# 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 compareTaskByName() . . . 11

5.1.4.2 uvSVCTaskManager() . . . 11

5.1.5 Variable Documentation . . . 11

5.1.5.1 _next_svc_task_id . . . 12

5.1.5.2 _next_task_id . . . 12

5.1.5.3 _task_register . . . 12

5.1.5.4 default_os_settings . . . 12

5.1.5.5 previous_state . . . 13

5.1.5.6 SCD_active . . . 13

5.1.5.7 scd_handle_ptr . . . 13

5.1.5.8 state_change_queue . . . 13

5.1.5.9 svc_task_manager . . . 13

5.1.5.10 task_manager . . . 14

5.1.5.11 task_name_lut . . . 14

5.1.5.12 task_name_tree . . . 14

5.1.5.13 vehicle_state . . . 14

5.2 State Engine API . . . 15

5.2.1 Detailed Description . . . 17

5.2.2 Macro Definition Documentation . . . 17

5.2.2.1 UV_TASK_AWAITING_DELETION . . . 17

5.2.2.2 UV_TASK_DEADLINE_FIRM . . . 17

5.2.2.3 UV_TASK_DEADLINE_HARD . . . 17

5.2.2.4 UV_TASK_DEADLINE_MASK . . . 17

5.2.2.5 UV_TASK_DEADLINE_NOT_ENFORCED . . . 18

5.2.2.6 UV_TASK_DEFER_DELETION . . . 18

5.2.2.7 UV_TASK_DELAYING . . . 18
5.2.2.8 UV_TASK_DORMANT_SVC . . . 18
5.2.2.9 UV_TASK_ERR_IN_CHILD . . . 18
5.2.2.10 UV_TASK_GENERIC_SVC . . . 18
5.2.2.11 UV_TASK_IS_CHILD . . . 19
5.2.2.12 UV_TASK_IS_ORPHAN . . . 19
5.2.2.13 UV_TASK_IS_PARENT . . . 19
5.2.2.14 UV_TASK_LOG_MEM_USAGE . . . 19
5.2.2.15 UV_TASK_LOG_START_STOP_TIME . . . 19
5.2.2.16 UV_TASK_MANAGER_MASK . . . 19
5.2.2.17 UV_TASK_MISSION_CRITICAL . . . 20
5.2.2.18 UV_TASK_PERIODIC_SVC . . . 20
5.2.2.19 UV_TASK_PRIO_INCREMENTATION . . . 20
5.2.2.20 UV_TASK_SCD_IGNORE . . . 20
5.2.2.21 UV_TASK_VEHICLE_APPLICATION . . . 20
5.2.2.22 uvTaskDelay . . . 20
5.2.2.23 uvTaskDelayUntil . . . 21
5.2.2.24 uvTaskIsDelaying . . . 21
5.2.2.25 uvTaskResetDelayBit . . . 21
5.2.2.26 uvTaskResetDeletionBit . . . 22
5.2.2.27 uvTaskSetDelayBit . . . 22
5.2.2.28 uvTaskSetDeletionBit . . . 22
5.2.3 Typedef Documentation . . . 22
5.2.3.1 task_management_info . . . 22
5.2.3.2 task_priority . . . 22
5.2.3.3 task_status_block . . . 23
5.2.3.4 uv_os_settings . . . 23
5.2.3.5 uv_scd_response . . . 23
5.2.3.6 uv_task_cmd . . . 23
5.2.3.7 uv_task_info . . . 23
5.2.3.8 uv_task_status . . . 23
5.2.3.9 uv_vehicle_state . . . 24
5.2.4 Enumeration Type Documentation . . . 24
5.2.4.1 task_priority . . . 24
5.2.4.2 uv_scd_response_e . . . 24
5.2.4.3 uv_task_cmd_e . . . 25
5.2.4.4 uv_task_state_t . . . 25
5.2.4.5 uv_vehicle_state_t . . . 25
5.2.5 Function Documentation . . . 26
5.2.5.1 changeVehicleState() . . . 26
5.2.5.2 uvCreateTask() . . . 27
5.2.5.3 uvDeInitStateEngine() . . . 27

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

5.5.2.7 deserializeSmallE16 . . . 44
5.5.2.8 deserializeSmallE32 . . . 44
5.5.2.9 endianSwap . . . 44
5.5.2.10 endianSwap16 . . . 44
5.5.2.11 endianSwap32 . . . 44
5.5.2.12 endianSwap8 . . . 45
5.5.2.13 false . . . 45
5.5.2.14 isPowerOfTwo . . . 45
5.5.2.15 safePtrRead . . . 45
5.5.2.16 safePtrWrite . . . 45
5.5.2.17 serializeBigE16 . . . 46
5.5.2.18 serializeBigE32 . . . 46
5.5.2.19 serializeSmallE16 . . . 46
5.5.2.20 serializeSmallE32 . . . 46
5.5.2.21 setBits . . . 46
5.5.2.22 true . . . 47
5.6 UVFR Vehicle Commands . . . 48
5.7 UVFR CANbus API . . . 49
5.7.1 Detailed Description . . . 49
5.7.2 Function Documentation . . . 49
5.7.2.1 insertCANMessageHandler() . . . 49
5.7.2.2 uvSendCanMSG() . . . 49
5.8 CMSIS . . . 50
5.8.1 Detailed Description . . . 50
5.9 Stm32f4xx_system . . . 51
5.9.1 Detailed Description . . . 51
5.10 STM32F4xx_System_Private_Includes . . . 52
5.10.1 Detailed Description . . . 52
5.10.2 Macro Definition Documentation . . . 52
5.10.2.1 HSE_VALUE . . . 52
5.10.2.2 HSI_VALUE . . . 52
5.11 STM32F4xx_System_Private_TypesDefinitions . . . 53
5.12 STM32F4xx_System_Private_Defines . . . 54
5.13 STM32F4xx_System_Private_Macros . . . 55
5.14 STM32F4xx_System_Private_Variables . . . 56
5.14.1 Detailed Description . . . 56
5.14.2 Variable Documentation . . . 56
5.14.2.1 AHBPrescTable . . . 56
5.14.2.2 APBPrescTable . . . 56
5.14.2.3 SystemCoreClock . . . 56
5.15 STM32F4xx_System_Private_FunctionPrototypes . . . 57
5.16 STM32F4xx_System_Private_Functions . . . 58

5.16.1 Detailed Description . . . 58
5.16.2 Function Documentation . . . 58
5.16.2.1 SystemCoreClockUpdate() . . . 58
5.16.2.2 SystemInit() . . . 58

**6 Data Structure Documentation** **59**

6.1 access_control_info Union Reference . . . 59
6.1.1 Detailed Description . . . 59
6.1.2 Field Documentation . . . 59
6.1.2.1 bin_semaphore . . . 59
6.1.2.2 mutex . . . 59
6.1.2.3 semaphore . . . 60
6.2 bms_settings_t Struct Reference . . . 60
6.2.1 Detailed Description . . . 60
6.2.2 Field Documentation . . . 60
6.2.2.1 mc_CAN_timeout . . . 60
6.3 CAN_Callback Struct Reference . . . 60
6.3.1 Detailed Description . . . 61
6.3.2 Field Documentation . . . 61
6.3.2.1 CAN_id . . . 61
6.3.2.2 function . . . 61
6.3.2.3 next . . . 61
6.4 daq_child_task Struct Reference . . . 61
6.4.1 Detailed Description . . . 62
6.4.2 Field Documentation . . . 62
6.4.2.1 meta_task_handle . . . 62
6.4.2.2 param_list . . . 62
6.4.2.3 period . . . 62
6.4.2.4 treenode . . . 62
6.5 daq_datapoint Struct Reference . . . 63
6.5.1 Detailed Description . . . 63
6.5.2 Field Documentation . . . 63
6.5.2.1 can_id . . . 63
6.5.2.2 period . . . 63
6.5.2.3 type . . . 63
6.6 daq_loop_args Struct Reference . . . 64
6.6.1 Detailed Description . . . 64
6.6.2 Field Documentation . . . 64
6.6.2.1 datapoints . . . 64
6.6.2.2 minimum_daq_period . . . 64
6.6.2.3 padding . . . 64
6.6.2.4 padding2 . . . 65

6.6.2.5 throttle_daq_to_preserve_performance . . . 65
6.7 daq_param_list_node Struct Reference . . . 65
6.7.1 Detailed Description . . . 65
6.7.2 Field Documentation . . . 65
6.7.2.1 next . . . 65
6.7.2.2 param_idx . . . 66
6.8 driving_loop_args Struct Reference . . . 66
6.8.1 Detailed Description . . . 66
6.8.2 Field Documentation . . . 66
6.8.2.1 absolute_max_acc_pwr . . . 67
6.8.2.2 absolute_max_accum_current . . . 67
6.8.2.3 absolute_max_motor_rpm . . . 67
6.8.2.4 absolute_max_motor_torque . . . 67
6.8.2.5 accum_regen_soc_threshold . . . 67
6.8.2.6 apps_bottom . . . 68
6.8.2.7 apps_implausibility_recovery_threshold . . . 68
6.8.2.8 apps_plausibility_check_threshold . . . 68
6.8.2.9 apps_top . . . 68
6.8.2.10 bps_implausibility_recovery_threshold . . . 68
6.8.2.11 bps_plausibility_check_threshold . . . 69
6.8.2.12 dmodes . . . 69
6.8.2.13 max_accum_current_5s . . . 69
6.8.2.14 max_apps_offset . . . 69
6.8.2.15 max_apps_value . . . 69
6.8.2.16 max_BPS_value . . . 70
6.8.2.17 min_apps_offset . . . 70
6.8.2.18 min_apps_value . . . 70
6.8.2.19 min_BPS_value . . . 70
6.8.2.20 num_driving_modes . . . 70
6.8.2.21 period . . . 71
6.8.2.22 regen_rpm_cutoff . . . 71
6.9 drivingLoopArgs Struct Reference . . . 71
6.9.1 Detailed Description . . . 71
6.10 drivingMode Struct Reference . . . 71
6.10.1 Detailed Description . . . 72
6.10.2 Field Documentation . . . 72
6.10.2.1 control_map_fn . . . 72
6.10.2.2 dm_name . . . 72
6.10.2.3 flags . . . 72
6.10.2.4 map_fn_params . . . 73
6.10.2.5 max_acc_pwr . . . 73
6.10.2.6 max_current . . . 73

6.10.2.7 max_motor_torque . . . 73
6.11 drivingModeParams Union Reference . . . 73
6.11.1 Detailed Description . . . 73
6.12 exp_torque_map_args Struct Reference . . . 74
6.12.1 Detailed Description . . . 74
6.12.2 Field Documentation . . . 74
6.12.2.1 gamma . . . 74
6.12.2.2 offset . . . 74
6.13 linear_torque_map_args Struct Reference . . . 74
6.13.1 Detailed Description . . . 75
6.13.2 Field Documentation . . . 75
6.13.2.1 offset . . . 75
6.13.2.2 slope . . . 75
6.14 motor_controllor_settings Struct Reference . . . 75
6.14.1 Detailed Description . . . 76
6.14.2 Field Documentation . . . 76
6.14.2.1 can_id_rx . . . 76
6.14.2.2 can_id_tx . . . 76
6.14.2.3 integral_memory_max . . . 76
6.14.2.4 integral_time_constant . . . 76
6.14.2.5 mc_CAN_timeout . . . 76
6.14.2.6 proportional_gain . . . 77
6.15 p_status Struct Reference . . . 77
6.15.1 Detailed Description . . . 77
6.15.2 Field Documentation . . . 77
6.15.2.1 activation_time . . . 77
6.15.2.2 peripheral_status . . . 77
6.16 rbnode Struct Reference . . . 78
6.16.1 Detailed Description . . . 78
6.16.2 Field Documentation . . . 78
6.16.2.1 color . . . 78
6.16.2.2 data . . . 78
6.16.2.3 left . . . 79
6.16.2.4 parent . . . 79
6.16.2.5 right . . . 79
6.17 rbtree Struct Reference . . . 79
6.17.1 Detailed Description . . . 80
6.17.2 Field Documentation . . . 80
6.17.2.1 compare . . . 80
6.17.2.2 count . . . 80
6.17.2.3 destroy . . . 81
6.17.2.4 min . . . 81

6.17.2.5 nil . . . 81
6.17.2.6 print . . . 81
6.17.2.7 root . . . 82
6.18 s_curve_torque_map_args Struct Reference . . . 82
6.18.1 Detailed Description . . . 82
6.18.2 Field Documentation . . . 82
6.18.2.1 a . . . 82
6.18.2.2 b . . . 83
6.18.2.3 c . . . 83
6.19 state_change_daemon_args Struct Reference . . . 83
6.19.1 Detailed Description . . . 83
6.19.2 Field Documentation . . . 83
6.19.2.1 meta_task_handle . . . 83
6.20 task_management_info Struct Reference . . . 84
6.20.1 Detailed Description . . . 84
6.20.2 Field Documentation . . . 84
6.20.2.1 parent_msg_queue . . . 84
6.20.2.2 task_handle . . . 84
6.21 task_status_block Struct Reference . . . 85
6.21.1 Detailed Description . . . 85
6.21.2 Field Documentation . . . 85
6.21.2.1 task_high_water_mark . . . 85
6.21.2.2 task_report_time . . . 85
6.22 uv_binary_semaphore_info Struct Reference . . . 85
6.22.1 Detailed Description . . . 86
6.22.2 Field Documentation . . . 86
6.22.2.1 handle . . . 86
6.23 uv_CAN_msg Struct Reference . . . 86
6.23.1 Detailed Description . . . 86
6.23.2 Field Documentation . . . 86
6.23.2.1 data . . . 87
6.23.2.2 dlc . . . 87
6.23.2.3 flags . . . 87
6.23.2.4 msg_id . . . 87
6.24 uv_init_struct Struct Reference . . . 88
6.24.1 Detailed Description . . . 88
6.24.2 Field Documentation . . . 88
6.24.2.1 use_default_settings . . . 88
6.25 uv_init_task_args Struct Reference . . . 88
6.25.1 Detailed Description . . . 89
6.25.2 Field Documentation . . . 89
6.25.2.1 init_info_queue . . . 89

6.25.2.2 meta_task_handle . . . 89
6.25.2.3 specific_args . . . 89
6.26 uv_init_task_response Struct Reference . . . 89
6.26.1 Detailed Description . . . 90
6.26.2 Field Documentation . . . 90
6.26.2.1 device . . . 90
6.26.2.2 errmsg . . . 90
6.26.2.3 nchar . . . 90
6.26.2.4 status . . . 91
6.27 uv_internal_params Struct Reference . . . 91
6.27.1 Detailed Description . . . 91
6.27.2 Field Documentation . . . 91
6.27.2.1 e_code . . . 91
6.27.2.2 init_params . . . 92
6.27.2.3 peripheral_status . . . 92
6.27.2.4 vehicle_settings . . . 92
6.28 uv_mutex_info Struct Reference . . . 92
6.28.1 Detailed Description . . . 92
6.28.2 Field Documentation . . . 92
6.28.2.1 handle . . . 93
6.29 uv_os_settings Struct Reference . . . 93
6.29.1 Detailed Description . . . 93
6.29.2 Field Documentation . . . 93
6.29.2.1 max_svc_task_period . . . 93
6.29.2.2 max_task_period . . . 94
6.29.2.3 min_task_period . . . 94
6.29.2.4 svc_task_manager_period . . . 94
6.29.2.5 task_manager_period . . . 94
6.30 uv_scd_response Struct Reference . . . 94
6.30.1 Detailed Description . . . 94
6.30.2 Field Documentation . . . 95
6.30.2.1 meta_id . . . 95
6.30.2.2 response_val . . . 95
6.31 uv_semaphore_info Struct Reference . . . 95
6.31.1 Detailed Description . . . 95
6.31.2 Field Documentation . . . 95
6.31.2.1 handle . . . 96
6.32 uv_task_info Struct Reference . . . 96
6.32.1 Detailed Description . . . 96
6.32.2 Field Documentation . . . 97
6.32.2.1 active_states . . . 97
6.32.2.2 cmd_data . . . 97

6.32.2.3 deletion_delay . . . 97
6.32.2.4 deletion_states . . . 97
6.32.2.5 parent . . . 98
6.32.2.6 stack_size . . . 98
6.32.2.7 suspension_states . . . 98
6.32.2.8 task_args . . . 98
6.32.2.9 task_flags . . . 99
6.32.2.10 task_function . . . 99
6.32.2.11 task_handle . . . 100
6.32.2.12 task_id . . . 100
6.32.2.13 task_name . . . 100
6.32.2.14 task_period . . . 100
6.32.2.15 task_priority . . . 101
6.32.2.16 task_rx_mailbox . . . 101
6.32.2.17 task_state . . . 101
6.32.2.18 tmi . . . 101
6.33 uv_task_msg_t Struct Reference . . . 101
6.33.1 Detailed Description . . . 102
6.33.2 Field Documentation . . . 102
6.33.2.1 intended_recipient . . . 102
6.33.2.2 message_size . . . 102
6.33.2.3 message_type . . . 102
6.33.2.4 msg_contents . . . 103
6.33.2.5 sender . . . 103
6.33.2.6 time_sent . . . 103
6.34 uv_vehicle_settings Struct Reference . . . 103
6.34.1 Detailed Description . . . 103
6.34.2 Field Documentation . . . 104
6.34.2.1 bms_settings . . . 104
6.34.2.2 daq_settings . . . 104
6.34.2.3 driving_loop_settings . . . 104
6.34.2.4 imd_settings . . . 104
6.34.2.5 is_default . . . 104
6.34.2.6 mc_settings . . . 105
6.34.2.7 os_settings . . . 105
6.34.2.8 pdu_settings . . . 105
6.35 veh_gen_info Struct Reference . . . 105
6.35.1 Detailed Description . . . 105

**7 File Documentation** **107**

7.1 Core/Inc/adc.h File Reference . . . 107
7.1.1 Detailed Description . . . 108

7.1.2 Macro Definition Documentation . . . 108
7.1.2.1 ADC1_BUF_LEN . . . 108
7.1.2.2 ADC1_CHNL_CNT . . . 108
7.1.2.3 ADC1_MAX_VOLT . . . 108
7.1.2.4 ADC1_MIN_VOLT . . . 108
7.1.2.5 ADC1_SAMPLES . . . 109
7.1.2.6 ADC2_BUF_LEN . . . 109
7.1.2.7 ADC2_CHNL_CNT . . . 109
7.1.2.8 ADC2_MAX_VOLT . . . 109
7.1.2.9 ADC2_MIN_VOLT . . . 109
7.1.2.10 ADC2_SAMPLES . . . 109
7.1.3 Function Documentation . . . 110
7.1.3.1 MX_ADC1_Init() . . . 110
7.1.3.2 MX_ADC2_Init() . . . 110
7.1.4 Variable Documentation . . . 110
7.1.4.1 hadc1 . . . 111
7.1.4.2 hadc2 . . . 111
7.2 Core/Inc/bms.h File Reference . . . 111
7.2.1 Macro Definition Documentation . . . 111
7.2.1.1 DEFAULT_BMS_CAN_TIMEOUT . . . 112
7.2.2 Typedef Documentation . . . 112
7.2.2.1 bms_settings_t . . . 112
7.2.3 Function Documentation . . . 112
7.2.3.1 BMS_Init() . . . 112
7.3 Core/Inc/can.h File Reference . . . 112
7.3.1 Detailed Description . . . 113
7.3.2 Macro Definition Documentation . . . 113
7.3.2.1 CAN_RX_DAEMON_NAME . . . 113
7.3.2.2 CAN_TX_DAEMON_NAME . . . 114
7.3.3 Typedef Documentation . . . 114
7.3.3.1 uv_CAN_msg . . . 114
7.3.3.2 uv_status . . . 114
7.3.4 Function Documentation . . . 114
7.3.4.1 CANbusRxSvcDaemon() . . . 114
7.3.4.2 CANbusTxSvcDaemon() . . . 115
7.3.4.3 HAL_CAN_RxFifo0MsgPendingCallback() . . . 115
7.3.4.4 HAL_CAN_RxFifo1MsgPendingCallback() . . . 115
7.3.4.5 MX_CAN2_Init() . . . 115
7.3.4.6 nuke_hash_table() . . . 116
7.3.5 Variable Documentation . . . 116
7.3.5.1 hcan2 . . . 116
7.4 Core/Inc/constants.h File Reference . . . 116

7.4.1 Enumeration Type Documentation . . . 116
7.4.1.1 CAN_IDs . . . 116
7.4.2 Variable Documentation . . . 117
7.4.2.1 RxData . . . 117
7.4.2.2 RxHeader . . . 117
7.4.2.3 TxData . . . 117
7.4.2.4 TxHeader . . . 118
7.4.2.5 TxMailbox . . . 118
7.5 Core/Inc/daq.h File Reference . . . 118
7.5.1 Macro Definition Documentation . . . 119
7.5.1.1 _NUM_LOGGABLE_PARAMS . . . 119
7.5.2 Typedef Documentation . . . 119
7.5.2.1 daq_child_task . . . 119
7.5.2.2 daq_datapoint . . . 119
7.5.2.3 daq_loop_args . . . 120
7.5.2.4 daq_param_list_node . . . 120
7.5.3 Enumeration Type Documentation . . . 120
7.5.3.1 loggable_params . . . 120
7.5.4 Function Documentation . . . 121
7.5.4.1 daqMasterTask() . . . 121
7.5.4.2 initDaqTask() . . . 121
7.5.5 Variable Documentation . . . 121
7.5.5.1 param_LUT . . . 121
7.6 Core/Inc/dash.h File Reference . . . 122
7.6.1 Enumeration Type Documentation . . . 122
7.6.1.1 dash_can_ids . . . 122
7.6.2 Function Documentation . . . 122
7.6.2.1 Update_Batt_Temp() . . . 122
7.6.2.2 Update_RPM() . . . 123
7.6.2.3 Update_State_Of_Charge() . . . 123
7.7 Core/Inc/dma.h File Reference . . . 123
7.7.1 Detailed Description . . . 123
7.7.2 Function Documentation . . . 124
7.7.2.1 MX_DMA_Init() . . . 124
7.8 Core/Inc/driving_loop.h File Reference . . . 124
7.8.1 Typedef Documentation . . . 125
7.8.1.1 driving_loop_args . . . 125
7.8.1.2 drivingMode . . . 125
7.8.1.3 drivingModeParams . . . 125
7.8.1.4 exp_torque_map_args . . . 125
7.8.1.5 linear_torque_map_args . . . 126
7.8.1.6 MC_POWER . . . 126

7.8.1.7 MC_RPM . . . 126
7.8.1.8 MC_Torque . . . 126
7.8.1.9 s_curve_torque_map_args . . . 126
7.8.2 Enumeration Type Documentation . . . 126
7.8.2.1 DL_internal_state . . . 126
7.8.2.2 map_mode . . . 127
7.8.3 Function Documentation . . . 127
7.8.3.1 initDrivingLoop() . . . 127
7.8.3.2 StartDrivingLoop() . . . 128
7.9 Core/Inc/errorLUT.h File Reference . . . 129
7.9.1 Macro Definition Documentation . . . 129
7.9.1.1 _NUM_ERRORS_ . . . 129
7.10 Core/Inc/FreeRTOSConfig.h File Reference . . . 129
7.10.1 Macro Definition Documentation . . . 130
7.10.1.1 configASSERT . . . 130
7.10.1.2 configCHECK_FOR_STACK_OVERFLOW [1/2] . . . 131
7.10.1.3 configCHECK_FOR_STACK_OVERFLOW [2/2] . . . 131
7.10.1.4 configCPU_CLOCK_HZ . . . 131
7.10.1.5 configENABLE_BACKWARD_COMPATIBILITY . . . 131
7.10.1.6 configENABLE_FPU . . . 131
7.10.1.7 configENABLE_MPU . . . 131
7.10.1.8 configKERNEL_INTERRUPT_PRIORITY . . . 132
7.10.1.9 configLIBRARY_LOWEST_INTERRUPT_PRIORITY . . . 132
7.10.1.10 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY . . . 132
7.10.1.11 configMAX_CO_ROUTINE_PRIORITIES . . . 132
7.10.1.12 configMAX_PRIORITIES . . . 132
7.10.1.13 configMAX_SYSCALL_INTERRUPT_PRIORITY . . . 132
7.10.1.14 configMAX_TASK_NAME_LEN . . . 133
7.10.1.15 configMESSAGE_BUFFER_LENGTH_TYPE . . . 133
7.10.1.16 configMINIMAL_STACK_SIZE . . . 133
7.10.1.17 configPRIO_BITS . . . 133
7.10.1.18 configQUEUE_REGISTRY_SIZE . . . 133
7.10.1.19 configRECORD_STACK_HIGH_ADDRESS . . . 133
7.10.1.20 configSUPPORT_DYNAMIC_ALLOCATION . . . 134
7.10.1.21 configSUPPORT_STATIC_ALLOCATION . . . 134
7.10.1.22 configTICK_RATE_HZ . . . 134
7.10.1.23 configTIMER_QUEUE_LENGTH . . . 134
7.10.1.24 configTIMER_TASK_PRIORITY . . . 134
7.10.1.25 configTIMER_TASK_STACK_DEPTH . . . 134
7.10.1.26 configTOTAL_HEAP_SIZE . . . 135
7.10.1.27 configUSE_16_BIT_TICKS . . . 135
7.10.1.28 configUSE_APPLICATION_TASK_TAG . . . 135

