [178](#)
 - PHY_BSR, [178](#)
 - PHY_CONFIG_DELAY, [178](#)
 - PHY_DUPLEX_STATUS, [178](#)
 - PHY_FULLDUPLEX_100M, [178](#)
 - PHY_FULLDUPLEX_10M, [179](#)
 - PHY_HALFDUPLEX_100M, [179](#)
 - PHY_HALFDUPLEX_10M, [179](#)
 - PHY_ISOLATE, [179](#)
 - PHY_JABBER_DETECTION, [179](#)
 - PHY_LINKED_STATUS, [180](#)
 - PHY_LOOPBACK, [180](#)
 - PHY_POWERDOWN, [180](#)
 - PHY_READ_TO, [180](#)
 - PHY_RESET, [180](#)
 - PHY_RESET_DELAY, [181](#)
 - PHY_RESTART_AUTONEGOTIATION, [181](#)
 - PHY_SPEED_STATUS, [181](#)
 - PHY_SR, [181](#)
 - PHY_WRITE_TO, [181](#)
 - PREFETCH_ENABLE, [182](#)
 - TICK_INT_PRIORITY, [182](#)
 - USE_HAL_ADC_REGISTER_CALLBACKS, [182](#)
 - USE_HAL_CAN_REGISTER_CALLBACKS, [182](#)
 - USE_HAL_CEC_REGISTER_CALLBACKS, [182](#)
 - USE_HAL_Cryp_REGISTER_CALLBACKS, [182](#)
 - USE_HAL_DAC_REGISTER_CALLBACKS, [183](#)
 - USE_HAL_DCMI_REGISTER_CALLBACKS, [183](#)
 - USE_HAL_DFSDM_REGISTER_CALLBACKS, [183](#)
 - USE_HAL_DMA2D_REGISTER_CALLBACKS, [183](#)
 - USE_HAL_DSI_REGISTER_CALLBACKS, [183](#)
 - USE_HAL_ETH_REGISTER_CALLBACKS, [183](#)
 - USE_HAL_FMPI2C_REGISTER_CALLBACKS, [184](#)
 - USE_HAL_FMPMBUS_REGISTER_CALLBACKS, [184](#)
 - USE_HAL_HASH_REGISTER_CALLBACKS, [184](#)
 - USE_HAL_HCD_REGISTER_CALLBACKS, [184](#)
 - USE_HAL_I2C_REGISTER_CALLBACKS, [184](#)
 - USE_HAL_I2S_REGISTER_CALLBACKS, [184](#)
 - USE_HAL_IRDA_REGISTER_CALLBACKS, [185](#)
 - USE_HAL_LPTIM_REGISTER_CALLBACKS, [185](#)
 - USE_HAL_LTDC_REGISTER_CALLBACKS, [185](#)

- USE_HAL_MMC_REGISTER_CALLBACKS, [185](#)
- USE_HAL_NAND_REGISTER_CALLBACKS, [185](#)
- USE_HAL_NOR_REGISTER_CALLBACKS, [185](#)
- USE_HAL_PCCARD_REGISTER_CALLBACKS, [186](#)
- USE_HAL_PCD_REGISTER_CALLBACKS, [186](#)
- USE_HAL_QSPI_REGISTER_CALLBACKS, [186](#)
- USE_HAL_RNG_REGISTER_CALLBACKS, [186](#)
- USE_HAL_RTC_REGISTER_CALLBACKS, [186](#)
- USE_HAL_SAI_REGISTER_CALLBACKS, [186](#)
- USE_HAL_SD_REGISTER_CALLBACKS, [187](#)
- USE_HAL_SDRAM_REGISTER_CALLBACKS, [187](#)
- USE_HAL_SMARTCARD_REGISTER_CALLBACKS, [187](#)
- USE_HAL_SMBUS_REGISTER_CALLBACKS, [187](#)
- USE_HAL_SPDIFRX_REGISTER_CALLBACKS, [187](#)
- USE_HAL_SPI_REGISTER_CALLBACKS, [187](#)
- USE_HAL_SRAM_REGISTER_CALLBACKS, [188](#)
- USE_HAL_TIM_REGISTER_CALLBACKS, [188](#)
- USE_HAL_UART_REGISTER_CALLBACKS, [188](#)
- USE_HAL_USART_REGISTER_CALLBACKS, [188](#)
- USE_HAL_WWDG_REGISTER_CALLBACKS, [188](#)
- USE_RTOS, [188](#)
- USE_SPI_CRC, [189](#)
- VDD_VALUE, [189](#)
- stm32f4xx_hal_msp.c
 - HAL_Msplnit, [271](#)
- stm32f4xx_hal_timebase_tim.c
 - HAL_InitTick, [272](#)
 - HAL_ResumeTick, [273](#)
 - HAL_SuspendTick, [273](#)
 - htim1, [273](#)
- stm32f4xx_it.c
 - BusFault_Handler, [275](#)
 - CAN2_RX0_IRQHandler, [275](#)
 - CAN2_RX1_IRQHandler, [275](#)
 - CAN2_TX_IRQHandler, [275](#)
 - DebugMon_Handler, [276](#)
 - DMA2_Stream0_IRQHandler, [276](#)
 - EXTI0_IRQHandler, [276](#)
 - HardFault_Handler, [276](#)
 - hcan2, [278](#)
 - hdma_adc1, [278](#)
 - htim1, [278](#)
 - MemManage_Handler, [277](#)
 - NMI_Handler, [277](#)
 - TIM1_UP_TIM10_IRQHandler, [277](#)
 - UsageFault_Handler, [277](#)
- stm32f4xx_it.h
 - BusFault_Handler, [190](#)
 - CAN2_RX0_IRQHandler, [190](#)
 - CAN2_RX1_IRQHandler, [190](#)
 - CAN2_TX_IRQHandler, [190](#)
 - DebugMon_Handler, [191](#)
 - DMA2_Stream0_IRQHandler, [191](#)
 - EXTI0_IRQHandler, [191](#)
 - HardFault_Handler, [191](#)
 - MemManage_Handler, [192](#)
 - NMI_Handler, [192](#)
 - TIM1_UP_TIM10_IRQHandler, [192](#)
 - UsageFault_Handler, [192](#)
- Stm32f4xx_system, [49](#)
- STM32F4xx_System_Private_Defines, [52](#)
- STM32F4xx_System_Private_FunctionPrototypes, [55](#)
- STM32F4xx_System_Private_Functions, [56](#)
 - SystemCoreClockUpdate, [56](#)
 - SystemInit, [56](#)
- STM32F4xx_System_Private_Includes, [50](#)
 - HSE_VALUE, [50](#)
 - HSI_VALUE, [50](#)
- STM32F4xx_System_Private_Macros, [53](#)
- STM32F4xx_System_Private_TypesDefinitions, [51](#)
- STM32F4xx_System_Private_Variables, [54](#)
 - AHBPrescTable, [54](#)
 - APBPrescTable, [54](#)
 - SystemCoreClock, [54](#)
- stopDaqSubTasks
 - daq.c, [229](#)
- suspendSelf
 - State Engine Internals, [32](#)
- suspension_states
 - uv_task_info, [93](#)
- svc_task_manager
 - State Engine, [13](#)
- svc_task_manager_period
 - uv_os_settings, [89](#)
- SVC_TASK_MAX_CHECKIN_PERIOD
 - uvfr_state_engine.h, [202](#)
- syscalls.c
 - __attribute__, [279](#)
 - __io_getchar, [280](#)
 - __io_putchar, [280](#)
 - _close, [280](#)
 - _execve, [280](#)
 - _exit, [280](#)
 - _fork, [281](#)
 - _fstat, [281](#)
 - _getpid, [281](#)
 - _isatty, [281](#)
 - _kill, [281](#)
 - _link, [282](#)
 - _lseek, [282](#)
 - _open, [282](#)
 - _stat, [282](#)
 - _times, [282](#)
 - _unlink, [283](#)
 - _wait, [283](#)
 - environ, [283](#)
 - initialise_monitor_handles, [283](#)
- systemem.c
 - __sbrk_heap_end, [285](#)

- [_sbrk](#), [284](#)
- SystemClock_Config
 - [main.c](#), [254](#)
- SystemCoreClock
 - STM32F4xx_System_Private_Variables, [54](#)
- SystemCoreClockUpdate
 - STM32F4xx_System_Private_Functions, [56](#)
- SystemInit
 - STM32F4xx_System_Private_Functions, [56](#)
- task_args
 - [uv_task_info](#), [93](#)
- task_flags
 - [uv_task_info](#), [94](#)
- task_function
 - [uv_task_info](#), [94](#)
- task_handle
 - [task_management_info](#), [80](#)
 - [uv_task_info](#), [94](#)
- task_high_water_mark
 - [task_status_block](#), [81](#)
- task_id
 - [uv_task_info](#), [95](#)
- task_management_info, [80](#)
 - [parent_msg_queue](#), [80](#)
 - State Engine API, [21](#)
 - [task_handle](#), [80](#)
- task_manager
 - State Engine, [13](#)
- task_manager_period
 - [uv_os_settings](#), [89](#)
- task_name
 - [uv_task_info](#), [95](#)
- task_name_lut
 - State Engine, [13](#)
- task_period
 - [uv_task_info](#), [95](#)
- task_priority
 - State Engine API, [21](#), [23](#)
 - [uv_task_info](#), [95](#)
- task_report_time
 - [task_status_block](#), [81](#)
- task_rx_mailbox
 - [uv_task_info](#), [96](#)
- task_state
 - [uv_task_info](#), [96](#)
- task_status_block, [81](#)
 - State Engine API, [21](#)
 - [task_high_water_mark](#), [81](#)
 - [task_report_time](#), [81](#)
- temp_monitoring.c
 - [initTempMonitor](#), [286](#)