7.10.1.29 configUSE_CO_ROUTINES . . . 135
7.10.1.30 configUSE_COUNTING_SEMAPHORES . . . 135
7.10.1.31 configUSE_IDLE_HOOK . . . 135
7.10.1.32 configUSE_MALLOC_FAILED_HOOK [1/2] . . . 136
7.10.1.33 configUSE_MALLOC_FAILED_HOOK [2/2] . . . 136
7.10.1.34 configUSE_MUTEXES . . . 136
7.10.1.35 configUSE_PORT_OPTIMISED_TASK_SELECTION . . . 136
7.10.1.36 configUSE_PREEMPTION . . . 136
7.10.1.37 configUSE_TICK_HOOK . . . 136
7.10.1.38 configUSE_TIMERS . . . 137
7.10.1.39 INCLUDE_eTaskGetState . . . 137
7.10.1.40 INCLUDE_pcTaskGetTaskName . . . 137
7.10.1.41 INCLUDE_uxTaskGetStackHighWaterMark . . . 137
7.10.1.42 INCLUDE_uxTaskGetStackHighWaterMark2 . . . 137
7.10.1.43 INCLUDE_uxTaskPriorityGet . . . 137
7.10.1.44 INCLUDE_vTaskCleanUpResources . . . 138
7.10.1.45 INCLUDE_vTaskDelay . . . 138
7.10.1.46 INCLUDE_vTaskDelayUntil . . . 138
7.10.1.47 INCLUDE_vTaskDelete . . . 138
7.10.1.48 INCLUDE_vTaskPrioritySet . . . 138
7.10.1.49 INCLUDE_vTaskSuspend . . . 138
7.10.1.50 INCLUDE_xEventGroupSetBitFromISR . . . 139
7.10.1.51 INCLUDE_xQueueGetMutexHolder . . . 139
7.10.1.52 INCLUDE_xSemaphoreGetMutexHolder . . . 139
7.10.1.53 INCLUDE_xTaskAbortDelay . . . 139
7.10.1.54 INCLUDE_xTaskDelayUntil . . . 139
7.10.1.55 INCLUDE_xTaskGetCurrentTaskHandle . . . 139
7.10.1.56 INCLUDE_xTaskGetHandle . . . 140
7.10.1.57 INCLUDE_xTaskGetSchedulerState . . . 140
7.10.1.58 INCLUDE_xTimerPendFunctionCall . . . 140
7.10.1.59 vPortSVCHandler . . . 140
7.10.1.60 xPortPendSVHandler . . . 140
7.10.1.61 xPortSysTickHandler . . . 140
7.11 Core/Inc/gpio.h File Reference . . . 141
7.11.1 Detailed Description . . . 141
7.11.2 Function Documentation . . . 141
7.11.2.1 MX_GPIO_Init() . . . 141
7.12 Core/Inc/imd.h File Reference . . . 141
7.12.1 Enumeration Type Documentation . . . 142
7.12.1.1 imd_error_flags . . . 142
7.12.1.2 imd_high_resolution_measurements . . . 143
7.12.1.3 imd_manufacturer_requests . . . 143

7.12.1.4 imd_status_bits . . . 144
7.12.1.5 imd_status_requests . . . 144
7.12.2 Function Documentation . . . 145
7.12.2.1 IMD_Check_Battery_Voltage() . . . 145
7.12.2.2 IMD_Check_Error_Flags() . . . 145
7.12.2.3 IMD_Check_Isolation_Capacitances() . . . 145
7.12.2.4 IMD_Check_Isolation_Resistances() . . . 145
7.12.2.5 IMD_Check_Isolation_State() . . . 146
7.12.2.6 IMD_Check_Max_Battery_Working_Voltage() . . . 146
7.12.2.7 IMD_Check_Part_Name() . . . 146
7.12.2.8 IMD_Check_Safety_Touch_Current() . . . 146
7.12.2.9 IMD_Check_Safety_Touch_Energy() . . . 147
7.12.2.10 IMD_Check_Serial_Number() . . . 147
7.12.2.11 IMD_Check_Status_Bits() . . . 147
7.12.2.12 IMD_Check_Temperature() . . . 147
7.12.2.13 IMD_Check_Uptime() . . . 148
7.12.2.14 IMD_Check_Version() . . . 148
7.12.2.15 IMD_Check_Voltages_Vp_and_Vn() . . . 148
7.12.2.16 IMD_Parse_Message() . . . 148
7.12.2.17 IMD_Request_Status() . . . 149
7.12.2.18 IMD_Startup() . . . 149
7.12.2.19 initIMD() . . . 149
7.13 Core/Inc/main.h File Reference . . . 149
7.13.1 Detailed Description . . . 150
7.13.2 Macro Definition Documentation . . . 150
7.13.2.1 Blue_LED_GPIO_Port . . . 150
7.13.2.2 Blue_LED_Pin . . . 150
7.13.2.3 Orange_LED_GPIO_Port . . . 151
7.13.2.4 Orange_LED_Pin . . . 151
7.13.2.5 Red_LED_GPIO_Port . . . 151
7.13.2.6 Red_LED_Pin . . . 151
7.13.2.7 Start_Button_Input_EXTI_IRQn . . . 151
7.13.2.8 Start_Button_Input_GPIO_Port . . . 151
7.13.2.9 Start_Button_Input_Pin . . . 152
7.13.3 Function Documentation . . . 152
7.13.3.1 Error_Handler() . . . 152
7.14 Core/Inc/motor_controller.h File Reference . . . 152
7.14.1 Macro Definition Documentation . . . 154
7.14.1.1 DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT . . . 154
7.14.1.2 FIRMWARE_VERSION_REGISTER . . . 154
7.14.1.3 SERIAL_NUMBER_REGISTER . . . 154
7.14.2 Typedef Documentation . . . 154

7.14.2.1 motor_controller_settings . . . 154
7.14.3 Enumeration Type Documentation . . . 154
7.14.3.1 motor_controller_current_parameters . . . 154
7.14.3.2 motor_controller_io . . . 155
7.14.3.3 motor_controller_limp_mode . . . 155
7.14.3.4 motor_controller_measurements . . . 155
7.14.3.5 motor_controller_motor_constants . . . 156
7.14.3.6 motor_controller_PI_values . . . 156
7.14.3.7 motor_controller_repeating_time . . . 157
7.14.3.8 motor_controller_speed_parameters . . . 157
7.14.3.9 motor_controller_startup . . . 157
7.14.3.10 motor_controller_status_information_errors_warnings . . . 157
7.14.3.11 motor_controller_temperatures . . . 158
7.14.4 Function Documentation . . . 159
7.14.4.1 MC_Startup() . . . 159
7.15 Core/Inc/odometer.h File Reference . . . 159
7.15.1 Function Documentation . . . 160
7.15.1.1 initOdometer() . . . 160
7.15.1.2 odometerTask() . . . 160
7.16 Core/Inc/oled.h File Reference . . . 160
7.16.1 Function Documentation . . . 161
7.16.1.1 amogus() . . . 161
7.16.1.2 oled_config() . . . 161
7.16.1.3 oled_Write() . . . 161
7.16.1.4 oled_Write_Cmd() . . . 161
7.16.1.5 oled_Write_Data() . . . 161
7.16.1.6 refresh_OLED() . . . 162
7.16.1.7 wait() . . . 162
7.17 Core/Inc/pdu.h File Reference . . . 162
7.17.1 Enumeration Type Documentation . . . 162
7.17.1.1 pdu_messages_20A . . . 162
7.17.1.2 pdu_messages_5A . . . 163
7.17.2 Function Documentation . . . 163
7.17.2.1 initPDU() . . . 163
7.17.2.2 PDU_disable_brake_light() . . . 164
7.17.2.3 PDU_disable_coolant_pump() . . . 164
7.17.2.4 PDU_disable_cooling_fans() . . . 164
7.17.2.5 PDU_disable_motor_controller() . . . 164
7.17.2.6 PDU_disable_shutdown_circuit() . . . 164
7.17.2.7 PDU_enable_brake_light() . . . 165
7.17.2.8 PDU_enable_coolant_pump() . . . 165
7.17.2.9 PDU_enable_cooling_fans() . . . 165

7.17.2.10 PDU_enable_motor_controller() . . . 165
7.17.2.11 PDU_enable_shutdown_circuit() . . . 165
7.17.2.12 PDU_speaker_chirp() . . . 166
7.18 Core/Inc/rb_tree.h File Reference . . . 166
7.18.1 Macro Definition Documentation . . . 167
7.18.1.1 BLACK . . . 167
7.18.1.2 RB_APPLY . . . 167
7.18.1.3 RB_DUP . . . 167
7.18.1.4 RB_FIRST . . . 168
7.18.1.5 RB_ISEMPTY . . . 168
7.18.1.6 RB_MIN . . . 168
7.18.1.7 RB_MINIMAL . . . 168
7.18.1.8 RB_NIL . . . 168
7.18.1.9 RB_ROOT . . . 168
7.18.1.10 RED . . . 169
7.18.2 Typedef Documentation . . . 169
7.18.2.1 rbnode . . . 169
7.18.3 Enumeration Type Documentation . . . 169
7.18.3.1 rbtraversal . . . 169
7.18.4 Function Documentation . . . 169
7.18.4.1 rbApplyNode() . . . 169
7.18.4.2 rbCheckBlackHeight() . . . 170
7.18.4.3 rbCheckOrder() . . . 170
7.18.4.4 rbCreate() . . . 170
7.18.4.5 rbDelete() . . . 170
7.18.4.6 rbDestroy() . . . 171
7.18.4.7 rbFind() . . . 171
7.18.4.8 rbInsert() . . . 171
7.18.4.9 rbPrint() . . . 171
7.18.4.10 rbSuccessor() . . . 172
7.19 Core/Inc/spi.h File Reference . . . 172
7.19.1 Detailed Description . . . 172
7.19.2 Function Documentation . . . 172
7.19.2.1 MX_SPI1_Init() . . . 173
7.19.3 Variable Documentation . . . 173
7.19.3.1 hspi1 . . . 173
7.20 Core/Inc/stm32f4xx_hal_conf.h File Reference . . . 173
7.20.1 Detailed Description . . . 176
7.20.2 Macro Definition Documentation . . . 176
7.20.2.1 assert_param . . . 176
7.20.2.2 DATA_CACHE_ENABLE . . . 176
7.20.2.3 DP83848_PHY_ADDRESS . . . 177

7.20.2.4 ETH_RX_BUF_SIZE . . . 177
7.20.2.5 ETH_RXBUFNB . . . 177
7.20.2.6 ETH_TX_BUF_SIZE . . . 177
7.20.2.7 ETH_TXBUFNB . . . 177
7.20.2.8 EXTERNAL_CLOCK_VALUE . . . 177
7.20.2.9 HAL_ADC_MODULE_ENABLED . . . 178
7.20.2.10 HAL_CAN_MODULE_ENABLED . . . 178
7.20.2.11 HAL_CORTEX_MODULE_ENABLED . . . 178
7.20.2.12 HAL_DMA_MODULE_ENABLED . . . 178
7.20.2.13 HAL_EXTI_MODULE_ENABLED . . . 178
7.20.2.14 HAL_FLASH_MODULE_ENABLED . . . 178
7.20.2.15 HAL_GPIO_MODULE_ENABLED . . . 179
7.20.2.16 HAL_MODULE_ENABLED . . . 179
7.20.2.17 HAL_PWR_MODULE_ENABLED . . . 179
7.20.2.18 HAL_RCC_MODULE_ENABLED . . . 179
7.20.2.19 HAL_SPI_MODULE_ENABLED . . . 179
7.20.2.20 HAL_TIM_MODULE_ENABLED . . . 179
7.20.2.21 HSE_STARTUP_TIMEOUT . . . 180
7.20.2.22 HSE_VALUE . . . 180
7.20.2.23 HSI_VALUE . . . 180
7.20.2.24 INSTRUCTION_CACHE_ENABLE . . . 180
7.20.2.25 LSE_STARTUP_TIMEOUT . . . 180
7.20.2.26 LSE_VALUE . . . 181
7.20.2.27 LSI_VALUE . . . 181
7.20.2.28 MAC_ADDR0 . . . 181
7.20.2.29 MAC_ADDR1 . . . 181
7.20.2.30 MAC_ADDR2 . . . 181
7.20.2.31 MAC_ADDR3 . . . 182
7.20.2.32 MAC_ADDR4 . . . 182
7.20.2.33 MAC_ADDR5 . . . 182
7.20.2.34 PHY_AUTONEGO_COMPLETE . . . 182
7.20.2.35 PHY_AUTONEGOTIATION . . . 182
7.20.2.36 PHY_BCR . . . 183
7.20.2.37 PHY_BSR . . . 183
7.20.2.38 PHY_CONFIG_DELAY . . . 183
7.20.2.39 PHY_DUPLEX_STATUS . . . 183
7.20.2.40 PHY_FULLDUPLEX_100M . . . 183
7.20.2.41 PHY_FULLDUPLEX_10M . . . 184
7.20.2.42 PHY_HALFDUPLEX_100M . . . 184
7.20.2.43 PHY_HALFDUPLEX_10M . . . 184
7.20.2.44 PHY_ISOLATE . . . 184
7.20.2.45 PHY_JABBER_DETECTION . . . 184

7.20.2.46 PHY_LINKED_STATUS . . . 185
7.20.2.47 PHY_LOOPBACK . . . 185
7.20.2.48 PHY_POWERDOWN . . . 185
7.20.2.49 PHY_READ_TO . . . 185
7.20.2.50 PHY_RESET . . . 185
7.20.2.51 PHY_RESET_DELAY . . . 186
7.20.2.52 PHY_RESTART_AUTONEGOTIATION . . . 186
7.20.2.53 PHY_SPEED_STATUS . . . 186
7.20.2.54 PHY_SR . . . 186
7.20.2.55 PHY_WRITE_TO . . . 186
7.20.2.56 PREFETCH_ENABLE . . . 187
7.20.2.57 TICK_INT_PRIORITY . . . 187
7.20.2.58 USE_HAL_ADC_REGISTER_CALLBACKS . . . 187
7.20.2.59 USE_HAL_CAN_REGISTER_CALLBACKS . . . 187
7.20.2.60 USE_HAL_CEC_REGISTER_CALLBACKS . . . 187
7.20.2.61 USE_HAL_CRYP_REGISTER_CALLBACKS . . . 187
7.20.2.62 USE_HAL_DAC_REGISTER_CALLBACKS . . . 188
7.20.2.63 USE_HAL_DCMI_REGISTER_CALLBACKS . . . 188
7.20.2.64 USE_HAL_DFSDM_REGISTER_CALLBACKS . . . 188
7.20.2.65 USE_HAL_DMA2D_REGISTER_CALLBACKS . . . 188
7.20.2.66 USE_HAL_DSI_REGISTER_CALLBACKS . . . 188
7.20.2.67 USE_HAL_ETH_REGISTER_CALLBACKS . . . 188
7.20.2.68 USE_HAL_FMPI2C_REGISTER_CALLBACKS . . . 189
7.20.2.69 USE_HAL_FMPSMBUS_REGISTER_CALLBACKS . . . 189
7.20.2.70 USE_HAL_HASH_REGISTER_CALLBACKS . . . 189
7.20.2.71 USE_HAL_HCD_REGISTER_CALLBACKS . . . 189
7.20.2.72 USE_HAL_I2C_REGISTER_CALLBACKS . . . 189
7.20.2.73 USE_HAL_I2S_REGISTER_CALLBACKS . . . 189
7.20.2.74 USE_HAL_IRDA_REGISTER_CALLBACKS . . . 190
7.20.2.75 USE_HAL_LPTIM_REGISTER_CALLBACKS . . . 190
7.20.2.76 USE_HAL_LTDC_REGISTER_CALLBACKS . . . 190
7.20.2.77 USE_HAL_MMC_REGISTER_CALLBACKS . . . 190
7.20.2.78 USE_HAL_NAND_REGISTER_CALLBACKS . . . 190
7.20.2.79 USE_HAL_NOR_REGISTER_CALLBACKS . . . 190
7.20.2.80 USE_HAL_PCCARD_REGISTER_CALLBACKS . . . 191
7.20.2.81 USE_HAL_PCD_REGISTER_CALLBACKS . . . 191
7.20.2.82 USE_HAL_QSPI_REGISTER_CALLBACKS . . . 191
7.20.2.83 USE_HAL_RNG_REGISTER_CALLBACKS . . . 191
7.20.2.84 USE_HAL_RTC_REGISTER_CALLBACKS . . . 191
7.20.2.85 USE_HAL_SAI_REGISTER_CALLBACKS . . . 191
7.20.2.86 USE_HAL_SD_REGISTER_CALLBACKS . . . 192
7.20.2.87 USE_HAL_SDRAM_REGISTER_CALLBACKS . . . 192

7.20.2.88 USE_HAL_SMARTCARD_REGISTER_CALLBACKS . . . 192
7.20.2.89 USE_HAL_SMBUS_REGISTER_CALLBACKS . . . 192
7.20.2.90 USE_HAL_SPDIFRX_REGISTER_CALLBACKS . . . 192
7.20.2.91 USE_HAL_SPI_REGISTER_CALLBACKS . . . 192
7.20.2.92 USE_HAL_SRAM_REGISTER_CALLBACKS . . . 193
7.20.2.93 USE_HAL_TIM_REGISTER_CALLBACKS . . . 193
7.20.2.94 USE_HAL_UART_REGISTER_CALLBACKS . . . 193
7.20.2.95 USE_HAL_USART_REGISTER_CALLBACKS . . . 193
7.20.2.96 USE_HAL_WWDG_REGISTER_CALLBACKS . . . 193
7.20.2.97 USE_RTOS . . . 193
7.20.2.98 USE_SPI_CRC . . . 194
7.20.2.99 VDD_VALUE . . . 194
7.21 Core/Inc/stm32f4xx_it.h File Reference . . . 194
7.21.1 Detailed Description . . . 195
7.21.2 Function Documentation . . . 195
7.21.2.1 BusFault_Handler() . . . 195
7.21.2.2 CAN2_RX0_IRQHandler() . . . 195
7.21.2.3 CAN2_RX1_IRQHandler() . . . 195
7.21.2.4 CAN2_TX_IRQHandler() . . . 196
7.21.2.5 DebugMon_Handler() . . . 196
7.21.2.6 DMA2_Stream0_IRQHandler() . . . 196
7.21.2.7 EXTI0_IRQHandler() . . . 196
7.21.2.8 HardFault_Handler() . . . 197
7.21.2.9 MemManage_Handler() . . . 197
7.21.2.10 NMI_Handler() . . . 197
7.21.2.11 TIM1_UP_TIM10_IRQHandler() . . . 197
7.21.2.12 UsageFault_Handler() . . . 197
7.22 Core/Inc/temp_monitoring.h File Reference . . . 198
7.22.1 Function Documentation . . . 198
7.22.1.1 initTempMonitor() . . . 198
7.22.1.2 tempMonitorTask() . . . 198
7.23 Core/Inc/tim.h File Reference . . . 199
7.23.1 Detailed Description . . . 199
7.23.2 Function Documentation . . . 199
7.23.2.1 MX_TIM3_Init() . . . 199
7.23.3 Variable Documentation . . . 199
7.23.3.1 htim3 . . . 200
7.24 Core/Inc/uvfr_global_config.h File Reference . . . 200
7.24.1 Macro Definition Documentation . . . 200
7.24.1.1 ECUMASTER_PMU . . . 200
7.24.1.2 STM32_F407 . . . 200
7.24.1.3 STM32_H7xx . . . 200

7.24.1.4 USE_OS_MEM_MGMT . . . 201
7.24.1.5 UV19_PDU . . . 201
7.24.1.6 UV_MALLOC_LIMIT . . . 201
7.25 Core/Inc/uvfr_settings.h File Reference . . . 201
7.25.1 Macro Definition Documentation . . . 202
7.25.1.1 ENABLE_FLASH_SETTINGS . . . 202
7.25.2 Typedef Documentation . . . 202
7.25.2.1 uv_vehicle_settings . . . 202
7.25.2.2 veh_gen_info . . . 202
7.25.3 Function Documentation . . . 202
7.25.3.1 nukeSettings() . . . 202
7.25.3.2 uvSettingsInit() . . . 203
7.25.4 Variable Documentation . . . 203
7.25.4.1 current_vehicle_settings . . . 203
7.26 Core/Inc/uvfr_state_engine.h File Reference . . . 203
7.26.1 Macro Definition Documentation . . . 206
7.26.1.1 _LONGEST_SC_TIME . . . 206
7.26.1.2 _SC_DAEMON_PERIOD . . . 206
7.26.1.3 _UV_DEFAULT_TASK_INSTANCES . . . 206
7.26.1.4 _UV_DEFAULT_TASK_PERIOD . . . 207
7.26.1.5 _UV_DEFAULT_TASK_STACK_SIZE . . . 207
7.26.1.6 _UV_MIN_TASK_PERIOD . . . 207
7.26.1.7 SVC_TASK_MAX_CHECKIN_PERIOD . . . 207
7.26.2 Typedef Documentation . . . 207
7.26.2.1 uv_status . . . 207
7.26.2.2 uv_task_id . . . 207
7.26.2.3 uv_timespan_ms . . . 208
7.26.3 Function Documentation . . . 208
7.26.3.1 getSVCTaskID() . . . 208
7.26.3.2 updateRunningTasks() . . . 208
7.26.3.3 uvGetTaskById() . . . 208
7.26.3.4 uvRegisterTask() . . . 208
7.27 Core/Inc/uvfr_utils.h File Reference . . . 209
7.27.1 Detailed Description . . . 212
7.27.2 Macro Definition Documentation . . . 212
7.27.2.1 INIT_CHECK_PERIOD . . . 212
7.27.2.2 MAX_INIT_TIME . . . 212
7.27.2.3 USE_OLED_DEBUG . . . 212
7.27.2.4 UV_CAN1 . . . 212
7.27.2.5 UV_CAN2 . . . 212
7.27.2.6 UV_CAN_CHANNEL_MASK . . . 213
7.27.2.7 UV_CAN_DYNAMIC_MEM . . . 213

7.27.2.8 UV_CAN_EXTENDED_ID . . . 213
7.27.3 Typedef Documentation . . . 213
7.27.3.1 access_control_info . . . 213
7.27.3.2 access_control_type . . . 213
7.27.3.3 bool . . . 213
7.27.3.4 p_status . . . 214
7.27.3.5 uv_CAN_msg . . . 214
7.27.3.6 uv_ext_device_id . . . 214
7.27.3.7 uv_init_struct . . . 214
7.27.3.8 uv_init_task_args . . . 214
7.27.3.9 uv_init_task_response . . . 214
7.27.3.10 uv_internal_params . . . 215
7.27.3.11 uv_msg_type . . . 215
7.27.3.12 uv_status . . . 215
7.27.3.13 uv_task_cmd . . . 215
7.27.3.14 uv_task_id . . . 215
7.27.3.15 uv_task_msg . . . 215
7.27.3.16 uv_timespan_ms . . . 216
7.27.4 Enumeration Type Documentation . . . 216
7.27.4.1 access_control_t . . . 216
7.27.4.2 data_type . . . 216
7.27.4.3 uv_driving_mode_t . . . 217
7.27.4.4 uv_external_device . . . 217
7.27.4.5 uv_msg_type_t . . . 217
7.27.4.6 uv_status_t . . . 218
7.27.5 Function Documentation . . . 218
7.27.5.1 __uvInitPanic() . . . 218
7.27.5.2 uvInit() . . . 219
7.27.5.3 uvIsPTRValid() . . . 221
7.27.6 Variable Documentation . . . 221
7.27.6.1 global_context . . . 221
7.28 Core/Inc/uvfr_vehicle_commands.h File Reference . . . 221
7.28.1 Macro Definition Documentation . . . 222
7.28.1.1 uvHonkHorn . . . 222
7.28.1.2 uvOpenSDC [1/2] . . . 222
7.28.1.3 uvOpenSDC [2/2] . . . 222
7.28.1.4 uvSilenceHorn [1/2] . . . 222
7.28.1.5 uvSilenceHorn [2/2] . . . 223
7.28.1.6 uvStartCoolantPump . . . 223
7.28.1.7 uvStartFans . . . 223
7.28.1.8 uvStopCoolantPump . . . 223
7.28.1.9 uvStopFans . . . 223

7.28.2 Function Documentation . . . 223
7.28.2.1 _uvCloseSDC_canBased() . . . 223
7.28.2.2 _uvHonkHorn_canBased() . . . 224
7.28.2.3 _uvOpenSDC_canBased() . . . 224
7.28.2.4 _uvSilenceHorn_canBased() . . . 224
7.28.2.5 _uvStartCoolantPump_canBased() . . . 224
7.28.2.6 _uvStopCoolantPump_canBased() . . . 224
7.28.2.7 uvSecureVehicle() . . . 224
7.29 Core/Src/adc.c File Reference . . . 225
7.29.1 Detailed Description . . . 225
7.29.2 Function Documentation . . . 225
7.29.2.1 HAL_ADC_MspDeInit() . . . 225
7.29.2.2 HAL_ADC_MspInit() . . . 226
7.29.2.3 MX_ADC1_Init() . . . 226
7.29.2.4 MX_ADC2_Init() . . . 226
7.29.3 Variable Documentation . . . 227
7.29.3.1 hadc1 . . . 227
7.29.3.2 hadc2 . . . 227
7.29.3.3 hdma_adc1 . . . 227
7.30 Core/Src/bms.c File Reference . . . 227
7.30.1 Function Documentation . . . 228
7.30.1.1 BMS_Init() . . . 228
7.31 Core/Src/can.c File Reference . . . 228
7.31.1 Detailed Description . . . 229
7.31.2 Macro Definition Documentation . . . 229
7.31.2.1 HAL_CAN_ERROR_INVALID_CALLBACK . . . 230
7.31.2.2 table_size . . . 230
7.31.3 Typedef Documentation . . . 230
7.31.3.1 CAN_Callback . . . 230
7.31.4 Function Documentation . . . 230
7.31.4.1 __uvCANtxCritSection() . . . 230
7.31.4.2 callFunctionFromCANid() . . . 231
7.31.4.3 CANbusRxSvcDaemon() . . . 231
7.31.4.4 CANbusTxSvcDaemon() . . . 231
7.31.4.5 generateHash() . . . 232
7.31.4.6 HAL_CAN_MspDeInit() . . . 232
7.31.4.7 HAL_CAN_MspInit() . . . 232
7.31.4.8 HAL_CAN_RxFifo0MsgPendingCallback() . . . 232
7.31.4.9 HAL_CAN_RxFifo1MsgPendingCallback() . . . 233
7.31.4.10 handleCANbusError() . . . 233
7.31.4.11 MX_CAN2_Init() . . . 233
7.31.4.12 nuke_hash_table() . . . 233