 - [tempMonitorTask](#), [287](#)
- temp_monitoring.h
 - [initTempMonitor](#), [193](#)
 - [tempMonitorTask](#), [193](#)
- temp_sensor_pt1
 - [motor_controller.h](#), [153](#)
- temp_sensor_pt2
 - [motor_controller.h](#), [153](#)
- temp_sensor_pt3
 - [motor_controller.h](#), [153](#)
- temp_sensor_pt4
 - [motor_controller.h](#), [153](#)
- Temperature
 - [imd.h](#), [139](#)
- tempMonitorTask
 - [temp_monitoring.c](#), [287](#)
 - [temp_monitoring.h](#), [193](#)
- throttle_daq_to_preserve_performance
 - [daq_loop_args](#), [62](#)
- TICK_INT_PRIORITY
 - [stm32f4xx_hal_conf.h](#), [182](#)
- tim.c
 - HAL_TIM_Base_MspDeInit, [288](#)
 - HAL_TIM_Base_MspInit, [288](#)
 - [htim3](#), [289](#)
 - MX_TIM3_Init, [288](#)
- tim.h
 - [htim3](#), [194](#)
 - MX_TIM3_Init, [194](#)
- TIM1_UP_TIM10_IRQHandler
 - [stm32f4xx_it.c](#), [277](#)
 - [stm32f4xx_it.h](#), [192](#)
- time_constant_rotor
 - [motor_controller.h](#), [150](#)
- time_constant_stator
 - [motor_controller.h](#), [150](#)
- time_sent
 - [uv_task_msg_t](#), [98](#)
- tmi
 - [uv_task_info](#), [96](#)
- todo1
 - [motor_controller.h](#), [149](#)
- todo6969
 - [motor_controller.h](#), [149](#)
- Touch_energy_fault
 - [imd.h](#), [139](#)
- treenode
 - [daq_child_task](#), [59](#)
- tripzone_glitch_detected
 - [motor_controller.h](#), [152](#)
- true
 - Utility Macros, [46](#)
- Tx_msg_queue
 - [can.c](#), [225](#)
- TxData
 - [constants.c](#), [226](#)
 - [constants.h](#), [111](#)
 - [uvfr_utils.c](#), [299](#)
- TxHeader
 - [constants.c](#), [226](#)
 - [constants.h](#), [111](#)
- TxMailbox
 - [constants.c](#), [226](#)
 - [constants.h](#), [111](#)
- type

- daq_datapoint, [60](#)
- Update_Batt_Temp
 - dash.c, [230](#)
 - dash.h, [116](#)
- Update_RPM
 - dash.c, [230](#)
 - dash.h, [116](#)
- Update_State_Of_Charge
 - dash.c, [230](#)
 - dash.h, [117](#)
- updateRunningTasks
 - uvfr_state_engine.h, [203](#)
- Uptime_counter
 - imd.h, [138](#)
- UsageFault_Handler
 - stm32f4xx_it.c, [277](#)
 - stm32f4xx_it.h, [192](#)
- use_default_settings
 - uv_init_struct, [84](#)
- USE_HAL_ADC_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [182](#)
- USE_HAL_CAN_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [182](#)
- USE_HAL_CEC_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [182](#)
- USE_HAL_Cryp_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [182](#)
- USE_HAL_DAC_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [183](#)
- USE_HAL_DCMI_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [183](#)
- USE_HAL_DFSDM_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [183](#)
- USE_HAL_DMA2D_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [183](#)
- USE_HAL_DSI_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [183](#)
- USE_HAL_ETH_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [183](#)
- USE_HAL_FMPI2C_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [184](#)
- USE_HAL_FMPSPMBUS_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [184](#)
- USE_HAL_HASH_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [184](#)
- USE_HAL_HCD_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [184](#)
- USE_HAL_I2C_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [184](#)
- USE_HAL_I2S_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [184](#)
- USE_HAL_IRDA_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [185](#)
- USE_HAL_LPTIM_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [185](#)
- USE_HAL_LTDC_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [185](#)
- USE_HAL_MMC_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [185](#)
- USE_HAL_NAND_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [185](#)
- USE_HAL_NOR_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [185](#)
- USE_HAL_PCCARD_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [186](#)
- USE_HAL_PCD_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [186](#)
- USE_HAL_QSPI_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [186](#)
- USE_HAL_RNG_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [186](#)
- USE_HAL_RTC_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [186](#)
- USE_HAL_SAI_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [186](#)
- USE_HAL_SD_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [187](#)
- USE_HAL_SDRAM_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [187](#)
- USE_HAL_SMARTCARD_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [187](#)
- USE_HAL_SMBUS_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [187](#)
- USE_HAL_SPDIFRX_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [187](#)
- USE_HAL_SPI_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [187](#)
- USE_HAL_SRAM_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [188](#)
- USE_HAL_TIM_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [188](#)
- USE_HAL_UART_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [188](#)
- USE_HAL_USART_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [188](#)
- USE_HAL_WWDG_REGISTER_CALLBACKS
 - stm32f4xx_hal_conf.h, [188](#)
- USE_OLED_DEBUG
 - uvfr_utils.h, [207](#)
- USE_OS_MEM_MGMT
 - uvfr_global_config.h, [195](#)
- USE_RTOS
 - stm32f4xx_hal_conf.h, [188](#)
- USE_SPI_CRC
 - stm32f4xx_hal_conf.h, [189](#)
- Utility Macros, [41](#)
 - _BV, [41](#)
 - _BV_16, [41](#)
 - _BV_32, [42](#)
 - _BV_8, [42](#)
 - deserializeBigE16, [42](#)
 - deserializeBigE32, [42](#)
 - deserializeSmallE16, [42](#)
 - deserializeSmallE32, [43](#)
 - endianSwap, [43](#)
 - endianSwap16, [43](#)

- endianSwap32, [43](#)
- endianSwap8, [43](#)
- false, [44](#)
- isPowerOfTwo, [44](#)
- safePtrRead, [44](#)
- safePtrWrite, [44](#)
- serializeBigE16, [44](#)
- serializeBigE32, [45](#)
- serializeSmallE16, [45](#)
- serializeSmallE32, [45](#)
- setBits, [45](#)
- true, [46](#)
- UV19_PDU
 - uvfr_global_config.h, [196](#)
- UV_ABORTED
 - uvfr_utils.h, [212](#)
- UV_ASSIGN_TASK
 - uvfr_utils.h, [212](#)
- UV_BINARY_SEMAPHORE
 - uvfr_utils.h, [211](#)
- uv_binary_semaphore_info, [81](#)
 - handle, [82](#)
- UV_BOOT
 - State Engine API, [25](#)
- UV_CAN1
 - uvfr_utils.h, [207](#)
- UV_CAN2
 - uvfr_utils.h, [207](#)