7.31.5 Variable Documentation . . . 233
7.31.5.1 callback_table_mutex . . . 234
7.31.5.2 CAN_callback_table . . . 234
7.31.5.3 hcan2 . . . 234
7.31.5.4 Rx_msg_queue . . . 234
7.31.5.5 Tx_msg_queue . . . 235
7.32 Core/Src/constants.c File Reference . . . 235
7.32.1 Variable Documentation . . . 235
7.32.1.1 RxData . . . 235
7.32.1.2 RxHeader . . . 235
7.32.1.3 TxData . . . 236
7.32.1.4 TxHeader . . . 236
7.32.1.5 TxMailbox . . . 236
7.33 Core/Src/daq.c File Reference . . . 236
7.33.1 Macro Definition Documentation . . . 237
7.33.1.1 _SRC_UVFR_DAQ . . . 237
7.33.2 Function Documentation . . . 237
7.33.2.1 daqMasterTask() . . . 237
7.33.2.2 daqSubTask() . . . 238
7.33.2.3 deleteDaqSubTask() . . . 238
7.33.2.4 deleteParamList() . . . 238
7.33.2.5 initDaqTask() . . . 238
7.33.2.6 startDaqSubTasks() . . . 238
7.33.2.7 stopDaqSubTasks() . . . 239
7.33.3 Variable Documentation . . . 239
7.33.3.1 param_LUT . . . 239
7.34 Core/Src/dash.c File Reference . . . 239
7.34.1 Function Documentation . . . 239
7.34.1.1 Update_Batt_Temp() . . . 239
7.34.1.2 Update_RPM() . . . 240
7.34.1.3 Update_State_Of_Charge() . . . 240
7.35 Core/Src/dma.c File Reference . . . 240
7.35.1 Detailed Description . . . 240
7.35.2 Function Documentation . . . 241
7.35.2.1 MX_DMA_Init() . . . 241
7.36 Core/Src/driving_loop.c File Reference . . . 241
7.36.1 Detailed Description . . . 241
7.36.2 Function Documentation . . . 242
7.36.2.1 initDrivingLoop() . . . 242
7.36.2.2 StartDrivingLoop() . . . 242
7.36.3 Variable Documentation . . . 243
7.36.3.1 adc1_APPS1 . . . 243

7.36.3.2 adc1_APPS2 . . . 243
7.36.3.3 adc1_BPS1 . . . 243
7.36.3.4 adc1_BPS2 . . . 244
7.37 Core/Src/freertos.c File Reference . . . 244
7.37.1 Function Documentation . . . 244
7.37.1.1 MX_FREERTOS_Init() . . . 244
7.37.1.2 StartDefaultTask() . . . 245
7.37.1.3 vApplicationGetIdleTaskMemory() . . . 245
7.37.1.4 vApplicationGetTimerTaskMemory() . . . 246
7.37.1.5 vApplicationIdleHook() . . . 246
7.37.1.6 vApplicationMallocFailedHook() . . . 246
7.37.1.7 vApplicationStackOverflowHook() . . . 246
7.37.1.8 vApplicationTickHook() . . . 246
7.37.2 Variable Documentation . . . 247
7.37.2.1 defaultTaskHandle . . . 247
7.37.2.2 init_settings . . . 247
7.37.2.3 init_task_handle . . . 247
7.37.2.4 xIdleStack . . . 247
7.37.2.5 xIdleTaskTCBBuffer . . . 248
7.37.2.6 xTimerStack . . . 248
7.37.2.7 xTimerTaskTCBBuffer . . . 248
7.38 Core/Src/gpio.c File Reference . . . 248
7.38.1 Detailed Description . . . 248
7.38.2 Function Documentation . . . 249
7.38.2.1 MX_GPIO_Init() . . . 249
7.39 Core/Src/imd.c File Reference . . . 249
7.39.1 Function Documentation . . . 250
7.39.1.1 IMD_Check_Battery_Voltage() . . . 250
7.39.1.2 IMD_Check_Error_Flags() . . . 250
7.39.1.3 IMD_Check_Isolation_Capacitances() . . . 251
7.39.1.4 IMD_Check_Isolation_Resistances() . . . 251
7.39.1.5 IMD_Check_Isolation_State() . . . 251
7.39.1.6 IMD_Check_Max_Battery_Working_Voltage() . . . 251
7.39.1.7 IMD_Check_Part_Name() . . . 252
7.39.1.8 IMD_Check_Safety_Touch_Current() . . . 252
7.39.1.9 IMD_Check_Safety_Touch_Energy() . . . 252
7.39.1.10 IMD_Check_Serial_Number() . . . 252
7.39.1.11 IMD_Check_Status_Bits() . . . 253
7.39.1.12 IMD_Check_Temperature() . . . 253
7.39.1.13 IMD_Check_Uptime() . . . 253
7.39.1.14 IMD_Check_Version() . . . 253
7.39.1.15 IMD_Check_Voltages_Vp_and_Vn() . . . 254

7.39.1.16 IMD_Parse_Message() . . . 254
7.39.1.17 IMD_Request_Status() . . . 254
7.39.1.18 IMD_Startup() . . . 254
7.39.1.19 initIMD() . . . 255
7.39.2 Variable Documentation . . . 255
7.39.2.1 IMD_error_flags_requested . . . 255
7.39.2.2 IMD_Expected_Part_Name . . . 255
7.39.2.3 IMD_Expected_Serial_Number . . . 255
7.39.2.4 IMD_Expected_Version . . . 256
7.39.2.5 IMD_High_Uncertainty . . . 256
7.39.2.6 IMD_Part_Name_0_Set . . . 256
7.39.2.7 IMD_Part_Name_1_Set . . . 256
7.39.2.8 IMD_Part_Name_2_Set . . . 256
7.39.2.9 IMD_Part_Name_3_Set . . . 257
7.39.2.10 IMD_Part_Name_Set . . . 257
7.39.2.11 IMD_Read_Part_Name . . . 257
7.39.2.12 IMD_Read_Serial_Number . . . 257
7.39.2.13 IMD_Read_Version . . . 257
7.39.2.14 IMD_Serial_Number_0_Set . . . 258
7.39.2.15 IMD_Serial_Number_1_Set . . . 258
7.39.2.16 IMD_Serial_Number_2_Set . . . 258
7.39.2.17 IMD_Serial_Number_3_Set . . . 258
7.39.2.18 IMD_Serial_Number_Set . . . 258
7.39.2.19 IMD_status_bits . . . 259
7.39.2.20 IMD_Temperature . . . 259
7.39.2.21 IMD_Version_0_Set . . . 259
7.39.2.22 IMD_Version_1_Set . . . 259
7.39.2.23 IMD_Version_2_Set . . . 259
7.39.2.24 IMD_Version_Set . . . 260
7.40 Core/Src/main.c File Reference . . . 260
7.40.1 Detailed Description . . . 261
7.40.2 Macro Definition Documentation . . . 261
7.40.2.1 DEBUG_CAN_IN_MAIN . . . 261
7.40.3 Function Documentation . . . 261
7.40.3.1 Error_Handler() . . . 261
7.40.3.2 HAL_ADC_ConvCpltCallback() . . . 262
7.40.3.3 HAL_ADC_LevelOutOfWindowCallback() . . . 262
7.40.3.4 HAL_GPIO_EXTI_Callback() . . . 262
7.40.3.5 HAL_TIM_PeriodElapsedCallback() . . . 262
7.40.3.6 main() . . . 263
7.40.3.7 MX_FREERTOS_Init() . . . 263
7.40.3.8 SystemClock_Config() . . . 264

7.40.4 Variable Documentation . . . 264
7.40.4.1 adc1_APPS1 . . . 264
7.40.4.2 adc1_APPS2 . . . 265
7.40.4.3 adc1_BPS1 . . . 265
7.40.4.4 adc1_BPS2 . . . 265
7.40.4.5 adc2_CoolantFlow . . . 265
7.40.4.6 adc2_CoolantTemp . . . 265
7.40.4.7 adc_buf1 . . . 266
7.40.4.8 adc_buf2 . . . 266
7.41 Core/Src/motor_controller.c File Reference . . . 266
7.41.1 Function Documentation . . . 267
7.41.1.1 MC_Check_Error_Warning() . . . 267
7.41.1.2 MC_Check_Firmware() . . . 267
7.41.1.3 MC_Check_Serial_Number() . . . 267
7.41.1.4 MC_Request_Data() . . . 267
7.41.1.5 MC_Startup() . . . 268
7.41.1.6 MC_Validate() . . . 268
7.41.1.7 MotorControllerErrorHandler() . . . 268
7.41.1.8 MotorControllerSpinTest() . . . 269
7.41.1.9 Parse_Bamocar_Response() . . . 269
7.41.1.10 WaitFor_CAN_Response() . . . 270
7.41.2 Variable Documentation . . . 270
7.41.2.1 canRxQueue . . . 270
7.41.2.2 canTxQueue . . . 270
7.41.2.3 max_motor_speed . . . 271
7.41.2.4 mc_default_settings . . . 271
7.41.2.5 MC_Expected_FW_Version . . . 271
7.41.2.6 MC_Expected_Serial_Number . . . 271
7.42 Core/Src/odometer.c File Reference . . . 271
7.42.1 Function Documentation . . . 272
7.42.1.1 initOdometer() . . . 272
7.42.1.2 odometerTask() . . . 272
7.43 Core/Src/oled.c File Reference . . . 273
7.44 Core/Src/pdu.c File Reference . . . 273
7.44.1 Function Documentation . . . 273
7.44.1.1 initPDU() . . . 273
7.44.1.2 PDU_disable_brake_light() . . . 274
7.44.1.3 PDU_disable_coolant_pump() . . . 274
7.44.1.4 PDU_disable_cooling_fans() . . . 274
7.44.1.5 PDU_disable_motor_controller() . . . 274
7.44.1.6 PDU_disable_shutdown_circuit() . . . 274
7.44.1.7 PDU_enable_brake_light() . . . 275

7.44.1.8 PDU_enable_coolant_pump() . . . 275
7.44.1.9 PDU_enable_cooling_fans() . . . 275
7.44.1.10 PDU_enable_motor_controller() . . . 275
7.44.1.11 PDU_enable_shutdown_circuit() . . . 275
7.44.1.12 PDU_speaker_chirp() . . . 276
7.45 Core/Src/rb_tree.c File Reference . . . 276
7.45.1 Function Documentation . . . 276
7.45.1.1 checkBlackHeight() . . . 277
7.45.1.2 checkOrder() . . . 277
7.45.1.3 deleteRepair() . . . 277
7.45.1.4 destroyAllNodes() . . . 277
7.45.1.5 insertRepair() . . . 278
7.45.1.6 print() . . . 278
7.45.1.7 rb_apply() . . . 278
7.45.1.8 rbCheckBlackHeight() . . . 278
7.45.1.9 rbCheckOrder() . . . 279
7.45.1.10 rbCreate() . . . 279
7.45.1.11 rbDelete() . . . 279
7.45.1.12 rbDestroy() . . . 279
7.45.1.13 rbFind() . . . 280
7.45.1.14 rbInsert() . . . 280
7.45.1.15 rbPrint() . . . 280
7.45.1.16 rbSuccessor() . . . 280
7.45.1.17 rotateLeft() . . . 281
7.45.1.18 rotateRight() . . . 281
7.46 Core/Src/spi.c File Reference . . . 281
7.46.1 Detailed Description . . . 282
7.46.2 Function Documentation . . . 282
7.46.2.1 HAL_SPI_MspDeInit() . . . 282
7.46.2.2 HAL_SPI_MspInit() . . . 282
7.46.2.3 MX_SPI1_Init() . . . 282
7.46.3 Variable Documentation . . . 283
7.46.3.1 hspi1 . . . 283
7.47 Core/Src/stm32f4xx_hal_msp.c File Reference . . . 283
7.47.1 Detailed Description . . . 283
7.47.2 Function Documentation . . . 283
7.47.2.1 HAL_MspInit() . . . 283
7.48 Core/Src/stm32f4xx_hal_timebase_tim.c File Reference . . . 284
7.48.1 Detailed Description . . . 284
7.48.2 Function Documentation . . . 284
7.48.2.1 HAL_InitTick() . . . 284
7.48.2.2 HAL_ResumeTick() . . . 285

7.48.2.3 HAL_SuspendTick() . . . 285
7.48.3 Variable Documentation . . . 286
7.48.3.1 htim1 . . . 286
7.49 Core/Src/stm32f4xx_it.c File Reference . . . 286
7.49.1 Detailed Description . . . 287
7.49.2 Function Documentation . . . 287
7.49.2.1 BusFault_Handler() . . . 287
7.49.2.2 CAN2_RX0_IRQHandler() . . . 288
7.49.2.3 CAN2_RX1_IRQHandler() . . . 288
7.49.2.4 CAN2_TX_IRQHandler() . . . 288
7.49.2.5 DebugMon_Handler() . . . 288
7.49.2.6 DMA2_Stream0_IRQHandler() . . . 289
7.49.2.7 EXTI0_IRQHandler() . . . 289
7.49.2.8 HardFault_Handler() . . . 289
7.49.2.9 MemManage_Handler() . . . 289
7.49.2.10 NMI_Handler() . . . 290
7.49.2.11 TIM1_UP_TIM10_IRQHandler() . . . 290
7.49.2.12 UsageFault_Handler() . . . 290
7.49.3 Variable Documentation . . . 290
7.49.3.1 hcan2 . . . 290
7.49.3.2 hdma_adc1 . . . 291
7.49.3.3 htim1 . . . 291
7.50 Core/Src/syscalls.c File Reference . . . 291
7.50.1 Detailed Description . . . 292
7.50.2 Function Documentation . . . 292
7.50.2.1 __attribute__() . . . 292
7.50.2.2 __io_getchar() . . . 292
7.50.2.3 __io_putchar() . . . 293
7.50.2.4 _close() . . . 293
7.50.2.5 _execve() . . . 293
7.50.2.6 _exit() . . . 293
7.50.2.7 _fork() . . . 293
7.50.2.8 _fstat() . . . 294
7.50.2.9 _getpid() . . . 294
7.50.2.10 _isatty() . . . 294
7.50.2.11 _kill() . . . 294
7.50.2.12 _link() . . . 294
7.50.2.13 _lseek() . . . 295
7.50.2.14 _open() . . . 295
7.50.2.15 _stat() . . . 295
7.50.2.16 _times() . . . 295
7.50.2.17 _unlink() . . . 295

7.50.2.18 _wait() . . . 296
7.50.2.19 initialise_monitor_handles() . . . 296
7.50.3 Variable Documentation . . . 296
7.50.3.1 environ . . . 296
7.51 Core/Src/sysmem.c File Reference . . . 296
7.51.1 Detailed Description . . . 297
7.51.2 Function Documentation . . . 297
7.51.2.1 _sbrk() . . . 297
7.51.3 Variable Documentation . . . 298
7.51.3.1 __sbrk_heap_end . . . 298
7.52 Core/Src/system_stm32f4xx.c File Reference . . . 298
7.52.1 Detailed Description . . . 299
7.53 Core/Src/temp_monitoring.c File Reference . . . 299
7.53.1 Function Documentation . . . 299
7.53.1.1 initTempMonitor() . . . 300
7.53.1.2 tempMonitorTask() . . . 300
7.53.1.3 testfunc() . . . 300
7.53.1.4 testfunc2() . . . 301
7.54 Core/Src/tim.c File Reference . . . 301
7.54.1 Detailed Description . . . 301
7.54.2 Function Documentation . . . 301
7.54.2.1 HAL_TIM_Base_MspDeInit() . . . 302
7.54.2.2 HAL_TIM_Base_MspInit() . . . 302
7.54.2.3 MX_TIM3_Init() . . . 302
7.54.3 Variable Documentation . . . 302
7.54.3.1 htim3 . . . 302
7.55 Core/Src/uvfr_settings.c File Reference . . . 302
7.55.1 Macro Definition Documentation . . . 303
7.55.1.1 SRC_UVFR_SETTINGS_C_ . . . 303
7.55.2 Function Documentation . . . 303
7.55.2.1 nukeSettings() . . . 303
7.55.2.2 setupDefaultSettings() . . . 304
7.55.2.3 uvSettingsInit() . . . 304
7.55.2.4 uvSettingsProgrammerTask() . . . 304
7.55.3 Variable Documentation . . . 304
7.55.3.1 current_vehicle_settings . . . 304
7.56 Core/Src/uvfr_state_engine.c File Reference . . . 305
7.56.1 Detailed Description . . . 307
7.56.2 Macro Definition Documentation . . . 307
7.56.2.1 UVFR_STATE_MACHINE_IMPLIMENTATION . . . 307
7.57 Core/Src/uvfr_utils.c File Reference . . . 307
7.57.1 Macro Definition Documentation . . . 308

7.57.1.1 UV_UTILS_SRC_IMPLIMENTATION . . . . . 308
7.57.2 Function Documentation . . . . . 308
7.57.2.1 __uvFreeCritSection() . . . . . 308
7.57.2.2 __uvFreeOS() . . . . . 308
7.57.2.3 __uvInitPanic() . . . . . 309
7.57.2.4 __uvMallocCritSection() . . . . . 309
7.57.2.5 __uvMallocOS() . . . . . 309
7.57.2.6 setup_extern_devices() . . . . . 309
7.57.2.7 uvInit() . . . . . 310
7.57.2.8 uvIsPTRValid() . . . . . 312
7.57.2.9 uvSysResetDaemon() . . . . . 312
7.57.2.10 uvUtilsReset() . . . . . 312
7.57.3 Variable Documentation . . . . . 313
7.57.3.1 init_task_handle . . . . . 313
7.57.3.2 reset_handle . . . . . 313
7.57.3.3 TxData . . . . . 313
7.58 Core/Src/uvfr_vehicle_commands.c File Reference . . . . . 313
7.58.1 Function Documentation . . . . . 313
7.58.1.1 uvSecureVehicle() . . . . . 313

**Index** **315**

# 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 . . . 15
State Engine Internals . . . 29
UVFR Utilities . . . 41
Utility Macros . . . 42
UVFR Vehicle Commands . . . 48
UVFR CANbus API . . . 49
CMSIS . . . 50
Stm32f4xx_system . . . 51
STM32F4xx_System_Private_Includes . . . 52
STM32F4xx_System_Private_TypesDefinitions . . . 53
STM32F4xx_System_Private_Defines . . . 54
STM32F4xx_System_Private_Macros . . . 55
STM32F4xx_System_Private_Variables . . . 56
STM32F4xx_System_Private_FunctionPrototypes . . . 57
STM32F4xx_System_Private_Functions . . . 58

# Chapter 3

# Data Structure Index

## 3.1 Data Structures

Here are the data structures with brief descriptions:


access_control_info . . . 59
bms_settings_t . . . 60
CAN_Callback . . . 60
daq_child_task . . . 61
daq_datapoint
    This struct holds info of what needs to be logged . . . 63
daq_loop_args . . . 64
daq_param_list_node . . . 65
driving_loop_args . . . 66
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 71
drivingMode
    This is where the driving mode and the drivingModeParams are at . . . 71
drivingModeParams
    This struct is designed to hold information about each drivingmode's map params . . . 73
exp_torque_map_args
    Struct to hold parameters used in an exponential torque map . . . 74
linear_torque_map_args . . . 74
motor_controllor_settings . . . 75
p_status . . . 77
rbnode
    Node of a Red-Black binary search tree . . . 78
rbtree
    Struct representing a binary search tree . . . 79
s_curve_torque_map_args
    Struct for s-curve parameters for torque . . . 82
state_change_daemon_args . . . 83
task_management_info
    Struct to contain data about a parent task . . . 84
task_status_block
    Information about the task . . . 85
uv_binary_semaphore_info . . . 85
uv_CAN_msg
    Representative of a CAN message . . . 86

uv_init_struct . . . . . 88
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 . . . . . 88
uv_init_task_response
Struct representing the response of one of the initialization tasks . . . . . 89
uv_internal_params
Data used by the uvfr_utils library to do what it needs to do :) . . . . . 91
uv_mutex_info . . . . . 92
uv_os_settings
Settings that dictate state engine behavior . . . . . 93
uv_scd_response . . . . . 94
uv_semaphore_info . . . . . 95
uv_task_info
This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine . . . . . 96
uv_task_msg_t
Struct containing a message between two tasks . . . . . 101
uv_vehicle_settings . . . . . 103
veh_gen_info . . . . . 105

# 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 . . . 107
Core/Inc/bms.h . . . 111
Core/Inc/can.h
: This file contains all the function prototypes for the can.c file . . . 112
Core/Inc/constants.h . . . 116
Core/Inc/daq.h . . . 118
Core/Inc/dash.h . . . 122
Core/Inc/dma.h
: This file contains all the function prototypes for the dma.c file . . . 123
Core/Inc/driving_loop.h . . . 124
Core/Inc/errorLUT.h . . . 129
Core/Inc/FreeRTOSConfig.h . . . 129
Core/Inc/gpio.h
: This file contains all the function prototypes for the gpio.c file . . . 141
Core/Inc/imd.h . . . 141
Core/Inc/main.h
: : Header for main.c file. This file contains the common defines of the application . . . 149
Core/Inc/motor_controller.h . . . 152
Core/Inc/odometer.h . . . 159
Core/Inc/oled.h . . . 160
Core/Inc/pdu.h . . . 162
Core/Inc/rb_tree.h . . . 166
Core/Inc/spi.h
: This file contains all the function prototypes for the spi.c file . . . 172
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 . . . 173
Core/Inc/stm32f4xx_it.h
: This file contains the headers of the interrupt handlers . . . 194
Core/Inc/temp_monitoring.h . . . 198
Core/Inc/tim.h
: This file contains all the function prototypes for the tim.c file . . . 199
Core/Inc/uvfr_global_config.h . . . 200
Core/Inc/uvfr_settings.h . . . 201

Core/Inc/uvfr_state_engine.h . . . 203
Core/Inc/uvfr_utils.h . . . 209
Core/Inc/uvfr_vehicle_commands.h . . . 221
Core/Src/adc.c
This file provides code for the configuration of the ADC instances . . . 225
Core/Src/bms.c . . . 227
Core/Src/can.c
This file provides code for the configuration of the CAN instances . . . 228
Core/Src/constants.c . . . 235
Core/Src/daq.c . . . 236
Core/Src/dash.c . . . 239
Core/Src/dma.c
This file provides code for the configuration of all the requested memory to memory DMA transfers . . . 240
Core/Src/driving_loop.c
File containing the meat and potatoes driving loop thread, and all supporting functions . . . 241
Core/Src/freertos.c . . . 244
Core/Src/gpio.c
This file provides code for the configuration of all used GPIO pins . . . 248
Core/Src/imd.c . . . 249
Core/Src/main.c
: Main program body . . . 260
Core/Src/motor_controller.c . . . 266
Core/Src/odometer.c . . . 271
Core/Src/oled.c . . . 273
Core/Src/pdu.c . . . 273
Core/Src/rb_tree.c . . . 276
Core/Src/spi.c
This file provides code for the configuration of the SPI instances . . . 281
Core/Src/stm32f4xx_hal_msp.c
This file provides code for the MSP Initialization and de-Initialization codes . . . 283
Core/Src/stm32f4xx_hal_timebase_tim.c
HAL time base based on the hardware TIM . . . 284
Core/Src/stm32f4xx_it.c
Interrupt Service Routines . . . 286
Core/Src/syscalls.c
STM32CubeIDE Minimal System calls file . . . 291
Core/Src/sysmem.c
STM32CubeIDE System Memory calls file . . . 296
Core/Src/system_stm32f4xx.c
CMSIS Cortex-M4 Device Peripheral Access Layer System Source File . . . 298
Core/Src/temp_monitoring.c . . . 299
Core/Src/tim.c
This file provides code for the configuration of the TIM instances . . . 301
Core/Src/uvfr_settings.c . . . 302
Core/Src/uvfr_state_engine.c
File containing the implementation of the vehicle's state engine and error handling infrastructure 305
Core/Src/uvfr_utils.c . . . 307
Core/Src/uvfr_vehicle_commands.c . . . 313

# 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*
- int compareTaskByName (uv_task_info *t1, uv_task_info *t2)

### 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
- 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
- rbtree * task_name_tree
- uv_os_settings default_os_settings

## 5.1.1 Detailed Description

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

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

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

## 5.1.2 Macro Definition Documentation

### 5.1.2.1 MAX_NUM_MANAGED_TASKS

```
#define MAX_NUM_MANAGED_TASKS 16
```

Definition at line 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 compareTaskByName()

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

Definition at line 1320 of file uvfr_state_engine.c.

References uv_task_info::task_name.

#### 5.1.4.2 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 1261 of file uvfr_state_engine.c.

References __uvInitPanic(), _task_register, uv_task_info::active_states, CAN_RX_DAEMON_NAME, CAN_TX_DAEMON_NAME, CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), uv_task_info::task_function, task_management_info::task_handle, uv_task_info::task_name, uvCreateServiceTask(), 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 uvValidate↩ ManagedTasks().

#### 5.1.5.3 _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(), uvCreate↩ ServiceTask(), uvCreateTask(), uvInitStateEngine(), and uvSVCTaskManager().

#### 5.1.5.4 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 47 of file uvfr_state_engine.c.

Referenced by setupDefaultSettings().

#### 5.1.5.5 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().

#### 5.1.5.6 SCD_active

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

Definition at line 34 of file uvfr_state_engine.c.

Referenced by _stateChangeDaemon().

#### 5.1.5.7 scd_handle_ptr

```
TaskHandle_t* scd_handle_ptr
```

Definition at line 31 of file uvfr_state_engine.c.

#### 5.1.5.8 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().

#### 5.1.5.9 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().

#### 5.1.5.10 task_manager

```
uv_task_info* task_manager = NULL
```

Definition at line 42 of file uvfr_state_engine.c.

Referenced by uvInitStateEngine(), and uvStartStateMachine().

#### 5.1.5.11 task_name_lut

```
rbtree* task_name_lut = NULL
```

Definition at line 37 of file uvfr_state_engine.c.

#### 5.1.5.12 task_name_tree

```
rbtree* task_name_tree
```

Definition at line 45 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(), testfunc(), 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 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_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

- 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 gnarlyness 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 neccessarily 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

| | |
|---|---|
| *x* | |
| *t* | 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

| | |
|---|---|
| *x* | |
| *lasttim* | is the variable storing the last delay time. |
| *per* | 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 neccessary information about an RTOS task that will be managed by uvfr_state_↩ engine.

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

#### 5.2.3.8 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

| | |
|---|---|
| *state* | is a member of uv_status, and therefore a power of two |

Return values

| | |
|---|---|
| *returns* | a memeber of uv_status 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 89 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(), testfunc(), 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. Theres some gnarlyness here, but not unacceptable levels. Pray this thing doesn't hang itself.

Do not exceed the number of tasks available

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

Definition at line 252 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 242 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 154 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 182 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, uvSVCTaskManager(), 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 __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 uvSendTaskStatusReport (uv_task_info *t)
- void _stateChangeDaemon (void *args) PRIVILEGED_FUNCTION

  *This collects all the data changing from different tasks, and makes sure that everything works properly.*
- uv_status 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.*
- 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 670 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

| | |
|---|---|
| *args* | This accepts a `void*` 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 neccessary
- bring the task state into the correct state

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

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

```
 */
    uv_scd_response* response = NULL;
    for(int i = 0; i < _LONGEST_SC_TIME/_SC_DAEMON_PERIOD; i++){ //This loop verifies to make sure things
      are actually chillin
        vTaskDelay(_SC_DAEMON_PERIOD);
        for(int j = 0;j<10;j++){ //What kinda magic number is this?  Why 10?
            if(xQueueReceive(state_change_queue,&response,1) == pdPASS){
                if(response == NULL){//definately not supposed to happen
                    uvPanic("null scd response",0);
                }
                if(proccessSCDMsg(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 845 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 311 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_function, uv_task_info::task_name, UV_ERROR, and UV_OK.

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 298 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 436 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 688 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 CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), daqMasterTask(), odometerTask(), StartDrivingLoop(), and tempMonitorTask().

#### 5.3.2.7 proccessSCDMsg()

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

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.

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 772 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 729 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 CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), 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 541 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 1115 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, UV_TASK_NOT_STARTED, and UV_TASK_SCD_IGNORE.

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 1209 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 481 of file uvfr_state_engine.c.

References uv_task_info::cmd_data, uv_task_info::task_handle, uv_task_info::task_id, uv_task_info::task_state, UV_ABORTED, UV_ERROR, UV_KILL_CMD, UV_OK, UV_TASK_DELETED, UV_TASK_NOT_STARTED, UV_↩ TASK_SUSPENDED, and uvTaskIsDelaying.

Referenced by _stateChangeDaemon(), and killEmAll().

#### 5.3.2.13 uvGetTaskFromName()

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

Sometimes you just gottta deal with it lol

Definition at line 1356 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

| | |
|---|---|
| *t_handle* | 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 1368 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 1026 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 457 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 neccessary 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 1237 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 *            *
```

#### 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, becuase then the task will go out of scope, and changing the task handle to a new instance will result in the task never being de-initialized, therefore causing a memory leak. We want to call `vTaskResume` instead, and just boot the task back into existence.

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

Definition at line 368 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 1194 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

| | |
|---|---|
| *tracker* | is a pointer to an int. If the task actually suspends, we update the tracker, since no further action is needed. |
| *t* | is a pointer to a uv_task_info struct. |

Definition at line 580 of file uvfr_state_engine.c.

References uv_task_info::cmd_data, uv_task_info::task_handle, uv_task_info::task_id, uv_task_info::task_state, UV_ERROR, UV_OK, UV_SUSPEND_CMD, UV_TASK_DELETED, UV_TASK_NOT_STARTED, UV_TASK_SUSPENDED, and uvTaskIsDelaying.

Referenced by _stateChangeDaemon().

#### 5.3.2.24 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 637 of file uvfr_state_engine.c.

References uv_task_info::task_flags, UV_ERROR, UV_OK, 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 us of utmost importance that this one DOES NOT CRASH. EVER. Wait for incoming instructions from tasks

Definition at line 1042 of file uvfr_state_engine.c.

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 346 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&0x00F↩F0000)>>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&0x00F↩F0000)>>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

| | |
|---|---|
| *x* | represents the value you want to edit. Can be any signed or unsigned integer type. |
| *msk* | Bits of X will only be altered if the matching bit of msk is a 1 |
| *data* | 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 UVFR CANbus API

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

### Functions

- void insertCANMessageHandler (uint32_t id, void *handlerfunc)

  *Function to insert an id and function into the lookup table of callback functions.*
- uv_status uvSendCanMSG (uv_CAN_msg *tx_msg)

  *Function to send CAN message.*

### 5.7.1 Detailed Description

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

### 5.7.2 Function Documentation

#### 5.7.2.1 insertCANMessageHandler()

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

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

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

Definition at line 395 of file can.c.

References callback_table_mutex, CAN_callback_table, CAN_Callback::CAN_id, CAN_Callback::function, generateHash(), and CAN_Callback::next.

Referenced by tempMonitorTask().

#### 5.7.2.2 uvSendCanMSG()

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

Function to send CAN message.

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

Definition at line 513 of file can.c.

References __uvCANtxCritSection(), Tx_msg_queue, UV_ERROR, and UV_OK.

Referenced by tempMonitorTask(), and testfunc2().

## 5.8 CMSIS

### Modules

- Stm32f4xx_system

### 5.8.1 Detailed Description

## 5.9 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

### 5.9.1 Detailed Description

## 5.10 STM32F4xx_System_Private_Includes

### Macros

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

### 5.10.1 Detailed Description

### 5.10.2 Macro Definition Documentation

#### 5.10.2.1 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.

#### 5.10.2.2 HSI_VALUE

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

Value of the Internal oscillator in Hz

Definition at line 55 of file system_stm32f4xx.c.

## 5.11 STM32F4xx_System_Private_TypesDefinitions

## 5.12 STM32F4xx_System_Private_Defines

## 5.13 STM32F4xx_System_Private_Macros

## 5.14 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}

### 5.14.1 Detailed Description

### 5.14.2 Variable Documentation

#### 5.14.2.1 AHBPrescTable

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

Definition at line 138 of file system_stm32f4xx.c.

Referenced by SystemCoreClockUpdate().

#### 5.14.2.2 APBPrescTable

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

Definition at line 139 of file system_stm32f4xx.c.

#### 5.14.2.3 SystemCoreClock

```
uint32_t SystemCoreClock = 16000000
```

Definition at line 137 of file system_stm32f4xx.c.

Referenced by SystemCoreClockUpdate().

## 5.15 STM32F4xx_System_Private_FunctionPrototypes

## 5.16 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.*

### 5.16.1 Detailed Description

### 5.16.2 Function Documentation

#### 5.16.2.1 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.

#### 5.16.2.2 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 254 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 256 of file uvfr_utils.h.

#### 6.1.2.2 mutex

`struct uv_mutex_info access_control_info::mutex`

Definition at line 255 of file uvfr_utils.h.

#### 6.1.2.3 semaphore

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

Definition at line 257 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 CAN_Callback Struct Reference

### Data Fields

- uint32_t CAN_id
- void * function
- struct CAN_Callback * next

### 6.3.1 Detailed Description

Definition at line 56 of file can.c.

### 6.3.2 Field Documentation

#### 6.3.2.1 CAN_id

```
uint32_t CAN_Callback::CAN_id
```

Definition at line 57 of file can.c.

Referenced by callFunctionFromCANid(), and insertCANMessageHandler().

#### 6.3.2.2 function

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

Definition at line 58 of file can.c.

Referenced by callFunctionFromCANid(), and insertCANMessageHandler().

#### 6.3.2.3 next

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

Definition at line 59 of file can.c.

Referenced by callFunctionFromCANid(), insertCANMessageHandler(), and nuke_hash_table().

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

- Core/Src/can.c

## 6.4 daq_child_task Struct Reference

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

## Data Fields

- struct rbnode treenode
- TaskHandle_t meta_task_handle
- daq_param_list_node ** param_list
- uint32_t period

### 6.4.1 Detailed Description

Definition at line 70 of file daq.h.

### 6.4.2 Field Documentation

#### 6.4.2.1 meta_task_handle

```
TaskHandle_t daq_child_task::meta_task_handle
```

Definition at line 72 of file daq.h.

#### 6.4.2.2 param_list

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

Definition at line 73 of file daq.h.

#### 6.4.2.3 period

```
uint32_t daq_child_task::period
```

Definition at line 74 of file daq.h.

#### 6.4.2.4 treenode

```
struct rbnode daq_child_task::treenode
```

Definition at line 71 of file daq.h.

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

- Core/Inc/daq.h

## 6.5 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

### 6.5.1 Detailed Description

This struct holds info of what needs to be logged.

Definition at line 56 of file daq.h.

### 6.5.2 Field Documentation

#### 6.5.2.1 can_id

`uint16_t daq_datapoint::can_id`

Definition at line 57 of file daq.h.

#### 6.5.2.2 period

`uint8_t daq_datapoint::period`

Definition at line 58 of file daq.h.

#### 6.5.2.3 type

`uint8_t daq_datapoint::type`

Definition at line 59 of file daq.h.

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

- Core/Inc/daq.h

## 6.6 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]

### 6.6.1 Detailed Description

Definition at line 62 of file daq.h.

### 6.6.2 Field Documentation

#### 6.6.2.1 datapoints

`daq_datapoint daq_loop_args::datapoints[MAX_LOGGABLE_PARAMS]`

Definition at line 67 of file daq.h.

#### 6.6.2.2 minimum_daq_period

`uint8_t daq_loop_args::minimum_daq_period`

Definition at line 64 of file daq.h.

#### 6.6.2.3 padding

`uint16_t daq_loop_args::padding`

Definition at line 65 of file daq.h.

#### 6.6.2.4 padding2

```
uint32_t daq_loop_args::padding2
```

Definition at line 66 of file daq.h.

#### 6.6.2.5 throttle_daq_to_preserve_performance

```
uint8_t daq_loop_args::throttle_daq_to_preserve_performance
```

Definition at line 63 of file daq.h.

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

- Core/Inc/daq.h

## 6.7 daq_param_list_node Struct Reference

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

### Data Fields

- uint16_t param_idx
- struct daq_param_list_node * next

### 6.7.1 Detailed Description

Definition at line 48 of file daq.h.

### 6.7.2 Field Documentation

#### 6.7.2.1 next

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

Definition at line 50 of file daq.h.

#### 6.7.2.2 param_idx

```
uint16_t daq_param_list_node::param_idx
```

Definition at line 49 of file daq.h.

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

- Core/Inc/daq.h

## 6.8 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]

### 6.8.1 Detailed Description

Definition at line 108 of file driving_loop.h.

### 6.8.2 Field Documentation

#### 6.8.2.1 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.

#### 6.8.2.2 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.

#### 6.8.2.3 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.

#### 6.8.2.4 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.

#### 6.8.2.5 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.

#### 6.8.2.6 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.

#### 6.8.2.7 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.

#### 6.8.2.8 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().

#### 6.8.2.9 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.

#### 6.8.2.10 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.

#### 6.8.2.11 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().

#### 6.8.2.12 dmodes

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

These are various driving modes

Definition at line 141 of file driving_loop.h.

#### 6.8.2.13 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.

#### 6.8.2.14 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().

#### 6.8.2.15 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().

#### 6.8.2.16 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().

#### 6.8.2.17 min_apps_offset

`uint16_t driving_loop_args::min_apps_offset`

minimum APPS offset

Definition at line 120 of file driving_loop.h.

#### 6.8.2.18 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.

#### 6.8.2.19 min_BPS_value

`uint16_t driving_loop_args::min_BPS_value`

are the brakes valid?

Definition at line 124 of file driving_loop.h.

#### 6.8.2.20 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.

#### 6.8.2.21 period

```
uint8_t driving_loop_args::period
```

how often does the driving loop execute

Definition at line 137 of file driving_loop.h.

#### 6.8.2.22 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

## 6.9 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>
```

### 6.9.1 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

## 6.10 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

### 6.10.1 Detailed Description

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

Definition at line 85 of file driving_loop.h.

### 6.10.2 Field Documentation

#### 6.10.2.1 control_map_fn

```
uint8_t drivingMode::control_map_fn
```

Definition at line 95 of file driving_loop.h.

#### 6.10.2.2 dm_name

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

Name of mode, 15 chars + /0

Definition at line 86 of file driving_loop.h.

#### 6.10.2.3 flags

```
uint16_t drivingMode::flags
```

Definition at line 92 of file driving_loop.h.

#### 6.10.2.4 map_fn_params

```
drivingModeParams drivingMode::map_fn_params
```

Definition at line 94 of file driving_loop.h.

#### 6.10.2.5 max_acc_pwr

```
uint32_t drivingMode::max_acc_pwr
```

Definition at line 87 of file driving_loop.h.

#### 6.10.2.6 max_current

```
uint32_t drivingMode::max_current
```

Definition at line 89 of file driving_loop.h.

#### 6.10.2.7 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

## 6.11 drivingModeParams Union Reference

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

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

### 6.11.1 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

## 6.12 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

### 6.12.1 Detailed Description

struct to hold parameters used in an exponential torque map

Definition at line 56 of file driving_loop.h.

### 6.12.2 Field Documentation

#### 6.12.2.1 gamma

`float exp_torque_map_args::gamma`

Definition at line 58 of file driving_loop.h.

#### 6.12.2.2 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

## 6.13 linear_torque_map_args Struct Reference

`#include <driving_loop.h>`

### Data Fields

- int32_t offset
- float slope

### 6.13.1 Detailed Description

Definition at line 48 of file driving_loop.h.

### 6.13.2 Field Documentation

#### 6.13.2.1 offset

```
int32_t linear_torque_map_args::offset
```

Definition at line 49 of file driving_loop.h.

#### 6.13.2.2 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

## 6.14 motor_controllor_settings Struct Reference

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

### Data Fields

- uint32_t can_id_tx
- uint32_t can_id_rx
- uint32_t mc_CAN_timeout
- uint8_t proportional_gain
- uint32_t integral_time_constant
- uint8_t integral_memory_max

### 6.14.1 Detailed Description

Definition at line 150 of file motor_controller.h.

### 6.14.2 Field Documentation

#### 6.14.2.1 can_id_rx

```
uint32_t motor_controllor_settings::can_id_rx
```

Definition at line 154 of file motor_controller.h.

#### 6.14.2.2 can_id_tx

```
uint32_t motor_controllor_settings::can_id_tx
```

Definition at line 152 of file motor_controller.h.

Referenced by MC_Request_Data(), and MotorControllerSpinTest().

#### 6.14.2.3 integral_memory_max

```
uint8_t motor_controllor_settings::integral_memory_max
```

Definition at line 163 of file motor_controller.h.

#### 6.14.2.4 integral_time_constant

```
uint32_t motor_controllor_settings::integral_time_constant
```

Definition at line 161 of file motor_controller.h.

#### 6.14.2.5 mc_CAN_timeout

```
uint32_t motor_controllor_settings::mc_CAN_timeout
```

Definition at line 156 of file motor_controller.h.

#### 6.14.2.6 proportional_gain

```
uint8_t motor_controllor_settings::proportional_gain
```

Definition at line 159 of file motor_controller.h.

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

- Core/Inc/motor_controller.h

## 6.15 p_status Struct Reference

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

### Data Fields

- uv_status peripheral_status
- TickType_t activation_time

### 6.15.1 Detailed Description

Definition at line 317 of file uvfr_utils.h.

### 6.15.2 Field Documentation

#### 6.15.2.1 activation_time

```
TickType_t p_status::activation_time
```

Definition at line 319 of file uvfr_utils.h.

#### 6.15.2.2 peripheral_status

```
uv_status p_status::peripheral_status
```

Definition at line 318 of file uvfr_utils.h.

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

- Core/Inc/uvfr_utils.h

## 6.16 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

### 6.16.1 Detailed Description

Node of a Red-Black binary search tree.

Definition at line 27 of file rb_tree.h.

### 6.16.2 Field Documentation

#### 6.16.2.1 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().

#### 6.16.2.2 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().

#### 6.16.2.3 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().

#### 6.16.2.4 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().

#### 6.16.2.5 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

## 6.17 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

### 6.17.1 Detailed Description

struct representing a binary search tree

Definition at line 39 of file rb_tree.h.

### 6.17.2 Field Documentation

#### 6.17.2.1 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().

#### 6.17.2.2 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().

#### 6.17.2.3 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().

#### 6.17.2.4 min

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

Pointer to minimum element

Definition at line 50 of file rb_tree.h.

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

#### 6.17.2.5 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().

#### 6.17.2.6 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.

#### 6.17.2.7 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

## 6.18 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]

### 6.18.1 Detailed Description

struct for s-curve parameters for torque

Definition at line 66 of file driving_loop.h.

### 6.18.2 Field Documentation

#### 6.18.2.1 a

```
int32_t s_curve_torque_map_args::a
```

Definition at line 67 of file driving_loop.h.

#### 6.18.2.2 b

```
int32_t s_curve_torque_map_args::b
```

Definition at line 68 of file driving_loop.h.

#### 6.18.2.3 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

## 6.19 state_change_daemon_args Struct Reference

### Data Fields

- TaskHandle_t meta_task_handle

### 6.19.1 Detailed Description

Definition at line 61 of file uvfr_state_engine.c.

### 6.19.2 Field Documentation

#### 6.19.2.1 meta_task_handle

```
TaskHandle_t state_change_daemon_args::meta_task_handle
```

Definition at line 62 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

## 6.20 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

### 6.20.1 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.

### 6.20.2 Field Documentation

#### 6.20.2.1 parent_msg_queue

`QueueHandle_t task_management_info::parent_msg_queue`

Definition at line 156 of file uvfr_state_engine.h.

#### 6.20.2.2 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

## 6.21 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

### 6.21.1 Detailed Description

Information about the task.

Definition at line 162 of file uvfr_state_engine.h.

### 6.21.2 Field Documentation

#### 6.21.2.1 task_high_water_mark

`uint32_t task_status_block::task_high_water_mark`

Definition at line 163 of file uvfr_state_engine.h.

#### 6.21.2.2 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

## 6.22 uv_binary_semaphore_info Struct Reference

`#include <uvfr_utils.h>`

### Data Fields

- SemaphoreHandle_t handle

### 6.22.1 Detailed Description

Definition at line 244 of file uvfr_utils.h.

### 6.22.2 Field Documentation

#### 6.22.2.1 handle

```
SemaphoreHandle_t uv_binary_semaphore_info::handle
```

Definition at line 245 of file uvfr_utils.h.

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

- Core/Inc/uvfr_utils.h

## 6.23 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]

### 6.23.1 Detailed Description

Representative of a CAN message.

Definition at line 270 of file uvfr_utils.h.

### 6.23.2 Field Documentation

#### 6.23.2.1 data

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

The actual data packet contained within the CAN message

Definition at line 277 of file uvfr_utils.h.

Referenced by __uvCANtxCritSection(), CANbusTxSvcDaemon(), HAL_CAN_RxFifo0MsgPendingCallback(), tempMonitorTask(), and testfunc2().

#### 6.23.2.2 dlc

```
uint8_t uv_CAN_msg::dlc
```

Data Length Code, representing how many bytes of data are present

Definition at line 275 of file uvfr_utils.h.

Referenced by __uvCANtxCritSection(), CANbusTxSvcDaemon(), HAL_CAN_RxFifo0MsgPendingCallback(), and tempMonitorTask().

#### 6.23.2.3 flags

```
uint8_t uv_CAN_msg::flags
```

Bitfield that contains some basic information about the message: -Bit 0: Is the message an extended ID message, or a standard ID message? 1 For extended. -Bits 1:2 Which CANbus is being used to send the message? 01 -> CAN1 10 -> CAN2 11-> CAN3 (doesnt exist yet). Will default to CAN

## 6.24 uv_init_struct Struct Reference

`#include <uvfr_utils.h>`

### Data Fields

- bool use_default_settings

### 6.24.1 Detailed Description

contains info relevant to initializing the vehicle

Definition at line 284 of file uvfr_utils.h.

### 6.24.2 Field Documentation

#### 6.24.2.1 use_default_settings

`bool uv_init_struct::use_default_settings`

Definition at line 285 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

## 6.25 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

### 6.25.1 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 329 of file uvfr_utils.h.

### 6.25.2 Field Documentation

#### 6.25.2.1 init_info_queue

```
QueueHandle_t uv_init_task_args::init_info_queue
```

Definition at line 331 of file uvfr_utils.h.

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

#### 6.25.2.2 meta_task_handle

```
TaskHandle_t uv_init_task_args::meta_task_handle
```

Definition at line 332 of file uvfr_utils.h.

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

#### 6.25.2.3 specific_args

```
void* uv_init_task_args::specific_args
```

Definition at line 330 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

## 6.26 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

### 6.26.1 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 355 of file uvfr_utils.h.

### 6.26.2 Field Documentation

#### 6.26.2.1 device

```
uv_ext_device_id uv_init_task_response::device
```

Definition at line 357 of file uvfr_utils.h.

Referenced by uvInit().

#### 6.26.2.2 errmsg

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

Definition at line 359 of file uvfr_utils.h.

Referenced by uvInit().

#### 6.26.2.3 nchar

```
uint8_t uv_init_task_response::nchar
```

Definition at line 358 of file uvfr_utils.h.

Referenced by uvInit().

#### 6.26.2.4 status

```
uv_status uv_init_task_response::status
```

Definition at line 356 of file uvfr_utils.h.

Referenced by uvInit().

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

- Core/Inc/uvfr_utils.h

## 6.27 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]

### 6.27.1 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 341 of file uvfr_utils.h.

### 6.27.2 Field Documentation

#### 6.27.2.1 e_code

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

Definition at line 345 of file uvfr_utils.h.

#### 6.27.2.2 init_params

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

Definition at line 342 of file uvfr_utils.h.

#### 6.27.2.3 peripheral_status

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

Definition at line 344 of file uvfr_utils.h.

#### 6.27.2.4 vehicle_settings

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

Definition at line 343 of file uvfr_utils.h.

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

- Core/Inc/uvfr_utils.h

## 6.28 uv_mutex_info Struct Reference

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

### Data Fields

- SemaphoreHandle_t handle

### 6.28.1 Detailed Description

Definition at line 239 of file uvfr_utils.h.

### 6.28.2 Field Documentation

#### 6.28.2.1 handle

```
SemaphoreHandle_t uv_mutex_info::handle
```

Definition at line 240 of file uvfr_utils.h.

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

- Core/Inc/uvfr_utils.h

## 6.29 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

### 6.29.1 Detailed Description

Settings that dictate state engine behavior.

Definition at line 171 of file uvfr_state_engine.h.

### 6.29.2 Field Documentation

#### 6.29.2.1 max_svc_task_period

```
TickType_t uv_os_settings::max_svc_task_period
```

Definition at line 174 of file uvfr_state_engine.h.

#### 6.29.2.2 max_task_period

```
TickType_t uv_os_settings::max_task_period
```

Definition at line 175 of file uvfr_state_engine.h.

#### 6.29.2.3 min_task_period

```
TickType_t uv_os_settings::min_task_period
```

Definition at line 176 of file uvfr_state_engine.h.

#### 6.29.2.4 svc_task_manager_period

```
TickType_t uv_os_settings::svc_task_manager_period
```

Definition at line 172 of file uvfr_state_engine.h.

#### 6.29.2.5 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

## 6.30 uv_scd_response Struct Reference

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

### Data Fields

- enum uv_scd_response_e response_val
- uv_task_id meta_id

### 6.30.1 Detailed Description

Definition at line 114 of file uvfr_state_engine.h.

### 6.30.2 Field Documentation

#### 6.30.2.1 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().

#### 6.30.2.2 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

## 6.31 uv_semaphore_info Struct Reference

`#include <uvfr_utils.h>`

### Data Fields

- SemaphoreHandle_t handle

### 6.31.1 Detailed Description

Definition at line 249 of file uvfr_utils.h.

### 6.31.2 Field Documentation

#### 6.31.2.1 handle