- UV_CAN_CHANNEL_MASK
 - uvfr_utils.h, [207](#)
- UV_CAN_DYNAMIC_MEM
 - uvfr_utils.h, [208](#)
- UV_CAN_EXTENDED_ID
 - uvfr_utils.h, [208](#)
- uv_CAN_msg, [82](#)
 - can.h, [108](#)
 - data, [82](#)
 - dlc, [83](#)
 - flags, [83](#)
 - msg_id, [83](#)
 - uvfr_utils.h, [209](#)
- UV_COMMAND_ACKNOWLEDGEMENT
 - uvfr_utils.h, [212](#)
- UV_COULDNT_DELETE
 - State Engine API, [23](#)
- UV_COULDNT_SUSPEND
 - State Engine API, [23](#)
- UV_DOUBLE
 - daq.h, [114](#)
- UV_DRIVING
 - State Engine API, [25](#)
- uv_driving_mode_t
 - uvfr_utils.h, [211](#)
- UV_DUMB_FLAG
 - uvfr_utils.h, [211](#)
- UV_ERROR
 - uvfr_utils.h, [212](#)
- UV_ERROR_REPORT
 - uvfr_utils.h, [212](#)
- UV_ERROR_STATE
 - State Engine API, [25](#)
- uv_ext_device_id
 - uvfr_utils.h, [209](#)
- uv_external_device
 - uvfr_utils.h, [211](#)
- UV_FLOAT
 - daq.h, [114](#)
- UV_HALT
 - State Engine API, [25](#)
- UV_INIT
 - State Engine API, [25](#)
- uv_init_struct, [83](#)
 - use_default_settings, [84](#)
 - uvfr_utils.h, [209](#)
- uv_init_task_args, [84](#)
 - init_info_queue, [85](#)
 - meta_task_handle, [85](#)
 - specific_args, [85](#)
 - uvfr_utils.h, [209](#)
- uv_init_task_response, [85](#)
 - device, [86](#)
 - errmsg, [86](#)
 - nchar, [86](#)
 - status, [86](#)
 - uvfr_utils.h, [209](#)
- UV_INT16
 - daq.h, [114](#)
- UV_INT32
 - daq.h, [114](#)
- UV_INT8
 - daq.h, [114](#)
- uv_internal_params, [87](#)
 - e_code, [87](#)
 - init_params, [87](#)
 - peripheral_status, [87](#)
 - uvfr_utils.h, [209](#)
 - vehicle_settings, [87](#)
- UV_INVALID_MSG
 - uvfr_utils.h, [212](#)
- UV_KILL_CMD
 - State Engine API, [24](#)
- UV_LAUNCH_CONTROL
 - State Engine API, [25](#)
- UV_MALLOC_LIMIT
 - uvfr_global_config.h, [196](#)
- uv_msg_type
 - uvfr_utils.h, [210](#)
- uv_msg_type_t
 - uvfr_utils.h, [212](#)
- UV_Mutex
 - uvfr_utils.h, [211](#)
- uv_mutex_info, [88](#)
 - handle, [88](#)
- UV_NO_CMD
 - State Engine API, [24](#)
- UV_NONE

- uvfr_utils.h, 211
- UV_OK
 - uvfr_utils.h, 212
- uv_os_settings, 88
 - max_svc_task_period, 89
 - max_task_period, 89
 - min_task_period, 89
 - State Engine API, 22
 - svc_task_manager_period, 89
 - task_manager_period, 89
- UV_PARAM_READY
 - uvfr_utils.h, 212
- UV_PARAM_REQUEST
 - uvfr_utils.h, 212
- UV_RAW_DATA_TRANSFER
 - uvfr_utils.h, 212
- UV_READY
 - State Engine API, 25
- UV_SC_COMMAND
 - uvfr_utils.h, 212
- uv_scd_response, 90
 - meta_id, 90
 - response_val, 90
 - State Engine API, 22
- uv_scd_response_e
 - State Engine API, 23
- UV_SEMAPHORE
 - uvfr_utils.h, 211
- uv_semaphore_info, 91
 - handle, 91
- uv_status
 - can.h, 108
 - uvfr_state_engine.h, 202
 - uvfr_utils.h, 210
- uv_status_t
 - uvfr_utils.h, 212
- UV_STRING
 - daq.h, 114
- UV_SUCCESSFUL_DELETION
 - State Engine API, 23
- UV_SUCCESSFUL_SUSPENSION
 - State Engine API, 23
- UV_SUSPEND_CMD
 - State Engine API, 24
- UV_SUSPENDED
 - State Engine API, 25
- UV_TASK_AWAITING_DELETION
 - State Engine API, 16
- uv_task_cmd
 - State Engine API, 22
 - uvfr_utils.h, 210
- uv_task_cmd_e
 - State Engine API, 24
- UV_TASK_DEADLINE_FIRM
 - State Engine API, 16
- UV_TASK_DEADLINE_HARD
 - State Engine API, 16
- UV_TASK_DEADLINE_MASK
 - State Engine API, 16
- UV_TASK_DEADLINE_NOT_ENFORCED
 - State Engine API, 16
- UV_TASK_DEFER_DELETION
 - State Engine API, 17
- UV_TASK_DELAYING
 - State Engine API, 17
- UV_TASK_DELETE_COMMAND
 - uvfr_utils.h, 212
- UV_TASK_DELETED
 - State Engine API, 24
- UV_TASK_DORMANT_SVC
 - State Engine API, 17
- UV_TASK_ERR_IN_CHILD
 - State Engine API, 17
- UV_TASK_GENERIC_SVC
 - State Engine API, 17
- uv_task_id
 - uvfr_state_engine.h, 202
 - uvfr_utils.h, 210
- uv_task_info, 91
 - active_states, 92
 - cmd_data, 92
 - deletion_delay, 92
 - deletion_states, 92
 - parent, 93
 - stack_size, 93
 - State Engine API, 22
 - suspension_states, 93
 - task_args, 93
 - task_flags, 94
 - task_function, 94
 - task_handle, 94
 - task_id, 95
 - task_name, 95
 - task_period, 95
 - task_priority, 95
 - task_rx_mailbox, 96
 - task_state, 96
 - tmi, 96
- UV_TASK_IS_CHILD
 - State Engine API, 17
- UV_TASK_IS_ORPHAN
 - State Engine API, 18
- UV_TASK_IS_PARENT
 - State Engine API, 18
- UV_TASK_LOG_MEM_USAGE
 - State Engine API, 18
- UV_TASK_LOG_START_STOP_TIME
 - State Engine API, 18
- UV_TASK_MANAGER_MASK
 - State Engine API, 18
- UV_TASK_MISSION_CRITICAL
 - State Engine API, 18
- uv_task_msg
 - uvfr_utils.h, 210
- uv_task_msg_t, 97
 - intended_recipient, 97

- message_size, [97](#)
- message_type, [97](#)
- msg_contents, [98](#)
- sender, [98](#)
- time_sent, [98](#)
- UV_TASK_NOT_STARTED
 - State Engine API, [24](#)
- UV_TASK_PERIODIC_SVC
 - State Engine API, [19](#)
- UV_TASK_PRIO_INCREMENTATION
 - State Engine API, [19](#)
- UV_TASK_RUNNING
 - State Engine API, [24](#)
- UV_TASK_SCD_IGNORE
 - State Engine API, [19](#)
- UV_TASK_START_CMD
 - State Engine API, [24](#)
- UV_TASK_START_COMMAND
 - uvfr_utils.h, [212](#)
- uv_task_state_t
 - State Engine API, [24](#)
- uv_task_status
 - State Engine API, [22](#)
- UV_TASK_STATUS_REPORT
 - uvfr_utils.h, [212](#)
- UV_TASK_SUSPEND_COMMAND
 - uvfr_utils.h, [212](#)
- UV_TASK_SUSPENDED
 - State Engine API, [24](#)
- UV_TASK_VEHICLE_APPLICATION
 - State Engine API, [19](#)
- uv_timespan_ms
 - uvfr_state_engine.h, [202](#)
 - uvfr_utils.h, [210](#)
- UV_UINT16
 - daq.h, [114](#)
- UV_UINT32
 - daq.h, [114](#)
- UV_UINT8
 - daq.h, [114](#)
- UV_UNSAFE_STATE
 - State Engine API, [23](#)
- UV_UTILS_SRC_IMPLIMENTATION
 - uvfr_utils.c, [294](#)
- uv_vehicle_settings, [98](#)
 - bms_settings, [99](#)
 - daq_settings, [99](#)
 - driving_loop_settings, [99](#)
 - imd_settings, [99](#)
 - is_default, [99](#)
 - mc_settings, [100](#)
 - os_settings, [100](#)
 - pdu_settings, [100](#)
 - uvfr_settings.h, [197](#)
- uv_vehicle_state
 - State Engine API, [22](#)
- uv_vehicle_state_t
 - State Engine API, [24](#)
- UV_WAKEUP
 - uvfr_utils.h, [212](#)
- UV_WARNING
 - uvfr_utils.h, [212](#)
- uvAbortTaskDeletion
 - State Engine Internals, [33](#)
- uvCreateServiceTask
 - State Engine Internals, [33](#)
- uvCreateTask
 - State Engine API, [26](#)
- uvDelnitStateEngine
 - State Engine API, [26](#)
- uvDeleteSVCTask
 - State Engine Internals, [33](#)
- uvDeleteTask