```
SemaphoreHandle_t uv_semaphore_info::handle
```

Definition at line 250 of file uvfr_utils.h.

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

- Core/Inc/uvfr_utils.h

## 6.32 uv_task_info Struct Reference

This struct is designed to hold neccessary 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

### 6.32.1 Detailed Description

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

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

Definition at line 209 of file uvfr_state_engine.h.

## 6.32.2 Field Documentation

### 6.32.2.1 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().

### 6.32.2.2 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 CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), daqMasterTask(), killSelf(), odometerTask(), StartDrivingLoop(), suspendSelf(), tempMonitorTask(), uvDeleteSVCTask(), uvDeleteTask(), and uvSuspendTask().

### 6.32.2.3 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.

### 6.32.2.4 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().

#### 6.32.2.5 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().

#### 6.32.2.6 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(), uv↩CreateTask(), uvStartStateMachine(), uvStartSVCTask(), and uvStartTask().

#### 6.32.2.7 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().

#### 6.32.2.8 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().

#### 6.32.2.9 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 Incrimentation - 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(), uvCreateServiceTask(), uvCreateTask(), uvScheduleTaskDeletion(), uv↩StartStateMachine(), uvStartSVCTask(), and uvTaskCrashHandler().

#### 6.32.2.10 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().

#### 6.32.2.11 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(), CANbusRxSvcDaemon(), killSelf(), proccessSCDMsg(), suspendSelf(), uvCreateServiceTask(), uvCreateTask(), uvDeleteSVCTask(), uvDeleteTask(), uvStartStateMachine(), uvStartSVCTask(), uvStartTask(), and uvSuspendTask().

#### 6.32.2.12 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().

#### 6.32.2.13 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(), compareTaskByName(), initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), uvCreateServiceTask(), uvCreateTask(), uvStartStateMachine(), uvStartSVCTask(), uvStartTask(), and uvSVCTaskManager().

#### 6.32.2.14 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().

#### 6.32.2.15 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().

#### 6.32.2.16 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.

#### 6.32.2.17 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().

#### 6.32.2.18 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.

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

- Core/Inc/uvfr_state_engine.h

## 6.33 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

### 6.33.1 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 301 of file uvfr_utils.h.

### 6.33.2 Field Documentation

#### 6.33.2.1 intended_recipient

```
uv_task_info* uv_task_msg_t::intended_recipient
```

Definition at line 304 of file uvfr_utils.h.

#### 6.33.2.2 message_size

```
size_t uv_task_msg_t::message_size
```

Definition at line 306 of file uvfr_utils.h.

#### 6.33.2.3 message_type

```
uint32_t uv_task_msg_t::message_type
```

Definition at line 302 of file uvfr_utils.h.

#### 6.33.2.4 msg_contents

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

Definition at line 307 of file uvfr_utils.h.

#### 6.33.2.5 sender

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

Definition at line 303 of file uvfr_utils.h.

#### 6.33.2.6 time_sent

```
TickType_t uv_task_msg_t::time_sent
```

Definition at line 305 of file uvfr_utils.h.

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

- Core/Inc/uvfr_utils.h

## 6.34 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

### 6.34.1 Detailed Description

Definition at line 32 of file uvfr_settings.h.

### 6.34.2 Field Documentation

#### 6.34.2.1 bms_settings

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

Definition at line 40 of file uvfr_settings.h.

Referenced by uvInit().

#### 6.34.2.2 daq_settings

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

Definition at line 42 of file uvfr_settings.h.

#### 6.34.2.3 driving_loop_settings

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

Definition at line 37 of file uvfr_settings.h.

#### 6.34.2.4 imd_settings

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

Definition at line 39 of file uvfr_settings.h.

Referenced by uvInit().

#### 6.34.2.5 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.

#### 6.34.2.6 mc_settings

```
struct motor_controller_settings* uv_vehicle_settings::mc_settings
```

Definition at line 35 of file uvfr_settings.h.

Referenced by uvInit().

#### 6.34.2.7 os_settings

```
struct uv_os_settings* uv_vehicle_settings::os_settings
```

Definition at line 34 of file uvfr_settings.h.

Referenced by setupDefaultSettings().

#### 6.34.2.8 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

## 6.35 veh_gen_info Struct Reference

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

### 6.35.1 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.*
- void CANbusRxSvcDaemon (void ∗args)

  *Background task that executes the CAN message callback functions.*
- void insertCANMessageHandler (uint32_t id, void ∗handlerfunc)

  *Function to insert an id and function into the lookup table of callback functions.*
- void nuke_hash_table ()

## 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.

/ /∗ USER CODE END Header

### 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 CANbusRxSvcDaemon()

```
void CANbusRxSvcDaemon (
            void * args )
```

Background task that executes the CAN message callback functions.

Basically just snoops through the hash table

Definition at line 618 of file can.c.

References callback_table_mutex, callFunctionFromCANid(), uv_task_info::cmd_data, killSelf(), Rx_msg_queue, suspendSelf(), uv_task_info::task_handle, UV_KILL_CMD, UV_OK, and UV_SUSPEND_CMD.

Referenced by uvSVCTaskManager().

#### 7.3.4.2 CANbusTxSvcDaemon()

```
void CANbusTxSvcDaemon (
            void * args )
```

Background task that handles any CAN messages that are being sent.

This task sits idle, until the time is right (it receives a notification from the uvSendCanMSG function) Once this condition has been met, it will actually call the `HAL_CAN_AddTxMessage` function. This is a very high priority task, meaning that it will pause whatever other code is going in order to run

Definition at line 551 of file can.c.

References uv_task_info::cmd_data, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, hcan2, killSelf(), uv_CAN_msg::msg_id, suspendSelf(), Tx_msg_queue, TxHeader, TxMailbox, UV_CAN_EXTENDED_ID, UV_KI↩ LL_CMD, and UV_SUSPEND_CMD.

Referenced by uvSVCTaskManager().

#### 7.3.4.3 HAL_CAN_RxFifo0MsgPendingCallback()

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

Definition at line 298 of file can.c.

References uv_CAN_msg::data, uv_CAN_msg::dlc, Error_Handler(), hcan2, uv_CAN_msg::msg_id, Rx_msg_↩ queue, and RxHeader.

#### 7.3.4.4 HAL_CAN_RxFifo1MsgPendingCallback()

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

Definition at line 338 of file can.c.

#### 7.3.4.5 MX_CAN2_Init()

```
void MX_CAN2_Init (
            void  )
```

Definition at line 150 of file can.c.

References Error_Handler(), hcan2, and TxHeader.

Referenced by main().

#### 7.3.4.6 nuke_hash_table()

```
void nuke_hash_table ( )
```

Function to free all malloced memory Index through the hash table and free all the malloced memory at each index

Definition at line 453 of file can.c.

References CAN_callback_table, CAN_Callback::next, and table_size.

### 7.3.5 Variable Documentation

#### 7.3.5.1 hcan2

```
CAN_HandleTypeDef hcan2
```

Definition at line 147 of file can.c.

Referenced by IMD_Request_Status(), main(), MC_Request_Data(), MotorControllerSpinTest(), PDU_disable_↩ brake_light(), PDU_disable_coolant_pump(), PDU_disable_cooling_fans(), PDU_disable_motor_controller(), PD↩ U_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(), 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 MC_Startup(), and MotorControllerSpinTest().

### 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(), 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 __uvCANtxCritSection(), CANbusTxSvcDaemon(), IMD_Request_Status(), main(), 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 __uvCANtxCritSection(), CANbusTxSvcDaemon(), IMD_Request_Status(), main(), MC_Request↩_Data(), MotorControllerSpinTest(), 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(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

## 7.5 Core/Inc/daq.h File Reference

```
#include "uvfr_utils.h"
#include "rb_tree.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 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 14 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 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 18 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

### 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 }

  *DL_PERIOD is meant to represent how often the driving loop executes, in ms.*
- 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 Typedef Documentation

#### 7.8.1.1 driving_loop_args

```
typedef struct driving_loop_args driving_loop_args
```

#### 7.8.1.2 drivingMode

```
typedef struct drivingMode drivingMode
```

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

#### 7.8.1.3 drivingModeParams

```
typedef union drivingModeParams drivingModeParams
```

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

#### 7.8.1.4 exp_torque_map_args

```
typedef struct exp_torque_map_args exp_torque_map_args
```

struct to hold parameters used in an exponential torque map

#### 7.8.1.5 linear_torque_map_args

```
typedef struct linear_torque_map_args linear_torque_map_args
```

#### 7.8.1.6 MC_POWER

```
typedef uint16_t MC_POWER
```

Definition at line 16 of file driving_loop.h.

#### 7.8.1.7 MC_RPM

```
typedef uint16_t MC_RPM
```

Definition at line 15 of file driving_loop.h.

#### 7.8.1.8 MC_Torque

```
typedef uint16_t MC_Torque
```

Definition at line 14 of file driving_loop.h.

#### 7.8.1.9 s_curve_torque_map_args

```
typedef struct s_curve_torque_map_args s_curve_torque_map_args
```

struct for s-curve parameters for torque

### 7.8.2 Enumeration Type Documentation

#### 7.8.2.1 DL_internal_state

```
enum DL_internal_state
```

Enumerator

| | |
|---|---|
| Plausible | |
| Implausible | |
| Erroneous | |

Definition at line 42 of file driving_loop.h.

#### 7.8.2.2 map_mode

```
enum map_mode
```

DL_PERIOD is meant to represent how often the driving loop executes, in ms.

This is a define since I would eventually like this to be configurable via a global variable, or possible be dynamic in the future.

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

enum meant to represent the different types of pedal map

This enum is meant to represent different functions that map the torque to speed.

Enumerator

| | |
|---|---|
| linear_speed_map | |
| s_curve_speed_map | |
| exp_speed_map | |
| linear_torque_map | |
| s_curve_torque_map | |
| exp_torque_map | |

Definition at line 33 of file driving_loop.h.

### 7.8.3 Function Documentation

#### 7.8.3.1 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_INIT, UV_LAUNCH_CONTROL, UV_OK, UV_READY, UV_SUSPENDED, and uvCreateTask().

Referenced by uvInitStateEngine().

#### 7.8.3.2 StartDrivingLoop()

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

Function implementing the ledTask thread.

Parameters

| | |
|---|---|
| *argument* | 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().

## 7.9 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 | |

Enumerator

| | |
|---|---|
| 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 | |
| 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(), I↩ MD_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(), 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)
- #define SERIAL_NUMBER_REGISTER 0x1A
- #define FIRMWARE_VERSION_REGISTER 0x1B

## 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 = 0x↩
  B1, nominal_magnitizing_current = 0xB2,
  motor_magnetising_inductance = 0xB3, rotor_resistance = 0xB4, minimum_magnetising_current = 0xB5,
  time_constant_rotor = 0xB6,
  leakage_inductance_ph_ph = 0xBB, stator_resistance_ph_ph = 0xBC, time_constant_stator = 0xBD }
- enum motor_controller_temperatures {
  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_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.

#### 7.14.1.2 FIRMWARE_VERSION_REGISTER

```
#define FIRMWARE_VERSION_REGISTER 0x1B
```

Definition at line 20 of file motor_controller.h.

#### 7.14.1.3 SERIAL_NUMBER_REGISTER

```
#define SERIAL_NUMBER_REGISTER 0x1A
```

Definition at line 19 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 30 of file motor_controller.h.

#### 7.14.3.2 motor_controller_io

```
enum motor_controller_io
```

Enumerator

| | |
|---|---|
| todo6969 | |

Definition at line 113 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 138 of file motor_controller.h.

#### 7.14.3.4 motor_controller_measurements

```
enum motor_controller_measurements
```

Enumerator

| | |
|---|---|
| DC_bus_voltage | |

Definition at line 68 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_magnitizing_current | |
| motor_magnetising_inductance | |
| rotor_resistance | |
| minimum_magnetising_current | |
| time_constant_rotor | |
| leakage_inductance_ph_ph | |
| stator_resistance_ph_ph | |
| time_constant_stator | |

Definition at line 34 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 117 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 133 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 23 of file motor_controller.h.

#### 7.14.3.9 motor_controller_startup

```
enum motor_controller_startup
```

Enumerator

| | |
|---|---|
| clear_errors | |
| firmware_version | |

Definition at line 144 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 | |

Enumerator

| | |
|---|---|
| temp_sensor_pt1 | |
| temp_sensor_pt2 | |
| temp_sensor_pt3 | |
| temp_sensor_pt4 | |

Definition at line 57 of file motor_controller.h.

### 7.14.4 Function Documentation

#### 7.14.4.1 MC_Startup()

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

Initializes the motor controller by performing the following steps:

1. Verifies the serial number from the motor controller.
2. Checks the firmware version to ensure compatibility.
3. Executes a motor spin test at low RPM to validate functionality.
4. Checks for errors and warnings from the motor controller.
5. Logs successful initialization if all checks pass.

Definition at line 739 of file motor_controller.c.

References firmware_version, FIRMWARE_VERSION_REGISTER, uv_init_task_args::init_info_queue, MC↩ _Expected_FW_Version, MC_Expected_Serial_Number, MC_Request_Data(), uv_init_task_args::meta_task↩ _handle, MOTOR_CONTROLLER, motor_controller_errors_warnings, MotorControllerErrorHandler(), Motor↩ ControllerSpinTest(), Parse_Bamocar_Response(), RxData, SERIAL_NUMBER_REGISTER, uv_init_task_args↩ ::specific_args, UV_OK, and WaitFor_CAN_Response().

Referenced by uvInit().

## 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

| | |
|---|---|
| enable_left_cooling_fan_msg | |
| disable_left_cooling_fan_msg | |
| enable_right_cooling_fan_msg | |
| disable_right_cooling_fan_msg | |
| enable_coolant_pump_msg | |
| disable_coolant_pump_msg | |

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 Tx↩ Mailbox.

#### 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_I↩ D_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_ISEMPTY(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_ISEMPTY

```
#define RB_ISEMPTY(
              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 /∗ CRYP register callback disabled ∗/
- #define USE_HAL_DAC_REGISTER_CALLBACKS 0U /∗ DAC register callback disabled ∗/
- #define USE_HAL_DCMI_REGISTER_CALLBACKS 0U /∗ DCMI register callback disabled ∗/
- #define USE_HAL_DFSDM_REGISTER_CALLBACKS 0U /∗ DFSDM register callback disabled ∗/
- #define USE_HAL_DMA2D_REGISTER_CALLBACKS 0U /∗ DMA2D register callback disabled ∗/
- #define USE_HAL_DSI_REGISTER_CALLBACKS 0U /∗ DSI register callback disabled ∗/
- #define USE_HAL_ETH_REGISTER_CALLBACKS 0U /∗ ETH register callback disabled ∗/
- #define USE_HAL_HASH_REGISTER_CALLBACKS 0U /∗ HASH register callback disabled ∗/
- #define USE_HAL_HCD_REGISTER_CALLBACKS 0U /∗ HCD register callback disabled ∗/
- #define USE_HAL_I2C_REGISTER_CALLBACKS 0U /∗ I2C register callback disabled ∗/
- #define USE_HAL_FMPI2C_REGISTER_CALLBACKS 0U /∗ FMPI2C register callback disabled ∗/
- #define USE_HAL_FMPSMBUS_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_FMPSMBUS_REGISTER_CALLBACKS

```
#define USE_HAL_FMPSMBUS_REGISTER_CALLBACKS 0U /* FMPSMBUS 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 70 of file temp_monitoring.c.

References uv_task_info::cmd_data, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, insertCANMessageHandler(), killSelf(), uv_CAN_msg::msg_id, suspendSelf(), uv_task_info::task_period, testfunc(), testfunc2(), TxData, TxHeader, UV_KILL_CMD, UV_SUSPEND_CMD, uvSendCanMSG(), 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 gnarlyness here, but not unacceptable levels. Pray this thing doesn't hang itself.*

- struct uv_task_info * uvCreateServiceTask ()

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

- struct uv_task_info * 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 *tsk_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&0x00F↩F0000)>>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&0x00F↩F0000)>>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

  *Data used by the uvfr_utils library to do what it needs to do :)*
- typedef struct uv_init_task_response uv_init_task_response

  *Struct representing the response of one of the initialization tasks.*

## Enumerations

- enum uv_status_t { UV_OK, UV_WARNING, UV_ERROR, UV_ABORTED }

  *This is meant to be a return type from functions that indicates what is actually going on.*
- enum data_type {
  UV_UINT8, UV_INT8, UV_UINT16, UV_INT16,
  UV_UINT32, UV_INT32, UV_FLOAT, UV_DOUBLE,
  UV_INT64, UV_UINT64, UV_STRING }

  *Represents the data type of some variable.*
- enum uv_driving_mode_t { normal, accel, econ, limp }
- enum uv_external_device { MOTOR_CONTROLLER = 0, BMS = 1, IMD = 2, PDU = 3 }

  *ID for external devices, which allows us to know what's good with them.*
- enum access_control_t {
  UV_NONE, UV_DUMB_FLAG, UV_MUTEX, UV_BINARY_SEMAPHORE,
  UV_SEMAPHORE }
- enum uv_msg_type_t {
  UV_TASK_START_COMMAND, UV_TASK_DELETE_COMMAND, UV_TASK_SUSPEND_COMMAND,
  UV_COMMAND_ACKNOWLEDGEMENT,
  UV_TASK_STATUS_REPORT, UV_ERROR_REPORT, UV_WAKEUP, UV_PARAM_REQUEST,
  UV_PARAM_READY, UV_RAW_DATA_TRANSFER, UV_SC_COMMAND, UV_INVALID_MSG,
  UV_ASSIGN_TASK }

  *Enum dictating the meaning of a generic message.*

## Functions

- void uvInit (void *arguments)

  *: Function that initializes all of the car's stuff.*
- void __uvInitPanic ()

  *Low Level Panic, that does not require the full UVFR utils functionality to be operational.*
- uv_status uvIsPTRValid (void *ptr)

  *function that checks to make sure a pointer points to a place it is allowed to point to*

## Variables

- uv_internal_params global_context

### 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 263 of file uvfr_utils.h.

#### 7.27.2.7 UV_CAN_DYNAMIC_MEM

```
#define UV_CAN_DYNAMIC_MEM 0b00001000
```

Definition at line 264 of file uvfr_utils.h.

#### 7.27.2.8 UV_CAN_EXTENDED_ID

```
#define UV_CAN_EXTENDED_ID 0x01
```

Definition at line 262 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 211 of file uvfr_utils.h.

#### 7.27.4.2 data_type

```
enum data_type
```

Represents the data type of some variable.

Enumerator

| | |
|---|---|
| UV_UINT8 | |
| UV_INT8 | |
| UV_UINT16 | |
| UV_INT16 | |
| UV_UINT32 | |
| UV_INT32 | |
| UV_FLOAT | |
| UV_DOUBLE | |
| UV_INT64 | |
| UV_UINT64 | |
| UV_STRING | |

Definition at line 177 of file uvfr_utils.h.

#### 7.27.4.3 uv_driving_mode_t

enum uv_driving_mode_t

Enumerator

| | |
|---|---|
| normal | |
| accel | |
| econ | |
| limp | |

Definition at line 194 of file uvfr_utils.h.

#### 7.27.4.4 uv_external_device

enum uv_external_device

ID for external devices, which allows us to know what's good with them.

Enumerator

| | |
|---|---|
| MOTOR_CONTROLLER | |
| BMS | |
| IMD | |
| PDU | |

Definition at line 204 of file uvfr_utils.h.

#### 7.27.4.5 uv_msg_type_t

enum uv_msg_type_t

Enum dictating the meaning of a generic message.

Enumerator

| | |
|---|---|
| UV_TASK_START_COMMAND | |
| UV_TASK_DELETE_COMMAND | |
| UV_TASK_SUSPEND_COMMAND | |
| UV_COMMAND_ACKNOWLEDGEMENT | |
| UV_TASK_STATUS_REPORT | |
| UV_ERROR_REPORT | |
| UV_WAKEUP | |

Enumerator

| | |
|---|---|
| UV_PARAM_REQUEST | |
| UV_PARAM_READY | |
| UV_RAW_DATA_TRANSFER | |
| UV_SC_COMMAND | |
| UV_INVALID_MSG | |
| UV_ASSIGN_TASK | |

Definition at line 222 of file uvfr_utils.h.

#### 7.27.4.6 uv_status_t

```
enum uv_status_t
```

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

Use this as a return value for functions you want to know the success of. In general, any function you write must return something, as well as account for any possible errors that may have occurred.

Enumerator

| | |
|---|---|
| UV_OK | |
| UV_WARNING | |
| UV_ERROR | |
| UV_ABORTED | |

Definition at line 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 271 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 implausibilitys, 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 existance of some devices. As a direct result, we can sorta just wait around and check that each task sends a message confirming that it has successfully executed. :) However, first we need to actually create a Queue for these tasks to use

```
  /
QueueHandle_t init_validation_queue = xQueueCreate(8,sizeof(uv_init_task_response));
if(init_validation_queue == NULL){
    __uvInitPanic();
}
```

The next big thing on our plate is checking the status of all external devices we need, and initializing them with appropriate parameters. These are split into tasks because it takes a bit of time, especially for devices that need to be configured via CANBus such as the motor controller. That is why it is split the way it is, to allow these to run somewhat concurrently

```
 */
BaseType_t retval;
//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, dumdum
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 39 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 doesnt 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 401 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() _uvStartCoolantPump_canBased()
- #define uvStopCoolantPump() _uvStopCoolantPump_canBased()
- #define uvHonkHorn() _uvHonkHorn_canBased()
- #define uvSilenceHorn() _uvSilenceHorn_canBased()
- #define uvSilenceHorn() _uvSilenceHorn_canBased()

## Functions

- void _uvOpenSDC_canBased ()
- void _uvCloseSDC_canBased ()
- void _uvStartCoolantPump_canBased ()
- void _uvStopCoolantPump_canBased ()
- void _uvHonkHorn_canBased ()
- void _uvSilenceHorn_canBased ()
- void uvSecureVehicle ()

  *Function to put vehicle into safe state.*

### 7.28.1 Macro Definition Documentation

#### 7.28.1.1 uvHonkHorn

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

Definition at line 95 of file uvfr_vehicle_commands.h.

#### 7.28.1.2 uvOpenSDC [1/2]

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

Definition at line 40 of file uvfr_vehicle_commands.h.

#### 7.28.1.3 uvOpenSDC [2/2]

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

Definition at line 40 of file uvfr_vehicle_commands.h.

#### 7.28.1.4 uvSilenceHorn [1/2]

```
#define uvSilenceHorn( ) _uvSilenceHorn_canBased()
```

Definition at line 110 of file uvfr_vehicle_commands.h.

#### 7.28.1.5 uvSilenceHorn [2/2]

```
#define uvSilenceHorn( ) _uvSilenceHorn_canBased()
```

Definition at line 110 of file uvfr_vehicle_commands.h.

#### 7.28.1.6 uvStartCoolantPump

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

Definition at line 72 of file uvfr_vehicle_commands.h.

#### 7.28.1.7 uvStartFans

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

Definition at line 51 of file uvfr_vehicle_commands.h.

#### 7.28.1.8 uvStopCoolantPump

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

Definition at line 83 of file uvfr_vehicle_commands.h.

#### 7.28.1.9 uvStopFans

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

Definition at line 61 of file uvfr_vehicle_commands.h.

### 7.28.2 Function Documentation

#### 7.28.2.1 _uvCloseSDC_canBased()

```
void _uvCloseSDC_canBased ( )
```

#### 7.28.2.2 _uvHonkHorn_canBased()

```
void _uvHonkHorn_canBased ( )
```

#### 7.28.2.3 _uvOpenSDC_canBased()

```
void _uvOpenSDC_canBased ( )
```

#### 7.28.2.4 _uvSilenceHorn_canBased()

```
void _uvSilenceHorn_canBased ( )
```

#### 7.28.2.5 _uvStartCoolantPump_canBased()

```
void _uvStartCoolantPump_canBased ( )
```

#### 7.28.2.6 _uvStopCoolantPump_canBased()

```
void _uvStopCoolantPump_canBased ( )
```

#### 7.28.2.7 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 11 of file uvfr_vehicle_commands.c.

Referenced by __uvPanic().

# 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 ----> AD↩ C1_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 ----—> AD↩ C1_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"
#include "task.h"
#include "stdlib.h"
#include "string.h"
```

### Data Structures

- struct CAN_Callback

### Macros

- #define HAL_CAN_ERROR_INVALID_CALLBACK (0x00400000U)
- #define table_size 128

### Typedefs

- typedef struct CAN_Callback CAN_Callback

## 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)
- unsigned int generateHash (uint32_t Incoming_CAN_id)
- static uv_status callFunctionFromCANid (uv_CAN_msg *msg)
- void insertCANMessageHandler (uint32_t id, void *handlerfunc)

  *Function to insert an id and function into the lookup table of callback functions.*
- void nuke_hash_table ()
- uv_status __uvCANtxCritSection (uv_CAN_msg *tx_msg)
- uv_status uvSendCanMSG (uv_CAN_msg *tx_msg)

  *Function to send CAN message.*
- void CANbusTxSvcDaemon (void *args)

  *Background task that handles any CAN messages that are being sent.*
- void CANbusRxSvcDaemon (void *args)

  *Background task that executes the CAN message callback functions.*

## Variables

- static QueueHandle_t Tx_msg_queue = NULL
- static QueueHandle_t Rx_msg_queue = NULL
- CAN_Callback CAN_callback_table [table_size] = {0}
- SemaphoreHandle_t callback_table_mutex = 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 46 of file can.c.

#### 7.31.2.2 table_size

```
#define table_size 128
```

Definition at line 52 of file can.c.

### 7.31.3 Typedef Documentation

#### 7.31.3.1 CAN_Callback

```
typedef struct CAN_Callback CAN_Callback
```

### 7.31.4 Function Documentation

#### 7.31.4.1 __uvCANtxCritSection()

```
uv_status __uvCANtxCritSection (
            uv_CAN_msg * tx_msg )
```

Definition at line 476 of file can.c.

References uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, hcan2, uv_CAN_msg::msg_id, TxHeader, TxMailbox, UV_CAN_EXTENDED_ID, UV_ERROR, and UV_OK.

Referenced by uvSendCanMSG().

#### 7.31.4.2 callFunctionFromCANid()

```
static uv_status callFunctionFromCANid (
            uv_CAN_msg * msg )  [inline], [static]
```

Function to take CAN id and find its corresponding function Given a CAN id, find it in the hash table and call the function if it exists If it doesn't exist, return 1 If it does exist but there are multiple can ids with the same hash follow the next pointer until the right CAN id is found Then call the function

Definition at line 368 of file can.c.