 - State Engine Internals, [34](#)
- UVFR Utilities, [40](#)
- UVFR Vehicle Commands, [47](#)
- uvfr_global_config.h
 - ECUMASTER_PMU, [195](#)
 - STM32_F407, [195](#)
 - STM32_H7xx, [195](#)
 - USE_OS_MEM_MGMT, [195](#)
 - UV19_PDU, [196](#)
 - UV_MALLOC_LIMIT, [196](#)
- uvfr_settings.c
 - current_vehicle_settings, [291](#)
 - nukeSettings, [290](#)
 - setupDefaultSettings, [290](#)
 - SRC_UVFR_SETTINGS_C_, [289](#)
 - uvSettingsInit, [290](#)
 - uvSettingsProgrammerTask, [290](#)
- uvfr_settings.h
 - current_vehicle_settings, [198](#)
 - ENABLE_FLASH_SETTINGS, [197](#)
 - nukeSettings, [197](#)
 - uv_vehicle_settings, [197](#)
 - uvSettingsInit, [197](#)
 - veh_gen_info, [197](#)
- uvfr_state_engine.c
 - UVFR_STATE_MACHINE_IMPLIMENTATION, [293](#)
- uvfr_state_engine.h
 - _LONGEST_SC_TIME, [201](#)
 - _SC_DAEMON_PERIOD, [201](#)
 - _UV_DEFAULT_TASK_INSTANCES, [201](#)
 - _UV_DEFAULT_TASK_PERIOD, [201](#)
 - _UV_DEFAULT_TASK_STACK_SIZE, [202](#)
 - _UV_MIN_TASK_PERIOD, [202](#)
 - getSVCTaskID, [203](#)
 - SVC_TASK_MAX_CHECKIN_PERIOD, [202](#)
 - updateRunningTasks, [203](#)
 - uv_status, [202](#)
 - uv_task_id, [202](#)
 - uv_timespan_ms, [202](#)
 - uvGetTaskByld, [203](#)
 - uvRegisterTask, [203](#)
- UVFR_STATE_MACHINE_IMPLIMENTATION
 - uvfr_state_engine.c, [293](#)

- uvfr_utils.c
 - __uvFreeCriticalSection, 295
 - __uvFreeOS, 295
 - __uvInitPanic, 295
 - __uvMallocCriticalSection, 295
 - __uvMallocOS, 296
 - init_task_handle, 299
 - setup_extern_devices, 296
 - TxData, 299
 - UV_UTILS_SRC_IMPLIMENTATION, 294
 - uvInit, 296
 - uvIsPTRValid, 298
 - uvUtilsReset, 299
- uvfr_utils.h
 - __uvInitPanic, 213
 - accel, 211
 - access_control_info, 208
 - access_control_t, 211
 - access_control_type, 208
 - BMS, 212
 - bool, 208
 - econ, 211
 - global_context, 215
 - IMD, 212
 - INIT_CHECK_PERIOD, 207
 - limp, 211
 - MAX_INIT_TIME, 207
 - MOTOR_CONTROLLER, 212
 - normal, 211
 - p_status, 208
 - PDU, 212
 - USE_OLED_DEBUG, 207
 - UV_ABORTED, 212
 - UV_ASSIGN_TASK, 212
 - UV_BINARY_SEMAPHORE, 211
 - UV_CAN1, 207
 - UV_CAN2, 207
 - UV_CAN_CHANNEL_MASK, 207
 - UV_CAN_DYNAMIC_MEM, 208
 - UV_CAN_EXTENDED_ID, 208
 - uv_CAN_msg, 209
 - UV_COMMAND_ACKNOWLEDGEMENT, 212
 - uv_driving_mode_t, 211
 - UV_DUMB_FLAG, 211
 - UV_ERROR, 212
 - UV_ERROR_REPORT, 212
 - uv_ext_device_id, 209
 - uv_external_device, 211
 - uv_init_struct, 209
 - uv_init_task_args, 209
 - uv_init_task_response, 209
 - uv_internal_params, 209
 - UV_INVALID_MSG, 212
 - uv_msg_type, 210
 - uv_msg_type_t, 212
 - UV_Mutex, 211
 - UV_NONE, 211
 - UV_OK, 212
 - UV_PARAM_READY, 212
 - UV_PARAM_REQUEST, 212
 - UV_RAW_DATA_TRANSFER, 212
 - UV_SC_COMMAND, 212
 - UV_SEMAPHORE, 211
 - uv_status, 210
 - uv_status_t, 212
 - uv_task_cmd, 210
 - UV_TASK_DELETE_COMMAND, 212
 - uv_task_id, 210
 - uv_task_msg, 210
 - UV_TASK_START_COMMAND, 212
 - UV_TASK_STATUS_REPORT, 212
 - UV_TASK_SUSPEND_COMMAND, 212
 - uv_timespan_ms, 210
 - UV_WAKEUP, 212
 - UV_WARNING, 212
 - uvInit, 213
 - uvIsPTRValid, 215
- uvfr_vehicle_commands.h
 - uvHonkHorn, 216
 - uvOpenSDC, 216