References CAN_callback_table, CAN_Callback::CAN_id, CAN_Callback::function, generateHash(), uv_CAN_msg::msg_id, CAN_Callback::next, UV_ERROR, UV_OK, and UV_WARNING.

Referenced by CANbusRxSvcDaemon().

#### 7.31.4.3 CANbusRxSvcDaemon()

```
void CANbusRxSvcDaemon (
            void * args )
```

Background task that executes the CAN message callback functions.

Basically just snoops through the hash table

Definition at line 618 of file can.c.

References callback_table_mutex, callFunctionFromCANid(), uv_task_info::cmd_data, killSelf(), Rx_msg_queue, suspendSelf(), uv_task_info::task_handle, UV_KILL_CMD, UV_OK, and UV_SUSPEND_CMD.

Referenced by uvSVCTaskManager().

#### 7.31.4.4 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 551 of file can.c.

References uv_task_info::cmd_data, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, hcan2, killSelf(), uv_CAN_msg::msg_id, suspendSelf(), Tx_msg_queue, TxHeader, TxMailbox, UV_CAN_EXTENDED_ID, UV_KILL_CMD, and UV_SUSPEND_CMD.

Referenced by uvSVCTaskManager().

#### 7.31.4.5 generateHash()

```
unsigned int generateHash (
            uint32_t Incoming_CAN_id )
```

HASH FUNCTION Take a can id and return a "random" hash id The hash id is in range from 0 to table_size The hash id is similar to an array index in its implementation

Definition at line 351 of file can.c.

References table_size.

Referenced by callFunctionFromCANid(), and insertCANMessageHandler().

#### 7.31.4.6 HAL_CAN_MspDeInit()

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

CAN2 GPIO Configuration PB12 ----—> CAN2_RX PB13 ----—> CAN2_TX

Definition at line 266 of file can.c.

#### 7.31.4.7 HAL_CAN_MspInit()

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

CAN2 GPIO Configuration PB12 ----—> CAN2_RX PB13 ----—> CAN2_TX

Definition at line 228 of file can.c.

#### 7.31.4.8 HAL_CAN_RxFifo0MsgPendingCallback()

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

Definition at line 298 of file can.c.

References uv_CAN_msg::data, uv_CAN_msg::dlc, Error_Handler(), hcan2, uv_CAN_msg::msg_id, Rx_msg_↩ queue, and RxHeader.

#### 7.31.4.9 HAL_CAN_RxFifo1MsgPendingCallback()

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

Definition at line 338 of file can.c.

#### 7.31.4.10 handleCANbusError()

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

Definition at line 71 of file can.c.

References HAL_CAN_ERROR_INVALID_CALLBACK.

Referenced by main().

#### 7.31.4.11 MX_CAN2_Init()

```
void MX_CAN2_Init (
            void  )
```

Definition at line 150 of file can.c.

References Error_Handler(), hcan2, and TxHeader.

Referenced by main().

#### 7.31.4.12 nuke_hash_table()

```
void nuke_hash_table ( )
```

Function to free all malloced memory Index through the hash table and free all the malloced memory at each index

Definition at line 453 of file can.c.

References CAN_callback_table, CAN_Callback::next, and table_size.

### 7.31.5 Variable Documentation

#### 7.31.5.1 callback_table_mutex

```
SemaphoreHandle_t callback_table_mutex = NULL
```

Definition at line 69 of file can.c.

Referenced by CANbusRxSvcDaemon(), and insertCANMessageHandler().

#### 7.31.5.2 CAN_callback_table

```
CAN_Callback CAN_callback_table[table_size] = {0}
```

Hash Table To Store CAN Messages Creates a hash table of size table_size and type CAN_Message Initialize all CAN messages in the hash table

Definition at line 67 of file can.c.

Referenced by callFunctionFromCANid(), insertCANMessageHandler(), and nuke_hash_table().

#### 7.31.5.3 hcan2

```
CAN_HandleTypeDef hcan2
```

Definition at line 147 of file can.c.

Referenced by __uvCANtxCritSection(), CAN2_RX0_IRQHandler(), CAN2_RX1_IRQHandler(), CAN2_TX_IRQHandler(), CANbusTxSvcDaemon(), HAL_CAN_RxFifo0MsgPendingCallback(), IMD_Request_Status(), main(), MC_Request_Data(), MotorControllerSpinTest(), 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(), Update_Batt_Temp(), Update_RPM(), and Update_State_Of_Charge().

#### 7.31.5.4 Rx_msg_queue

```
QueueHandle_t Rx_msg_queue = NULL  [static]
```

Definition at line 50 of file can.c.

Referenced by CANbusRxSvcDaemon(), and HAL_CAN_RxFifo0MsgPendingCallback().

#### 7.31.5.5 Tx_msg_queue

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

Definition at line 49 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 MC_Startup(), and MotorControllerSpinTest().

#### 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(), 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().

#### 7.32.1.4 TxHeader

```
CAN_TxHeaderTypeDef TxHeader
```

Definition at line 4 of file constants.c.

Referenced by __uvCANtxCritSection(), CANbusTxSvcDaemon(), IMD_Request_Status(), main(), 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 __uvCANtxCritSection(), CANbusTxSvcDaemon(), IMD_Request_Status(), main(), MC_Request_Data(), MotorControllerSpinTest(), 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(), 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.

References UV_ERROR.

#### 7.33.2.7 stopDaqSubTasks()

```
uv_status stopDaqSubTasks ( )
```

Definition at line 21 of file daq.c.

References UV_ERROR.

### 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_INIT, UV_LAUNCH_CONTROL, UV_OK, UV_READY, UV_SUSPENDED, and uvCreateTask().

Referenced by uvInitStateEngine().

### 7.36.2.2 StartDrivingLoop()

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

Function implementing the ledTask thread.

Parameters

| | |
|---|---|
| *argument* | 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_SUSPEN↩D_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 Start↩DrivingLoop().

### 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 Start↩DrivingLoop().

### 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 **ppxIdleTask↩ StackBuffer, uint32_t *pulIdleTaskStackSize)
- void vApplicationGetTimerTaskMemory (StaticTask_t **ppxTimerTaskTCBBuffer, StackType_t **ppxTimer↩ TaskStackBuffer, 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.

Parameters

| |
|---|
| *None* |

Return values

| |
|---|
| *None* |

Attention

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

Definition at line 160 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 209 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

#### 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(), I↩MD_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_APPS1
- uint16_t adc1_APPS2
- 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(), 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

| | |
|---|---|
| *htim* | : TIM handle |

Return values

| | |
|---|---|
| *None* | |

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

| | |
|---|---|
| *int* | |

Definition at line 97 of file main.c.

References adc1_APPS1, adc1_APPS2, 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(), System↩ Clock_Config(), TxData, TxHeader, TxMailbox, and Update_RPM().

#### 7.40.3.7 MX_FREERTOS_Init()

```
void MX_FREERTOS_Init (
             void  )
```

FreeRTOS initialization.

Parameters

| | |
|---|---|
| *None* | |

Return values

| | |
|---|---|
| *None* | |

Attention

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

Definition at line 160 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 Start↩ DrivingLoop().

#### 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 Start↩ DrivingLoop().

#### 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 "cmsis_os.h"
#include "pdu.h"
```

### Functions

- static uint32_t Parse_Bamocar_Response (uint8_t *data, uint8_t length)
- static void MotorControllerErrorHandler (uint8_t *data, uint8_t length)
- static uint16_t MotorControllerSpinTest (void)
- static bool WaitFor_CAN_Response (void)
- void MC_Request_Data (uint8_t RegID)
- 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

- QueueHandle_t canRxQueue
- QueueHandle_t canTxQueue
- const uint32_t MC_Expected_Serial_Number = 0x627E7A01
- const uint16_t MC_Expected_FW_Version = 0xDC01
- const uint32_t max_motor_speed = 3277
- motor_controller_settings mc_default_settings

## 7.41.1 Function Documentation

### 7.41.1.1 MC_Check_Error_Warning()

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

Definition at line 591 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_AC_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.

### 7.41.1.2 MC_Check_Firmware()

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

Definition at line 725 of file motor_controller.c.

### 7.41.1.3 MC_Check_Serial_Number()

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

Definition at line 721 of file motor_controller.c.

### 7.41.1.4 MC_Request_Data()

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

Sends a CAN request to the motor controller to retrieve a specific register value. Constructs a CAN message with the specified register ID and sends it via the CAN queue.

Parameters

| | |
|---|---|
| *RegID* | The ID of the register to request data from. |

Definition at line 230 of file motor_controller.c.

References motor_controllor_settings::can_id_tx, hcan2, mc_default_settings, and TxMailbox.

Referenced by MC_Startup(), and MotorControllerSpinTest().

#### 7.41.1.5 MC_Startup()

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

Initializes the motor controller by performing the following steps:

1. Verifies the serial number from the motor controller.
2. Checks the firmware version to ensure compatibility.
3. Executes a motor spin test at low RPM to validate functionality.
4. Checks for errors and warnings from the motor controller.
5. Logs successful initialization if all checks pass.

Definition at line 739 of file motor_controller.c.

References firmware_version, FIRMWARE_VERSION_REGISTER, uv_init_task_args::init_info_queue, MC↩_Expected_FW_Version, MC_Expected_Serial_Number, MC_Request_Data(), uv_init_task_args::meta_task↩_handle, MOTOR_CONTROLLER, motor_controller_errors_warnings, MotorControllerErrorHandler(), Motor↩ControllerSpinTest(), Parse_Bamocar_Response(), RxData, SERIAL_NUMBER_REGISTER, uv_init_task_args↩::specific_args, UV_OK, and WaitFor_CAN_Response().

Referenced by uvInit().

#### 7.41.1.6 MC_Validate()

```
void MC_Validate ( )
```

Definition at line 717 of file motor_controller.c.

#### 7.41.1.7 MotorControllerErrorHandler()

```
static void MotorControllerErrorHandler (
            uint8_t * data,
            uint8_t length )  [static]
```

Processes error and warning information from the motor controller.

1. Extracts error and warning flags from the CAN message payload.
2. Logs or triggers a panic if critical errors are detected.

Parameters

| | |
|---|---|
| *data* | Pointer to the CAN message payload. |
| *length* | Length of the data payload. |

Definition at line 119 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_AC_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_Startup().

#### 7.41.1.8 MotorControllerSpinTest()

```
static uint16_t MotorControllerSpinTest (
            void  )  [static]
```

Commands the motor to spin at a low RPM and validates the motor's response:

1. Sends a spin command via CAN.
2. Waits for the motor to reach the desired speed.
3. Checks the actual speed from the motor controller.
4. Stops the motor after validation.

   Returns

   0 if the test is successful, 1 for timeout, or 2 for insufficient speed.

Definition at line 62 of file motor_controller.c.

References motor_controllor_settings::can_id_tx, hcan2, mc_default_settings, MC_Request_Data(), N_actual, N_set, RxData, TxMailbox, and WaitFor_CAN_Response().

Referenced by MC_Startup().

#### 7.41.1.9 Parse_Bamocar_Response()

```
static uint32_t Parse_Bamocar_Response (
            uint8_t * data,
            uint8_t length )  [static]
```

Parses a 32-bit response value from a Bamocar CAN message. Combines the four bytes of the payload into a single 32-bit integer.

Parameters

| | |
|---|---|
| *data* | Pointer to the CAN message payload. |
| *length* | Length of the data payload (expected to be 4 bytes). |

Returns

Parsed 32-bit value.

Definition at line 107 of file motor_controller.c.

Referenced by MC_Startup().

#### 7.41.1.10 WaitFor_CAN_Response()

```
static bool WaitFor_CAN_Response (
            void  )  [static]
```

Waits for a CAN response from the motor controller. Uses an RTOS semaphore to synchronize and check if a response is received within the timeout period.

Returns

True if a response is received, otherwise false.

Definition at line 257 of file motor_controller.c.

Referenced by MC_Startup(), and MotorControllerSpinTest().

### 7.41.2 Variable Documentation

#### 7.41.2.1 canRxQueue

```
QueueHandle_t canRxQueue
```

#### 7.41.2.2 canTxQueue

```
QueueHandle_t canTxQueue
```

#### 7.41.2.3 max_motor_speed

```
const uint32_t max_motor_speed = 3277
```

Definition at line 21 of file motor_controller.c.

#### 7.41.2.4 mc_default_settings

```
motor_controller_settings mc_default_settings
```

**Initial value:**

```
= {
    .can_id_tx = 0x200,
    .can_id_rx = 0x201,
    .mc_CAN_timeout = 2,
    .proportional_gain = 10,
    .integral_time_constant = 400,
    .integral_memory_max = 0.6
}
```

Definition at line 26 of file motor_controller.c.

Referenced by MC_Request_Data(), and MotorControllerSpinTest().

#### 7.41.2.5 MC_Expected_FW_Version

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

Definition at line 20 of file motor_controller.c.

Referenced by MC_Startup().

#### 7.41.2.6 MC_Expected_Serial_Number

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

Definition at line 19 of file motor_controller.c.

Referenced by MC_Startup().

## 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 Tx↩ Mailbox.

#### 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 Tx↩ Mailbox.

#### 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 Tx↩ Mailbox.

#### 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 Tx↩ Mailbox.

#### 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_I↩ D_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

| | |
|---|---|
| *TickPriority* | Tick interrupt priority. |

Return values

| | |
|---|---|
| *HAL* | 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.

Parameters

| | |
|---|---|
| *None* | |

Return values

| | |
|---|---|
| *None* | |

Definition at line 121 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.

Parameters

| |
|---|
| *None* | |

Return values

| |
|---|
| *None* | |

Definition at line 109 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 147 of file can.c.

Referenced by __uvCANtxCritSection(), CAN2_RX0_IRQHandler(), CAN2_RX1_IRQHandler(), CAN2_TX_IRQ↩Handler(), CANbusTxSvcDaemon(), 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

| | |
|---|---|
| *incr* | 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 testfunc (uv_CAN_msg *msg)
- void testfunc2 (uv_CAN_msg *msg)
- 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 70 of file temp_monitoring.c.

References uv_task_info::cmd_data, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, insertCANMessageHandler(), killSelf(), uv_CAN_msg::msg_id, suspendSelf(), uv_task_info::task_period, testfunc(), testfunc2(), TxData, TxHeader, UV_KILL_CMD, UV_SUSPEND_CMD, uvSendCanMSG(), and uvTaskDelayUntil.

Referenced by initTempMonitor().

#### 7.53.1.3 testfunc()

```
void testfunc (
            uv_CAN_msg * msg )
```

Definition at line 42 of file temp_monitoring.c.

References changeVehicleState(), UV_DRIVING, UV_ERROR_STATE, UV_READY, and vehicle_state.

Referenced by tempMonitorTask().

#### 7.53.1.4 testfunc2()

```
void testfunc2 (
            uv_CAN_msg * msg )
```

Definition at line 52 of file temp_monitoring.c.

References uv_CAN_msg::data, uv_CAN_msg::flags, uv_CAN_msg::msg_id, and uvSendCanMSG().

Referenced by tempMonitorTask().

## 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 neccessary 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 neccessary 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.*
- int compareTaskByName (uv_task_info *t1, uv_task_info *t2)
- 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 gnarlyness 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 __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 uvSendTaskStatusReport (uv_task_info *t)
- void _stateChangeDaemon (void *args) PRIVILEGED_FUNCTION

  *This collects all the data changing from different tasks, and makes sure that everything works properly.*
- uv_status 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
- 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
- rbtree * task_name_tree
- 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.*
- void uvSysResetDaemon (void *args)
- 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 * __uvMallocCritSection (size_t memrequest)
  *Wrapper function for `malloc()` that makes it thread safe.*
- uv_status __uvFreeCritSection (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]
- TaskHandle_t reset_handle = NULL

### 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 328 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 379 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 271 of file uvfr_utils.c.

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

#### 7.57.2.4 __uvMallocCritSection()

```
void* __uvMallocCritSection (
            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 292 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 wanto to be using pvPortMalloc() rather than regular stdlib malloc() is to consolodate the heap between RTOS and non-RTOS functions.

Definition at line 353 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 259 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 implausibilitys, 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 existance of some devices. As a direct result, we can sorta just wait around and check that each task sends a message confirming that it has successfully executed. :) However, first we need to actually create a Queue for these tasks to use

```
  /
QueueHandle_t init_validation_queue = xQueueCreate(8,sizeof(uv_init_task_response));
if(init_validation_queue == NULL){
    __uvInitPanic();
}
```

The next big thing on our plate is checking the status of all external devices we need, and initializing them with appropriate parameters. These are split into tasks because it takes a bit of time, especially for devices that need to be configured via CANBus such as the motor controller. That is why it is split the way it is, to allow these to run somewhat concurrently

```
 */
BaseType_t retval;
//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, dumdum
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 39 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().

#### 7.57.2.8 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 doesnt 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 401 of file uvfr_utils.c.

References UV_ERROR, UV_OK, and UV_WARNING.

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

#### 7.57.2.9 uvSysResetDaemon()

```
void uvSysResetDaemon (
            void * args )
```

Definition at line 233 of file uvfr_utils.c.

Referenced by uvUtilsReset().

#### 7.57.2.10 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 250 of file uvfr_utils.c.

References reset_handle, UV_OK, and uvSysResetDaemon().

### 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 reset_handle

```
TaskHandle_t reset_handle = NULL
```

Definition at line 15 of file uvfr_utils.c.

Referenced by uvUtilsReset().

#### 7.57.3.3 TxData

```
uint8_t TxData[8]
```

Definition at line 7 of file constants.c.

## 7.58 Core/Src/uvfr_vehicle_commands.c File Reference

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

### Functions

- void uvSecureVehicle ()
  *Function to put vehicle into safe state.*

### 7.58.1 Function Documentation

#### 7.58.1.1 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 11 of file uvfr_vehicle_commands.c.

Referenced by __uvPanic().

# Index


_BV
    Utility Macros, 42
_BV_16
    Utility Macros, 42
_BV_32
    Utility Macros, 43
_BV_8
    Utility Macros, 43
_LONGEST_SC_TIME
    uvfr_state_engine.h, 206
_NUM_ERRORS_
    errorLUT.h, 129
_NUM_LOGGABLE_PARAMS
    daq.h, 119
_SC_DAEMON_PERIOD
    uvfr_state_engine.h, 206
_SRC_UVFR_DAQ
    daq.c, 237
_UV_DEFAULT_TASK_INSTANCES
    uvfr_state_engine.h, 206
_UV_DEFAULT_TASK_PERIOD
    uvfr_state_engine.h, 206
_UV_DEFAULT_TASK_STACK_SIZE
    uvfr_state_engine.h, 207
_UV_MIN_TASK_PERIOD
    uvfr_state_engine.h, 207
__attribute__
    syscalls.c, 292
__io_getchar
    syscalls.c, 292
__io_putchar
    syscalls.c, 292
__sbrk_heap_end
    sysmem.c, 298
__uvCANtxCritSection
    can.c, 230
__uvFreeCritSection
    uvfr_utils.c, 308
__uvFreeOS
    uvfr_utils.c, 308
__uvInitPanic
    uvfr_utils.c, 308
    uvfr_utils.h, 218
__uvMallocCritSection
    uvfr_utils.c, 309
__uvMallocOS
    uvfr_utils.c, 309
__uvPanic
    State Engine Internals, 30
_close
    syscalls.c, 293
_execve
    syscalls.c, 293
_exit
    syscalls.c, 293
_fork
    syscalls.c, 293
_fstat
    syscalls.c, 293
_getpid
    syscalls.c, 294
_isatty
    syscalls.c, 294
_kill
    syscalls.c, 294
_link
    syscalls.c, 294
_lseek
    syscalls.c, 294
_next_svc_task_id
    State Engine, 11
_next_task_id
    State Engine, 12
_open
    syscalls.c, 295
_sbrk
    sysmem.c, 297
_stat
    syscalls.c, 295
_stateChangeDaemon
    State Engine Internals, 30
_task_register
    State Engine, 12
_times
    syscalls.c, 295
_unlink
    syscalls.c, 295
_uvCloseSDC_canBased
    uvfr_vehicle_commands.h, 223
_uvHonkHorn_canBased
    uvfr_vehicle_commands.h, 223
_uvOpenSDC_canBased
    uvfr_vehicle_commands.h, 224
_uvSilenceHorn_canBased
    uvfr_vehicle_commands.h, 224
_uvStartCoolantPump_canBased
    uvfr_vehicle_commands.h, 224
_uvStopCoolantPump_canBased

uvfr_vehicle_commands.h, 224
_uvValidateSpecificTask
State Engine Internals, 32
_wait
syscalls.c, 295

a
s_curve_torque_map_args, 82
ABOVE_NORMAL
State Engine API, 24
absolute_max_acc_pwr
driving_loop_args, 66
absolute_max_accum_current
driving_loop_args, 67
absolute_max_motor_rpm
driving_loop_args, 67
absolute_max_motor_torque
driving_loop_args, 67
AC_current_offset_fault
motor_controller.h, 158
ACC_POWER
daq.h, 120
ACC_POWER_LIMIT
daq.h, 120
accel
uvfr_utils.h, 217
accelerate_ramp
motor_controller.h, 156
ACCELERATOR_PEDAL_RATIO
daq.h, 120
access_control_info, 59
bin_semaphore, 59
mutex, 59
semaphore, 59
uvfr_utils.h, 213
access_control_t
uvfr_utils.h, 216
access_control_type
uvfr_utils.h, 213
accum_regen_soc_threshold
driving_loop_args, 67
activation_time
p_status, 77
active_states
uv_task_info, 97
adc.c
hadc1, 227
hadc2, 227
HAL_ADC_MspDeInit, 225
HAL_ADC_MspInit, 225
hdma_adc1, 227
MX_ADC1_Init, 226
MX_ADC2_Init, 226
adc.h
ADC1_BUF_LEN, 108
ADC1_CHNL_CNT, 108
ADC1_MAX_VOLT, 108
ADC1_MIN_VOLT, 108
ADC1_SAMPLES, 108
ADC2_BUF_LEN, 109
ADC2_CHNL_CNT, 109
ADC2_MAX_VOLT, 109
ADC2_MIN_VOLT, 109
ADC2_SAMPLES, 109
hadc1, 110
hadc2, 111
MX_ADC1_Init, 110
MX_ADC2_Init, 110
adc1_APPS1
driving_loop.c, 243
main.c, 264
adc1_APPS2
driving_loop.c, 243
main.c, 264
adc1_BPS1
driving_loop.c, 243
main.c, 265
adc1_BPS2
driving_loop.c, 243
main.c, 265
ADC1_BUF_LEN
adc.h, 108
ADC1_CHNL_CNT
adc.h, 108
ADC1_MAX_VOLT
adc.h, 108
ADC1_MIN_VOLT
adc.h, 108
ADC1_SAMPLES
adc.h, 108
ADC2_BUF_LEN
adc.h, 109
ADC2_CHNL_CNT
adc.h, 109
adc2_CoolantFlow
main.c, 265
adc2_CoolantTemp
main.c, 265
ADC2_MAX_VOLT
adc.h, 109
ADC2_MIN_VOLT
adc.h, 109
ADC2_SAMPLES
adc.h, 109
adc_buf1
main.c, 265
adc_buf2
main.c, 266
ADC_measurement_problem
motor_controller.h, 158
ADC_sequencer_problem
motor_controller.h, 158
addTaskToTaskRegister
State Engine Internals, 32
AHBPrescTable
STM32F4xx_System_Private_Variables, 56
air_temperature

motor_controller.h, 158
amogus
oled.h, 161
APBPrescTable
STM32F4xx_System_Private_Variables, 56
APPS1_ADC_VAL
daq.h, 120
APPS2_ADC_VAL
daq.h, 120
apps_bottom
driving_loop_args, 67
apps_implausibility_recovery_threshold
driving_loop_args, 68
apps_plausibility_check_threshold
driving_loop_args, 68
apps_top
driving_loop_args, 68
assert_param
stm32f4xx_hal_conf.h, 176
auxiliary_voltage_min_limit
motor_controller.h, 158
AVG_CELL_TEMP
daq.h, 120

b
s_curve_torque_map_args, 82
battery_voltage
imd.h, 144
BELOW_NORMAL
State Engine API, 24
bin_semaphore
access_control_info, 59
BLACK
rb_tree.h, 167
bleed_resistor_overload
motor_controller.h, 158
bleeder_resistor_warning
motor_controller.h, 158
Blue_LED_GPIO_Port
main.h, 150
Blue_LED_Pin
main.h, 150
BMS
uvfr_utils.h, 217
bms.c
BMS_Init, 228
bms.h
BMS_Init, 112
bms_settings_t, 112
DEFAULT_BMS_CAN_TIMEOUT, 111
BMS_CURRENT
daq.h, 120
BMS_ERRORS
daq.h, 120
BMS_Init
bms.c, 228
bms.h, 112
bms_settings
uv_vehicle_settings, 104
bms_settings_t, 60
bms.h, 112
mc_CAN_timeout, 60
BMS_VOLTAGE
daq.h, 120
bool
uvfr_utils.h, 213
BPS1_ADC_VAL
daq.h, 120
BPS2_ADC_VAL
daq.h, 120
bps_implausibility_recovery_threshold
driving_loop_args, 68
bps_plausibility_check_threshold
driving_loop_args, 68
BRAKE_PRESSURE_PA
daq.h, 120
BusFault_Handler
stm32f4xx_it.c, 287
stm32f4xx_it.h, 195

c
s_curve_torque_map_args, 83
callback_table_mutex
can.c, 233
callFunctionFromCANid
can.c, 230
can.c
__uvCANtxCritSection, 230
callback_table_mutex, 233
callFunctionFromCANid, 230
CAN_Callback, 230
CAN_callback_table, 234
CANbusRxSvcDaemon, 231
CANbusTxSvcDaemon, 231
generateHash, 231
HAL_CAN_ERROR_INVALID_CALLBACK, 229
HAL_CAN_MspDeInit, 232
HAL_CAN_MspInit, 232
HAL_CAN_RxFifo0MsgPendingCallback, 232
HAL_CAN_RxFifo1MsgPendingCallback, 232
handleCANbusError, 233
hcan2, 234
MX_CAN2_Init, 233
nuke_hash_table, 233
Rx_msg_queue, 234
table_size, 230
Tx_msg_queue, 234
can.h
CAN_RX_DAEMON_NAME, 113
CAN_TX_DAEMON_NAME, 113
CANbusRxSvcDaemon, 114
CANbusTxSvcDaemon, 114
HAL_CAN_RxFifo0MsgPendingCallback, 115
HAL_CAN_RxFifo1MsgPendingCallback, 115
hcan2, 116
MX_CAN2_Init, 115
nuke_hash_table, 115
uv_CAN_msg, 114

uv_status, 114
CAN2_RX0_IRQHandler
stm32f4xx_it.c, 287
stm32f4xx_it.h, 195
CAN2_RX1_IRQHandler
stm32f4xx_it.c, 288
stm32f4xx_it.h, 195
CAN2_TX_IRQHandler
stm32f4xx_it.c, 288
stm32f4xx_it.h, 195
CAN_Callback, 60
can.c, 230
CAN_id, 61
function, 61
next, 61
CAN_callback_table
can.c, 234
CAN_id
CAN_Callback, 61
can_id
daq_datapoint, 63
can_id_rx
motor_controllor_settings, 76
can_id_tx
motor_controllor_settings, 76
CAN_IDs
constants.h, 116
CAN_RX_DAEMON_NAME
can.h, 113
CAN_timeout_error
motor_controller.h, 158
CAN_TX_DAEMON_NAME
can.h, 113
CANbusRxSvcDaemon
can.c, 231
can.h, 114
CANbusTxSvcDaemon
can.c, 231
can.h, 114
canRxQueue
motor_controller.c, 270
canTxQueue
motor_controller.c, 270
changeVehicleState
State Engine API, 26
check_ecode_ID
motor_controller.h, 158
checkBlackHeight
rb_tree.c, 276
checkOrder
rb_tree.c, 277
clear_errors
motor_controller.h, 157
cmd_data
uv_task_info, 97
CMSIS, 50
color
rbnode, 78
compare
rbtree, 80
compareTaskByName
State Engine, 11
configASSERT
FreeRTOSConfig.h, 130
configCHECK_FOR_STACK_OVERFLOW
FreeRTOSConfig.h, 130, 131
configCPU_CLOCK_HZ
FreeRTOSConfig.h, 131
configENABLE_BACKWARD_COMPATIBILITY
FreeRTOSConfig.h, 131
configENABLE_FPU
FreeRTOSConfig.h, 131
configENABLE_MPU
FreeRTOSConfig.h, 131
configKERNEL_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 131
configLIBRARY_LOWEST_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 132
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 132
configMAX_CO_ROUTINE_PRIORITIES
FreeRTOSConfig.h, 132
configMAX_PRIORITIES
FreeRTOSConfig.h, 132
configMAX_SYSCALL_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 132
configMAX_TASK_NAME_LEN
FreeRTOSConfig.h, 132
configMESSAGE_BUFFER_LENGTH_TYPE
FreeRTOSConfig.h, 133
configMINIMAL_STACK_SIZE
FreeRTOSConfig.h, 133
configPRIO_BITS
FreeRTOSConfig.h, 133
configQUEUE_REGISTRY_SIZE
FreeRTOSConfig.h, 133
configRECORD_STACK_HIGH_ADDRESS
FreeRTOSConfig.h, 133
configSUPPORT_DYNAMIC_ALLOCATION
FreeRTOSConfig.h, 133
configSUPPORT_STATIC_ALLOCATION
FreeRTOSConfig.h, 134
configTICK_RATE_HZ
FreeRTOSConfig.h, 134
configTIMER_QUEUE_LENGTH
FreeRTOSConfig.h, 134
configTIMER_TASK_PRIORITY
FreeRTOSConfig.h, 134
configTIMER_TASK_STACK_DEPTH
FreeRTOSConfig.h, 134
configTOTAL_HEAP_SIZE
FreeRTOSConfig.h, 134
configUSE_16_BIT_TICKS
FreeRTOSConfig.h, 135
configUSE_APPLICATION_TASK_TAG
FreeRTOSConfig.h, 135

configUSE_CO_ROUTINES
FreeRTOSConfig.h, 135
configUSE_COUNTING_SEMAPHORES
FreeRTOSConfig.h, 135
configUSE_IDLE_HOOK
FreeRTOSConfig.h, 135
configUSE_MALLOC_FAILED_HOOK
FreeRTOSConfig.h, 135, 136
configUSE_MUTEXES
FreeRTOSConfig.h, 136
configUSE_PORT_OPTIMISED_TASK_SELECTION
FreeRTOSConfig.h, 136
configUSE_PREEMPTION
FreeRTOSConfig.h, 136
configUSE_TICK_HOOK
FreeRTOSConfig.h, 136
configUSE_TIMERS
FreeRTOSConfig.h, 136
constants.c
RxData, 235
RxHeader, 235
TxData, 235
TxHeader, 236
TxMailbox, 236
constants.h
CAN_IDs, 116
IMD_CAN_ID_Rx, 117
IMD_CAN_ID_Tx, 117
MC_CAN_ID_Rx, 117
MC_CAN_ID_Tx, 117
PDU_CAN_ID_Tx, 117
RxData, 117
RxHeader, 117
TxData, 117
TxHeader, 117
TxMailbox, 118
control_map_fn
drivingMode, 72
Core/Inc/adc.h, 107
Core/Inc/bms.h, 111
Core/Inc/can.h, 112
Core/Inc/constants.h, 116
Core/Inc/daq.h, 118
Core/Inc/dash.h, 122
Core/Inc/dma.h, 123
Core/Inc/driving_loop.h, 124
Core/Inc/errorLUT.h, 129
Core/Inc/FreeRTOSConfig.h, 129
Core/Inc/gpio.h, 141
Core/Inc/imd.h, 141
Core/Inc/main.h, 149
Core/Inc/motor_controller.h, 152
Core/Inc/odometer.h, 159
Core/Inc/oled.h, 160
Core/Inc/pdu.h, 162
Core/Inc/rb_tree.h, 166
Core/Inc/spi.h, 172
Core/Inc/stm32f4xx_hal_conf.h, 173
Core/Inc/stm32f4xx_it.h, 194
Core/Inc/temp_monitoring.h, 198
Core/Inc/tim.h, 199
Core/Inc/uvfr_global_config.h, 200
Core/Inc/uvfr_settings.h, 201
Core/Inc/uvfr_state_engine.h, 203
Core/Inc/uvfr_utils.h, 209
Core/Inc/uvfr_vehicle_commands.h, 221
Core/Src/adc.c, 225
Core/Src/bms.c, 227
Core/Src/can.c, 228
Core/Src/constants.c, 235
Core/Src/daq.c, 236
Core/Src/dash.c, 239
Core/Src/dma.c, 240
Core/Src/driving_loop.c, 241
Core/Src/freertos.c, 244
Core/Src/gpio.c, 248
Core/Src/imd.c, 249
Core/Src/main.c, 260
Core/Src/motor_controller.c, 266
Core/Src/odometer.c, 271
Core/Src/oled.c, 273
Core/Src/pdu.c, 273
Core/Src/rb_tree.c, 276
Core/Src/spi.c, 281
Core/Src/stm32f4xx_hal_msp.c, 283
Core/Src/stm32f4xx_hal_timebase_tim.c, 284
Core/Src/stm32f4xx_it.c, 286
Core/Src/syscalls.c, 291
Core/Src/sysmem.c, 296
Core/Src/system_stm32f4xx.c, 298
Core/Src/temp_monitoring.c, 299
Core/Src/tim.c, 301
Core/Src/uvfr_settings.c, 302
Core/Src/uvfr_state_engine.c, 305
Core/Src/uvfr_utils.c, 307
Core/Src/uvfr_vehicle_commands.c, 313
count
rbtree, 80
critical_AC_current
motor_controller.h, 158
current_derate_temperature
motor_controller.h, 158
CURRENT_DRIVING_MODE
daq.h, 120
current_feed_forward
motor_controller.h, 156
current_vehicle_settings
uvfr_settings.c, 304
uvfr_settings.h, 203

daq.c
_SRC_UVFR_DAQ, 237
daqMasterTask, 237
daqSubTask, 237
deleteDaqSubTask, 238
deleteParamList, 238
initDaqTask, 238

param_LUT, 239
startDaqSubTasks, 238
stopDaqSubTasks, 238
daq.h
_NUM_LOGGABLE_PARAMS, 119
ACC_POWER, 120
ACC_POWER_LIMIT, 120
ACCELERATOR_PEDAL_RATIO, 120
APPS1_ADC_VAL, 120
APPS2_ADC_VAL, 120
AVG_CELL_TEMP, 120
BMS_CURRENT, 120
BMS_ERRORS, 120
BMS_VOLTAGE, 120
BPS1_ADC_VAL, 120
BPS2_ADC_VAL, 120
BRAKE_PRESSURE_PA, 120
CURRENT_DRIVING_MODE, 120
daq_child_task, 119
daq_datapoint, 119
daq_loop_args, 119
daq_param_list_node, 120
daqMasterTask, 121
initDaqTask, 121
loggable_params, 120
MAX_CELL_TEMP, 120
MAX_LOGGABLE_PARAMS, 120
MC_CURRENT, 120
MC_ERRORS, 120
MC_TEMP, 120
MC_VOLTAGE, 120
MIN_CELL_TEMP, 120
MOTOR_CURRENT, 120
MOTOR_RPM, 120
MOTOR_TEMP, 120
param_LUT, 121
POWER_DERATE_FACTOR, 120
daq_child_task, 61
daq.h, 119
meta_task_handle, 62
param_list, 62
period, 62
treenode, 62
daq_datapoint, 63
can_id, 63
daq.h, 119
period, 63
type, 63
daq_loop_args, 64
daq.h, 119
datapoints, 64
minimum_daq_period, 64
padding, 64
padding2, 64
throttle_daq_to_preserve_performance, 65
daq_param_list_node, 65
daq.h, 120
next, 65
param_idx, 65
daq_settings
uv_vehicle_settings, 104
daqMasterTask
daq.c, 237
daq.h, 121
daqSubTask
daq.c, 237
dash.c
Update_Batt_Temp, 239
Update_RPM, 239
Update_State_Of_Charge, 240
dash.h
Dash_Battery_Temperature, 122
dash_can_ids, 122
Dash_Motor_Temperature, 122
Dash_RPM, 122
Dash_State_of_Charge, 122
Update_Batt_Temp, 122
Update_RPM, 122
Update_State_Of_Charge, 123
Dash_Battery_Temperature
dash.h, 122
dash_can_ids
dash.h, 122
Dash_Motor_Temperature
dash.h, 122
Dash_RPM
dash.h, 122
Dash_State_of_Charge
dash.h, 122
data
rbnode, 78
uv_CAN_msg, 86
DATA_CACHE_ENABLE
stm32f4xx_hal_conf.h, 176
data_type
uvfr_utils.h, 216
datapoints
daq_loop_args, 64
DC_bus_voltage
motor_controller.h, 155
DEBUG_CAN_IN_MAIN
main.c, 261
DebugMon_Handler
stm32f4xx_it.c, 288
stm32f4xx_it.h, 196
DEFAULT_BMS_CAN_TIMEOUT
bms.h, 111
DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT
motor_controller.h, 154
default_os_settings
State Engine, 12
defaultTaskHandle
freertos.c, 247
deleteDaqSubTask
daq.c, 238
deleteParamList

daq.c, 238
deleteRepair
rb_tree.c, 277
deletion_delay
uv_task_info, 97
deletion_states
uv_task_info, 97
deserializeBigE16
Utility Macros, 43
deserializeBigE32
Utility Macros, 43
deserializeSmallE16
Utility Macros, 43
deserializeSmallE32
Utility Macros, 44
destroy
rbtree, 80
destroyAllNodes
rb_tree.c, 277
device
uv_init_task_response, 90
disable_brake_light_msg
pdu.h, 163
disable_coolant_pump_msg
pdu.h, 163
disable_left_cooling_fan_msg
pdu.h, 163
disable_motor_controller_msg
pdu.h, 163
disable_right_cooling_fan_msg
pdu.h, 163
disable_shutdown_circuit_msg
pdu.h, 163
disable_speaker_msg
pdu.h, 163
dismantling_ramp
motor_controller.h, 156
DL_internal_state
driving_loop.h, 126
dlc
uv_CAN_msg, 87
dm_name
drivingMode, 72
dma.c
MX_DMA_Init, 241
dma.h
MX_DMA_Init, 124
DMA2_Stream0_IRQHandler
stm32f4xx_it.c, 288
stm32f4xx_it.h, 196
dmodes
driving_loop_args, 69
DP83848_PHY_ADDRESS
stm32f4xx_hal_conf.h, 176
driving_loop.c
adc1_APPS1, 243
adc1_APPS2, 243
adc1_BPS1, 243
adc1_BPS2, 243
initDrivingLoop, 242
StartDrivingLoop, 242
driving_loop.h
DL_internal_state, 126
driving_loop_args, 125
drivingMode, 125
drivingModeParams, 125
Erroneous, 127
exp_speed_map, 127
exp_torque_map, 127
exp_torque_map_args, 125
Implausible, 127
initDrivingLoop, 127
linear_speed_map, 127
linear_torque_map, 127
linear_torque_map_args, 125
map_mode, 127
MC_POWER, 126
MC_RPM, 126
MC_Torque, 126
Plausible, 127
s_curve_speed_map, 127
s_curve_torque_map, 127
s_curve_torque_map_args, 126
StartDrivingLoop, 128
driving_loop_args, 66
absolute_max_acc_pwr, 66
absolute_max_accum_current, 67
absolute_max_motor_rpm, 67
absolute_max_motor_torque, 67
accum_regen_soc_threshold, 67
apps_bottom, 67
apps_implausibility_recovery_threshold, 68
apps_plausibility_check_threshold, 68
apps_top, 68
bps_implausibility_recovery_threshold, 68
bps_plausibility_check_threshold, 68
dmodes, 69
driving_loop.h, 125
max_accum_current_5s, 69
max_apps_offset, 69
max_apps_value, 69
max_BPS_value, 69
min_apps_offset, 70
min_apps_value, 70
min_BPS_value, 70
num_driving_modes, 70
period, 70
regen_rpm_cutoff, 71
driving_loop_settings
uv_vehicle_settings, 104
drivingLoopArgs, 71
drivingMode, 71
control_map_fn, 72
dm_name, 72
driving_loop.h, 125
flags, 72

map_fn_params, 72
max_acc_pwr, 73
max_current, 73
max_motor_torque, 73
drivingModeParams, 73
driving_loop.h, 125

e_code
uv_internal_params, 91
ecode_timeout_error
motor_controller.h, 158
econ
uvfr_utils.h, 217
ECUMASTER_PMU
uvfr_global_config.h, 200
enable_brake_light_msg
pdu.h, 163
enable_coolant_pump_msg
pdu.h, 163
ENABLE_FLASH_SETTINGS
uvfr_settings.h, 202
enable_left_cooling_fan_msg
pdu.h, 163
enable_motor_controller_msg
pdu.h, 163
enable_right_cooling_fan_msg
pdu.h, 163
enable_shutdown_circuit_msg
pdu.h, 163
enable_speaker_msg
pdu.h, 163
endianSwap
Utility Macros, 44
endianSwap16
Utility Macros, 44
endianSwap32
Utility Macros, 44
endianSwap8
Utility Macros, 44
environ
syscalls.c, 296
eprom_read_error
motor_controller.h, 158
Err_CH
imd.h, 143
Err_clock
imd.h, 143
Err_temp
imd.h, 143
Err_Vexi
imd.h, 143
Err_Vpwr
imd.h, 143
Err_Vx1
imd.h, 143
Err_Vx2
imd.h, 143
Err_VxR
imd.h, 143
Err_Watchdog
imd.h, 143
errmsg
uv_init_task_response, 90
Erroneous
driving_loop.h, 127
Error_flags
imd.h, 144
Error_Handler
main.c, 261
main.h, 152
errorLUT.h
_NUM_ERRORS_, 129
ETH_RX_BUF_SIZE
stm32f4xx_hal_conf.h, 177
ETH_RXBUFNB
stm32f4xx_hal_conf.h, 177
ETH_TX_BUF_SIZE
stm32f4xx_hal_conf.h, 177
ETH_TXBUFNB
stm32f4xx_hal_conf.h, 177
Exc_off
imd.h, 144
exp_speed_map
driving_loop.h, 127
exp_torque_map
driving_loop.h, 127
exp_torque_map_args, 74
driving_loop.h, 125
gamma, 74
offset, 74
EXTERNAL_CLOCK_VALUE
stm32f4xx_hal_conf.h, 177
EXTI0_IRQHandler
stm32f4xx_it.c, 289
stm32f4xx_it.h, 196

false
Utility Macros, 45
feedback_signal_error
motor_controller.h, 158
feedback_signal_problem
motor_controller.h, 158
firmware_version
motor_controller.h, 157
FIRMWARE_VERSION_REGISTER
motor_controller.h, 154
flags
drivingMode, 72
uv_CAN_msg, 87
freertos.c
defaultTaskHandle, 247
init_settings, 247
init_task_handle, 247
MX_FREERTOS_Init, 244
StartDefaultTask, 245
vApplicationGetIdleTaskMemory, 245
vApplicationGetTimerTaskMemory, 245
vApplicationIdleHook, 246

vApplicationMallocFailedHook, 246
vApplicationStackOverflowHook, 246
vApplicationTickHook, 246
xIdleStack, 247
xIdleTaskTCBBuffer, 247
xTimerStack, 248
xTimerTaskTCBBuffer, 248
FreeRTOSConfig.h
configASSERT, 130
configCHECK_FOR_STACK_OVERFLOW, 130, 131
configCPU_CLOCK_HZ, 131
configENABLE_BACKWARD_COMPATIBILITY, 131
configENABLE_FPU, 131
configENABLE_MPU, 131
configKERNEL_INTERRUPT_PRIORITY, 131
configLIBRARY_LOWEST_INTERRUPT_PRIORITY, 132
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY, 132
configMAX_CO_ROUTINE_PRIORITIES, 132
configMAX_PRIORITIES, 132
configMAX_SYSCALL_INTERRUPT_PRIORITY, 132
configMAX_TASK_NAME_LEN, 132
configMESSAGE_BUFFER_LENGTH_TYPE, 133
configMINIMAL_STACK_SIZE, 133
configPRIO_BITS, 133
configQUEUE_REGISTRY_SIZE, 133
configRECORD_STACK_HIGH_ADDRESS, 133
configSUPPORT_DYNAMIC_ALLOCATION, 133
configSUPPORT_STATIC_ALLOCATION, 134
configTICK_RATE_HZ, 134
configTIMER_QUEUE_LENGTH, 134
configTIMER_TASK_PRIORITY, 134
configTIMER_TASK_STACK_DEPTH, 134
configTOTAL_HEAP_SIZE, 134
configUSE_16_BIT_TICKS, 135
configUSE_APPLICATION_TASK_TAG, 135
configUSE_CO_ROUTINES, 135
configUSE_COUNTING_SEMAPHORES, 135
configUSE_IDLE_HOOK, 135
configUSE_MALLOC_FAILED_HOOK, 135, 136
configUSE_MUTEXES, 136
configUSE_PORT_OPTIMISED_TASK_SELECTION, 136
configUSE_PREEMPTION, 136
configUSE_TICK_HOOK, 136
configUSE_TIMERS, 136
INCLUDE_eTaskGetState, 137
INCLUDE_pcTaskGetTaskName, 137
INCLUDE_uxTaskGetStackHighWaterMark, 137
INCLUDE_uxTaskGetStackHighWaterMark2, 137
INCLUDE_uxTaskPriorityGet, 137
INCLUDE_vTaskCleanUpResources, 137
INCLUDE_vTaskDelay, 138
INCLUDE_vTaskDelayUntil, 138
INCLUDE_vTaskDelete, 138
INCLUDE_vTaskPrioritySet, 138
INCLUDE_vTaskSuspend, 138
INCLUDE_xEventGroupSetBitFromISR, 138
INCLUDE_xQueueGetMutexHolder, 139
INCLUDE_xSemaphoreGetMutexHolder, 139
INCLUDE_xTaskAbortDelay, 139
INCLUDE_xTaskDelayUntil, 139
INCLUDE_xTaskGetCurrentTaskHandle, 139
INCLUDE_xTaskGetHandle, 139
INCLUDE_xTaskGetSchedulerState, 140
INCLUDE_xTimerPendFunctionCall, 140
vPortSVCHandler, 140
xPortPendSVHandler, 140
xPortSysTickHandler, 140
function
CAN_Callback, 61

gamma
exp_torque_map_args, 74
generateHash
can.c, 231
getSVCTaskID
uvfr_state_engine.h, 208
global_context
uvfr_utils.h, 221
gpio.c
MX_GPIO_Init, 249
gpio.h
MX_GPIO_Init, 141

hadc1
adc.c, 227
adc.h, 110
hadc2
adc.c, 227
adc.h, 111
HAL_ADC_ConvCpltCallback
main.c, 262
HAL_ADC_LevelOutOfWindowCallback
main.c, 262
HAL_ADC_MODULE_ENABLED
stm32f4xx_hal_conf.h, 177
HAL_ADC_MspDeInit
adc.c, 225
HAL_ADC_MspInit
adc.c, 225
HAL_CAN_ERROR_INVALID_CALLBACK
can.c, 229
HAL_CAN_MODULE_ENABLED
stm32f4xx_hal_conf.h, 178
HAL_CAN_MspDeInit
can.c, 232
HAL_CAN_MspInit
can.c, 232
HAL_CAN_RxFifo0MsgPendingCallback
can.c, 232
can.h, 115
HAL_CAN_RxFifo1MsgPendingCallback

can.c, 232
can.h, 115
HAL_CORTEX_MODULE_ENABLED
stm32f4xx_hal_conf.h, 178
HAL_DMA_MODULE_ENABLED
stm32f4xx_hal_conf.h, 178
HAL_EXTI_MODULE_ENABLED
stm32f4xx_hal_conf.h, 178
HAL_FLASH_MODULE_ENABLED
stm32f4xx_hal_conf.h, 178