 - uvStartCoolantPump, 217
 - uvStartFans, 217
 - uvStopCoolantPump, 217
 - uvStopFans, 217
- uvGetTaskById
 - uvfr_state_engine.h, 203
- uvGetTaskFromName
 - State Engine Internals, 34
- uvGetTaskFromRTOSHandle
 - State Engine Internals, 34
- uvHonkHorn
 - uvfr_vehicle_commands.h, 216
- uvInit
 - uvfr_utils.c, 296
 - uvfr_utils.h, 213
- uvInitStateEngine
 - State Engine API, 26
- uvInvokeSCD
 - State Engine Internals, 35
- uvIsPTRValid
 - uvfr_utils.c, 298
 - uvfr_utils.h, 215
- uvKillTaskViolently
 - State Engine Internals, 35
- uvOpenSDC
 - uvfr_vehicle_commands.h, 216
- uvRegisterTask
 - uvfr_state_engine.h, 203
- uvRestartSVCTask
 - State Engine Internals, 35
- uvScheduleTaskDeletion
 - State Engine Internals, 35
- uvSecureVehicle
 - State Engine Internals, 36
- uvSendCanMSG
 - can.c, 224

- can.h, 109
- uvSettingsInit
 - uvfr_settings.c, 290
 - uvfr_settings.h, 197
- uvSettingsProgrammerTask
 - uvfr_settings.c, 290
- uvStartCoolantPump
 - uvfr_vehicle_commands.h, 217
- uvStartFans
 - uvfr_vehicle_commands.h, 217
- uvStartStateMachine
 - State Engine API, 26
- uvStartSVCTask
 - State Engine Internals, 36
- uvStartTask
 - State Engine Internals, 36
- uvStopCoolantPump
 - uvfr_vehicle_commands.h, 217
- uvStopFans
 - uvfr_vehicle_commands.h, 217
- uvSuspendSVCTask
 - State Engine Internals, 37
- uvSuspendTask
 - State Engine Internals, 37
- uvSVCTaskManager
 - State Engine, 11
- uvTaskCrashHandler
 - State Engine Internals, 38
- uvTaskDelay
 - State Engine API, 19
- uvTaskDelayUntil
 - State Engine API, 20
- uvTaskIsDelaying
 - State Engine API, 20
- uvTaskManager
 - State Engine Internals, 38
- uvTaskResetDelayBit
 - State Engine API, 20
- uvTaskResetDeletionBit
 - State Engine API, 20
- uvTaskSetDelayBit
 - State Engine API, 21
- uvTaskSetDeletionBit
 - State Engine API, 21
- uvUtilsReset
 - uvfr_utils.c, 299
- uvValidateManagedTasks
 - State Engine Internals, 38
- vApplicationGetIdleTaskMemory
 - freertos.c, 235
- vApplicationGetTimerTaskMemory
 - freertos.c, 236
- vApplicationIdleHook
 - freertos.c, 236
- vApplicationMallocFailedHook
 - freertos.c, 236
- vApplicationStackOverflowHook
 - freertos.c, 236
- vApplicationTickHook
 - freertos.c, 236
- Vb_hi_res
 - imd.h, 138
- VDD_VALUE
 - stm32f4xx_hal_conf.h, 189
- veh_gen_info, 100
 - uvfr_settings.h, 197
- vehicle_settings
 - uv_internal_params, 87
- vehicle_state
 - State Engine, 13
- Version_0
 - imd.h, 138
- Version_1
 - imd.h, 138
- Version_2
 - imd.h, 138
- Vexc_hi_res
 - imd.h, 138
- Vn_hi_res
 - imd.h, 138
- voltages_Vp_and_Vn
 - imd.h, 139
- Vout_saturation_max_limit
 - motor_controller.h, 152
- Vp_hi_res
 - imd.h, 138
- vPortSVCHandler
 - FreeRTOSConfig.h, 135
- Vpwr_hi_res
 - imd.h, 138
- wait
 - oled.h, 157
- warning_5
 - motor_controller.h, 152
- warning_9
 - motor_controller.h, 152
- watchdog_reset
 - motor_controller.h, 152
- xIdleStack
 - freertos.c, 237
- xIdleTaskTCBBuffer
 - freertos.c, 238
- xPortPendSVHandler
 - FreeRTOSConfig.h, 135
- xPortSysTickHandler
 - FreeRTOSConfig.h, 135
- xTimerStack
 - freertos.c, 238
- xTimerTaskTCBBuffer
 - freertos.c, 238