HAL_GPIO_EXTI_Callback
main.c, 262
HAL_GPIO_MODULE_ENABLED
stm32f4xx_hal_conf.h, 178
HAL_InitTick
stm32f4xx_hal_timebase_tim.c, 284
HAL_MODULE_ENABLED
stm32f4xx_hal_conf.h, 179
HAL_MspInit
stm32f4xx_hal_msp.c, 283
HAL_PWR_MODULE_ENABLED
stm32f4xx_hal_conf.h, 179
HAL_RCC_MODULE_ENABLED
stm32f4xx_hal_conf.h, 179
HAL_ResumeTick
stm32f4xx_hal_timebase_tim.c, 285
HAL_SPI_MODULE_ENABLED
stm32f4xx_hal_conf.h, 179
HAL_SPI_MspDeInit
spi.c, 282
HAL_SPI_MspInit
spi.c, 282
HAL_SuspendTick
stm32f4xx_hal_timebase_tim.c, 285
HAL_TIM_Base_MspDeInit
tim.c, 301
HAL_TIM_Base_MspInit
tim.c, 302
HAL_TIM_MODULE_ENABLED
stm32f4xx_hal_conf.h, 179
HAL_TIM_PeriodElapsedCallback
main.c, 262
handle
uv_binary_semaphore_info, 86
uv_mutex_info, 92
uv_semaphore_info, 95
handleCANbusError
can.c, 233
HardFault_Handler
stm32f4xx_it.c, 289
stm32f4xx_it.h, 196
Hardware_Error
imd.h, 144
hardware_fault
motor_controller.h, 158
hcan2
can.c, 234
can.h, 116
stm32f4xx_it.c, 290
hdma_adc1
adc.c, 227
stm32f4xx_it.c, 290
High_Battery_Voltage
imd.h, 144
HIGH_PRIORITY
State Engine API, 24
High_Uncertainty
imd.h, 144
HSE_STARTUP_TIMEOUT
stm32f4xx_hal_conf.h, 179
HSE_VALUE
stm32f4xx_hal_conf.h, 180
STM32F4xx_System_Private_Includes, 52
HSI_VALUE
stm32f4xx_hal_conf.h, 180
STM32F4xx_System_Private_Includes, 52
hspi1
spi.c, 283
spi.h, 173
htim1
stm32f4xx_hal_timebase_tim.c, 286
stm32f4xx_it.c, 291
htim3
tim.c, 302
tim.h, 199

IDLE_TASK_PRIORITY
State Engine API, 24
IGBT_temp_max_limit
motor_controller.h, 158
igbt_temperature
motor_controller.h, 158
IGBT_temperature_warning
motor_controller.h, 158
IMD
uvfr_utils.h, 217
imd.c
IMD_Check_Battery_Voltage, 250
IMD_Check_Error_Flags, 250
IMD_Check_Isolation_Capacitances, 250
IMD_Check_Isolation_Resistances, 251
IMD_Check_Isolation_State, 251
IMD_Check_Max_Battery_Working_Voltage, 251
IMD_Check_Part_Name, 251
IMD_Check_Safety_Touch_Current, 252
IMD_Check_Safety_Touch_Energy, 252
IMD_Check_Serial_Number, 252
IMD_Check_Status_Bits, 252
IMD_Check_Temperature, 253
IMD_Check_Uptime, 253
IMD_Check_Version, 253
IMD_Check_Voltages_Vp_and_Vn, 253
IMD_error_flags_requested, 255
IMD_Expected_Part_Name, 255
IMD_Expected_Serial_Number, 255
IMD_Expected_Version, 255
IMD_High_Uncertainty, 256

IMD_Parse_Message, 254
IMD_Part_Name_0_Set, 256
IMD_Part_Name_1_Set, 256
IMD_Part_Name_2_Set, 256
IMD_Part_Name_3_Set, 256
IMD_Part_Name_Set, 257
IMD_Read_Part_Name, 257
IMD_Read_Serial_Number, 257
IMD_Read_Version, 257
IMD_Request_Status, 254
IMD_Serial_Number_0_Set, 257
IMD_Serial_Number_1_Set, 258
IMD_Serial_Number_2_Set, 258
IMD_Serial_Number_3_Set, 258
IMD_Serial_Number_Set, 258
IMD_Startup, 254
IMD_status_bits, 258
IMD_Temperature, 259
IMD_Version_0_Set, 259
IMD_Version_1_Set, 259
IMD_Version_2_Set, 259
IMD_Version_Set, 259
initIMD, 254

imd.h
battery_voltage, 144
Err_CH, 143
Err_clock, 143
Err_temp, 143
Err_Vexi, 143
Err_Vpwr, 143
Err_Vx1, 143
Err_Vx2, 143
Err_VxR, 143
Err_Watchdog, 143
Error_flags, 144
Exc_off, 144
Hardware_Error, 144
High_Battery_Voltage, 144
High_Uncertainty, 144
IMD_Check_Battery_Voltage, 145
IMD_Check_Error_Flags, 145
IMD_Check_Isolation_Capacitances, 145
IMD_Check_Isolation_Resistances, 145
IMD_Check_Isolation_State, 145
IMD_Check_Max_Battery_Working_Voltage, 146
IMD_Check_Part_Name, 146
IMD_Check_Safety_Touch_Current, 146
IMD_Check_Safety_Touch_Energy, 146
IMD_Check_Serial_Number, 147
IMD_Check_Status_Bits, 147
IMD_Check_Temperature, 147
IMD_Check_Uptime, 147
IMD_Check_Version, 148
IMD_Check_Voltages_Vp_and_Vn, 148
imd_error_flags, 142
imd_high_resolution_measurements, 143
imd_manufacturer_requests, 143
IMD_Parse_Message, 148
IMD_Request_Status, 148
IMD_Startup, 149
imd_status_bits, 144
imd_status_requests, 144
initIMD, 149
isolation_capacitances, 144
isolation_resistances, 144
isolation_state, 144
Isolation_status_bit0, 144
Isolation_status_bit1, 144
Low_Battery_Voltage, 144
Max_battery_working_voltage, 144
Part_name_0, 143
Part_name_1, 143
Part_name_2, 143
Part_name_3, 143
safety_touch_current, 144
safety_touch_energy, 144
Serial_number_0, 143
Serial_number_1, 143
Serial_number_2, 144
Serial_number_3, 144
Temperature, 144
Touch_energy_fault, 144
Uptime_counter, 144
Vb_hi_res, 143
Version_0, 143
Version_1, 143
Version_2, 143
Vexc_hi_res, 143
Vn_hi_res, 143
voltages_Vp_and_Vn, 144
Vp_hi_res, 143
Vpwr_hi_res, 143

IMD_CAN_ID_Rx
constants.h, 117

IMD_CAN_ID_Tx
constants.h, 117

IMD_Check_Battery_Voltage
imd.c, 250
imd.h, 145

IMD_Check_Error_Flags
imd.c, 250
imd.h, 145

IMD_Check_Isolation_Capacitances
imd.c, 250
imd.h, 145

IMD_Check_Isolation_Resistances
imd.c, 251
imd.h, 145

IMD_Check_Isolation_State
imd.c, 251
imd.h, 145

IMD_Check_Max_Battery_Working_Voltage
imd.c, 251
imd.h, 146

IMD_Check_Part_Name
imd.c, 251

imd.h, 146
IMD_Check_Safety_Touch_Current
imd.c, 252
imd.h, 146
IMD_Check_Safety_Touch_Energy
imd.c, 252
imd.h, 146
IMD_Check_Serial_Number
imd.c, 252
imd.h, 147
IMD_Check_Status_Bits
imd.c, 252
imd.h, 147
IMD_Check_Temperature
imd.c, 253
imd.h, 147
IMD_Check_Uptime
imd.c, 253
imd.h, 147
IMD_Check_Version
imd.c, 253
imd.h, 148
IMD_Check_Voltages_Vp_and_Vn
imd.c, 253
imd.h, 148
imd_error_flags
imd.h, 142
IMD_error_flags_requested
imd.c, 255
IMD_Expected_Part_Name
imd.c, 255
IMD_Expected_Serial_Number
imd.c, 255
IMD_Expected_Version
imd.c, 255
imd_high_resolution_measurements
imd.h, 143
IMD_High_Uncertainty
imd.c, 256
imd_manufacturer_requests
imd.h, 143
IMD_Parse_Message
imd.c, 254
imd.h, 148
IMD_Part_Name_0_Set
imd.c, 256
IMD_Part_Name_1_Set
imd.c, 256
IMD_Part_Name_2_Set
imd.c, 256
IMD_Part_Name_3_Set
imd.c, 256
IMD_Part_Name_Set
imd.c, 257
IMD_Read_Part_Name
imd.c, 257
IMD_Read_Serial_Number
imd.c, 257
IMD_Read_Version
imd.c, 257
IMD_Request_Status
imd.c, 254
imd.h, 148
IMD_Serial_Number_0_Set
imd.c, 257
IMD_Serial_Number_1_Set
imd.c, 258
IMD_Serial_Number_2_Set
imd.c, 258
IMD_Serial_Number_3_Set
imd.c, 258
IMD_Serial_Number_Set
imd.c, 258
imd_settings
uv_vehicle_settings, 104
IMD_Startup
imd.c, 254
imd.h, 149
IMD_status_bits
imd.c, 258
imd_status_bits
imd.h, 144
imd_status_requests
imd.h, 144
IMD_Temperature
imd.c, 259
IMD_Version_0_Set
imd.c, 259
IMD_Version_1_Set
imd.c, 259
IMD_Version_2_Set
imd.c, 259
IMD_Version_Set
imd.c, 259
Implausible
driving_loop.h, 127
INCLUDE_eTaskGetState
FreeRTOSConfig.h, 137
INCLUDE_pcTaskGetTaskName
FreeRTOSConfig.h, 137
INCLUDE_uxTaskGetStackHighWaterMark
FreeRTOSConfig.h, 137
INCLUDE_uxTaskGetStackHighWaterMark2
FreeRTOSConfig.h, 137
INCLUDE_uxTaskPriorityGet
FreeRTOSConfig.h, 137
INCLUDE_vTaskCleanUpResources
FreeRTOSConfig.h, 137
INCLUDE_vTaskDelay
FreeRTOSConfig.h, 138
INCLUDE_vTaskDelayUntil
FreeRTOSConfig.h, 138
INCLUDE_vTaskDelete
FreeRTOSConfig.h, 138
INCLUDE_vTaskPrioritySet
FreeRTOSConfig.h, 138

INCLUDE_vTaskSuspend
FreeRTOSConfig.h, 138
INCLUDE_xEventGroupSetBitFromISR
FreeRTOSConfig.h, 138
INCLUDE_xQueueGetMutexHolder
FreeRTOSConfig.h, 139
INCLUDE_xSemaphoreGetMutexHolder
FreeRTOSConfig.h, 139
INCLUDE_xTaskAbortDelay
FreeRTOSConfig.h, 139
INCLUDE_xTaskDelayUntil
FreeRTOSConfig.h, 139
INCLUDE_xTaskGetCurrentTaskHandle
FreeRTOSConfig.h, 139
INCLUDE_xTaskGetHandle
FreeRTOSConfig.h, 139
INCLUDE_xTaskGetSchedulerState
FreeRTOSConfig.h, 140
INCLUDE_xTimerPendFunctionCall
FreeRTOSConfig.h, 140
INIT_CHECK_PERIOD
uvfr_utils.h, 212
init_info_queue
uv_init_task_args, 89
init_params
uv_internal_params, 91
init_settings
freertos.c, 247
init_task_handle
freertos.c, 247
uvfr_utils.c, 313
initDaqTask
daq.c, 238
daq.h, 121
initDrivingLoop
driving_loop.c, 242
driving_loop.h, 127
initialise_monitor_handles
syscalls.c, 296
initIMD
imd.c, 254
imd.h, 149
initOdometer
odometer.c, 272
odometer.h, 160
initPDU
pdu.c, 273
pdu.h, 163
initTempMonitor
temp_monitoring.c, 299
temp_monitoring.h, 198
INORDER
rb_tree.h, 169
insertCANMessageHandler
UVFR CANbus API, 49
insertRepair
rb_tree.c, 277
INSTRUCTION_CACHE_ENABLE
stm32f4xx_hal_conf.h, 180
integral_memory_max
motor_controller.h, 156
motor_controllor_settings, 76
integral_time_constant
motor_controller.h, 156
motor_controllor_settings, 76
intended_recipient
uv_task_msg_t, 102
internal_hardware_voltage_problem
motor_controller.h, 158
is_default
uv_vehicle_settings, 104
isolation_capacitances
imd.h, 144
isolation_resistances
imd.h, 144
isolation_state
imd.h, 144
Isolation_status_bit0
imd.h, 144
Isolation_status_bit1
imd.h, 144
isPowerOfTwo
Utility Macros, 45

killEmAll
State Engine Internals, 32
killSelf
State Engine Internals, 32

leakage_inductance_ph_ph
motor_controller.h, 156
left
rbnode, 78
limp
uvfr_utils.h, 217
linear_speed_map
driving_loop.h, 127
linear_torque_map
driving_loop.h, 127
linear_torque_map_args, 74
driving_loop.h, 125
offset, 75
slope, 75
loggable_params
daq.h, 120
Low_Battery_Voltage
imd.h, 144
LOW_PRIORITY
State Engine API, 24
LSE_STARTUP_TIMEOUT
stm32f4xx_hal_conf.h, 180
LSE_VALUE
stm32f4xx_hal_conf.h, 180
LSI_VALUE
stm32f4xx_hal_conf.h, 181

MAC_ADDR0

stm32f4xx_hal_conf.h, 181
MAC_ADDR1
stm32f4xx_hal_conf.h, 181
MAC_ADDR2
stm32f4xx_hal_conf.h, 181
MAC_ADDR3
stm32f4xx_hal_conf.h, 181
MAC_ADDR4
stm32f4xx_hal_conf.h, 182
MAC_ADDR5
stm32f4xx_hal_conf.h, 182
main
main.c, 263
main.c
adc1_APPS1, 264
adc1_APPS2, 264
adc1_BPS1, 265
adc1_BPS2, 265
adc2_CoolantFlow, 265
adc2_CoolantTemp, 265
adc_buf1, 265
adc_buf2, 266
DEBUG_CAN_IN_MAIN, 261
Error_Handler, 261
HAL_ADC_ConvCpltCallback, 262
HAL_ADC_LevelOutOfWindowCallback, 262
HAL_GPIO_EXTI_Callback, 262
HAL_TIM_PeriodElapsedCallback, 262
main, 263
MX_FREERTOS_Init, 263
SystemClock_Config, 264
main.h
Blue_LED_GPIO_Port, 150
Blue_LED_Pin, 150
Error_Handler, 152
Orange_LED_GPIO_Port, 150
Orange_LED_Pin, 151
Red_LED_GPIO_Port, 151
Red_LED_Pin, 151
Start_Button_Input_EXTI_IRQn, 151
Start_Button_Input_GPIO_Port, 151
Start_Button_Input_Pin, 151
mains_voltage_max_limit
motor_controller.h, 158
mains_voltage_min_limit
motor_controller.h, 158
map_fn_params
drivingMode, 72
map_mode
driving_loop.h, 127
max_acc_pwr
drivingMode, 73
max_accum_current_5s
driving_loop_args, 69
max_apps_offset
driving_loop_args, 69
max_apps_value
driving_loop_args, 69
Max_battery_working_voltage
imd.h, 144
max_BPS_value
driving_loop_args, 69
MAX_CELL_TEMP
daq.h, 120
max_current
drivingMode, 73
MAX_INIT_TIME
uvfr_utils.h, 212
MAX_LOGGABLE_PARAMS
daq.h, 120
max_motor_speed
motor_controller.c, 270
max_motor_torque
drivingMode, 73
MAX_NUM_MANAGED_TASKS
State Engine, 10
max_svc_task_period
uv_os_settings, 93
max_task_period
uv_os_settings, 93
MC_CAN_ID_Rx
constants.h, 117
MC_CAN_ID_Tx
constants.h, 117
mc_CAN_timeout
bms_settings_t, 60
motor_controllor_settings, 76
MC_Check_Error_Warning
motor_controller.c, 267
MC_Check_Firmware
motor_controller.c, 267
MC_Check_Serial_Number
motor_controller.c, 267
MC_CURRENT
daq.h, 120
mc_default_settings
motor_controller.c, 271
MC_ERRORS
daq.h, 120
MC_Expected_FW_Version
motor_controller.c, 271
MC_Expected_Serial_Number
motor_controller.c, 271
MC_POWER
driving_loop.h, 126
MC_Request_Data
motor_controller.c, 267
MC_RPM
driving_loop.h, 126
mc_settings
uv_vehicle_settings, 104
MC_Startup
motor_controller.c, 268
motor_controller.h, 159
MC_TEMP
daq.h, 120

MC_Torque
driving_loop.h, 126
MC_Validate
motor_controller.c, 268
MC_VOLTAGE
daq.h, 120
MEDIUM_PRIORITY
State Engine API, 24
MemManage_Handler
stm32f4xx_it.c, 289
stm32f4xx_it.h, 197
message_size
uv_task_msg_t, 102
message_type
uv_task_msg_t, 102
meta_id
uv_scd_response, 95
meta_task_handle
daq_child_task, 62
state_change_daemon_args, 83
uv_init_task_args, 89
min
rbtree, 81
min_apps_offset
driving_loop_args, 70
min_apps_value
driving_loop_args, 70
min_BPS_value
driving_loop_args, 70
MIN_CELL_TEMP
daq.h, 120
min_task_period
uv_os_settings, 94
minimum_daq_period
daq_loop_args, 64
minimum_magnetising_current
motor_controller.h, 156
motor_continuous_current
motor_controller.h, 156
MOTOR_CONTROLLER
uvfr_utils.h, 217
motor_controller.c
canRxQueue, 270
canTxQueue, 270
max_motor_speed, 270
MC_Check_Error_Warning, 267
MC_Check_Firmware, 267
MC_Check_Serial_Number, 267
mc_default_settings, 271
MC_Expected_FW_Version, 271
MC_Expected_Serial_Number, 271
MC_Request_Data, 267
MC_Startup, 268
MC_Validate, 268
MotorControllerErrorHandler, 268
MotorControllerSpinTest, 269
Parse_Bamocar_Response, 269
WaitFor_CAN_Response, 270
motor_controller.h
AC_current_offset_fault, 158
accelerate_ramp, 156
ADC_measurement_problem, 158
ADC_sequencer_problem, 158
air_temperature, 158
auxiliary_voltage_min_limit, 158
bleed_resistor_overload, 158
bleeder_resistor_warning, 158
CAN_timeout_error, 158
check_ecode_ID, 158
clear_errors, 157
critical_AC_current, 158
current_derate_temperature, 158
current_feed_forward, 156
DC_bus_voltage, 155
DEFAULT_MOTOR_CONTROLLER_CAN_TIMEOUT, 154
dismantling_ramp, 156
ecode_timeout_error, 158
eprom_read_error, 158
feedback_signal_error, 158
feedback_signal_problem, 158
firmware_version, 157
FIRMWARE_VERSION_REGISTER, 154
hardware_fault, 158
IGBT_temp_max_limit, 158
igbt_temperature, 158
IGBT_temperature_warning, 158
integral_memory_max, 156
integral_time_constant, 156
internal_hardware_voltage_problem, 158
leakage_inductance_ph_ph, 156
mains_voltage_max_limit, 158
mains_voltage_min_limit, 158
MC_Startup, 159
minimum_magnetising_current, 156
motor_continuous_current, 156
motor_controller_current_parameters, 154
motor_controller_errors_warnings, 157
motor_controller_io, 155
motor_controller_limp_mode, 155
motor_controller_measurements, 155
motor_controller_motor_constants, 155
motor_controller_PI_values, 156
motor_controller_repeating_time, 156
motor_controller_settings, 154
motor_controller_speed_parameters, 157
motor_controller_startup, 157
motor_controller_status_information_errors_warnings, 157
motor_controller_temperatures, 158
motor_ke_constant, 156
motor_kt_constant, 156
motor_magnetising_inductance, 156
motor_max_current, 156
motor_pole_number, 156
motor_temp_max_limit, 158

motor_temperature, 158
motor_temperature_switch_off_point, 156
motor_temperature_warning, 158
N_actual, 157
N_cmd, 157
N_error, 157
N_lim, 155
N_lim_minus, 155
N_lim_plus, 155
N_set, 157
nominal_magnitizing_current, 156
nominal_motor_frequency, 156
nominal_motor_voltage, 156
none, 157
one_hundred_ms, 157
parameter_conflict_detected, 158
power_factor, 156
proportional_gain, 156
proportional_gain_2, 156
race_away_detected, 158
ramp_set_current, 156
rated_motor_speed, 156
recuperation_ramp, 156
rotate_field_enable_not_present_norun, 158
rotate_field_enable_not_present_run, 158
rotor_resistance, 156
SERIAL_NUMBER_REGISTER, 154
special_CPU_fault, 158
speed_actual_resolution_limit, 158
stator_leakage_inductance, 156
stator_resistance_ph_ph, 156
temp_sensor_pt1, 159
temp_sensor_pt2, 159
temp_sensor_pt3, 159
temp_sensor_pt4, 159
time_constant_rotor, 156
time_constant_stator, 156
todo1, 155
todo6969, 155
tripzone_glitch_detected, 158
Vout_saturation_max_limit, 158
warning_5, 158
warning_9, 158
watchdog_reset, 158
motor_controller_current_parameters
motor_controller.h, 154
motor_controller_errors_warnings
motor_controller.h, 157
motor_controller_io
motor_controller.h, 155
motor_controller_limp_mode
motor_controller.h, 155
motor_controller_measurements
motor_controller.h, 155
motor_controller_motor_constants
motor_controller.h, 155
motor_controller_PI_values
motor_controller.h, 156
motor_controller_repeating_time
motor_controller.h, 156
motor_controller_settings
motor_controller.h, 154
motor_controller_speed_parameters
motor_controller.h, 157
motor_controller_startup
motor_controller.h, 157
motor_controller_status_information_errors_warnings
motor_controller.h, 157
motor_controller_temperatures
motor_controller.h, 158
motor_controllor_settings, 75
can_id_rx, 76
can_id_tx, 76
integral_memory_max, 76
integral_time_constant, 76
mc_CAN_timeout, 76
proportional_gain, 76
MOTOR_CURRENT
daq.h, 120
motor_ke_constant
motor_controller.h, 156
motor_kt_constant
motor_controller.h, 156
motor_magnetising_inductance
motor_controller.h, 156
motor_max_current
motor_controller.h, 156
motor_pole_number
motor_controller.h, 156
MOTOR_RPM
daq.h, 120
MOTOR_TEMP
daq.h, 120
motor_temp_max_limit
motor_controller.h, 158
motor_temperature
motor_controller.h, 158
motor_temperature_switch_off_point
motor_controller.h, 156
motor_temperature_warning
motor_controller.h, 158
MotorControllerErrorHandler
motor_controller.c, 268
MotorControllerSpinTest
motor_controller.c, 269
msg_contents
uv_task_msg_t, 102
msg_id
uv_CAN_msg, 87
mutex
access_control_info, 59
MX_ADC1_Init
adc.c, 226
adc.h, 110
MX_ADC2_Init
adc.c, 226

adc.h, 110
MX_CAN2_Init
can.c, 233
can.h, 115
MX_DMA_Init
dma.c, 241
dma.h, 124
MX_FREERTOS_Init
freertos.c, 244
main.c, 263
MX_GPIO_Init
gpio.c, 249
gpio.h, 141
MX_SPI1_Init
spi.c, 282
spi.h, 172
MX_TIM3_Init
tim.c, 302
tim.h, 199

N_actual
motor_controller.h, 157
N_cmd
motor_controller.h, 157
N_error
motor_controller.h, 157
N_lim
motor_controller.h, 155
N_lim_minus
motor_controller.h, 155
N_lim_plus
motor_controller.h, 155
N_set
motor_controller.h, 157
nchar
uv_init_task_response, 90
next
CAN_Callback, 61
daq_param_list_node, 65
nil
rbtree, 81
NMI_Handler
stm32f4xx_it.c, 289
stm32f4xx_it.h, 197
nominal_magnitizing_current
motor_controller.h, 156
nominal_motor_frequency
motor_controller.h, 156
nominal_motor_voltage
motor_controller.h, 156
none
motor_controller.h, 157
normal
uvfr_utils.h, 217
nuke_hash_table
can.c, 233
can.h, 115
nukeSettings
uvfr_settings.c, 303
uvfr_settings.h, 202
num_driving_modes
driving_loop_args, 70

odometer.c
initOdometer, 272
odometerTask, 272
odometer.h
initOdometer, 160
odometerTask, 160
odometerTask
odometer.c, 272
odometer.h, 160
offset
exp_torque_map_args, 74
linear_torque_map_args, 75
oled.h
amogus, 161
oled_config, 161
oled_Write, 161
oled_Write_Cmd, 161
oled_Write_Data, 161
refresh_OLED, 161
wait, 162
oled_config
oled.h, 161
oled_Write
oled.h, 161
oled_Write_Cmd
oled.h, 161
oled_Write_Data
oled.h, 161
one_hundred_ms
motor_controller.h, 157
Orange_LED_GPIO_Port
main.h, 150
Orange_LED_Pin
main.h, 151
os_settings
uv_vehicle_settings, 105

p_status, 77
activation_time, 77
peripheral_status, 77
uvfr_utils.h, 213
padding
daq_loop_args, 64
padding2
daq_loop_args, 64
param_idx
daq_param_list_node, 65
param_list
daq_child_task, 62
param_LUT
daq.c, 239
daq.h, 121
parameter_conflict_detected
motor_controller.h, 158
parent

rbnode, 79
uv_task_info, 97
parent_msg_queue
task_management_info, 84
Parse_Bamocar_Response
motor_controller.c, 269
Part_name_0
imd.h, 143
Part_name_1
imd.h, 143
Part_name_2
imd.h, 143
Part_name_3
imd.h, 143
PDU
uvfr_utils.h, 217
pdu.c
initPDU, 273
PDU_disable_brake_light, 273
PDU_disable_coolant_pump, 274
PDU_disable_cooling_fans, 274
PDU_disable_motor_controller, 274
PDU_disable_shutdown_circuit, 274
PDU_enable_brake_light, 274
PDU_enable_coolant_pump, 275
PDU_enable_cooling_fans, 275
PDU_enable_motor_controller, 275
PDU_enable_shutdown_circuit, 275
PDU_speaker_chirp, 275
pdu.h
disable_brake_light_msg, 163
disable_coolant_pump_msg, 163
disable_left_cooling_fan_msg, 163
disable_motor_controller_msg, 163
disable_right_cooling_fan_msg, 163
disable_shutdown_circuit_msg, 163
disable_speaker_msg, 163
enable_brake_light_msg, 163
enable_coolant_pump_msg, 163
enable_left_cooling_fan_msg, 163
enable_motor_controller_msg, 163
enable_right_cooling_fan_msg, 163
enable_shutdown_circuit_msg, 163
enable_speaker_msg, 163
initPDU, 163
PDU_disable_brake_light, 163
PDU_disable_coolant_pump, 164
PDU_disable_cooling_fans, 164
PDU_disable_motor_controller, 164
PDU_disable_shutdown_circuit, 164
PDU_enable_brake_light, 164
PDU_enable_coolant_pump, 165
PDU_enable_cooling_fans, 165
PDU_enable_motor_controller, 165
PDU_enable_shutdown_circuit, 165
pdu_messages_20A, 162
pdu_messages_5A, 163
PDU_speaker_chirp, 165
PDU_CAN_ID_Tx
constants.h, 117
PDU_disable_brake_light
pdu.c, 273
pdu.h, 163
PDU_disable_coolant_pump
pdu.c, 274
pdu.h, 164
PDU_disable_cooling_fans
pdu.c, 274
pdu.h, 164
PDU_disable_motor_controller
pdu.c, 274
pdu.h, 164
PDU_disable_shutdown_circuit
pdu.c, 274
pdu.h, 164
PDU_enable_brake_light
pdu.c, 274
pdu.h, 164
PDU_enable_coolant_pump
pdu.c, 275
pdu.h, 165
PDU_enable_cooling_fans
pdu.c, 275
pdu.h, 165
PDU_enable_motor_controller
pdu.c, 275
pdu.h, 165
PDU_enable_shutdown_circuit
pdu.c, 275
pdu.h, 165
pdu_messages_20A
pdu.h, 162
pdu_messages_5A
pdu.h, 163
pdu_settings
uv_vehicle_settings, 105
PDU_speaker_chirp
pdu.c, 275
pdu.h, 165
period
daq_child_task, 62
daq_datapoint, 63
driving_loop_args, 70
peripheral_status
p_status, 77
uv_internal_params, 92
PHY_AUTONEGO_COMPLETE
stm32f4xx_hal_conf.h, 182
PHY_AUTONEGOTIATION
stm32f4xx_hal_conf.h, 182
PHY_BCR
stm32f4xx_hal_conf.h, 182
PHY_BSR
stm32f4xx_hal_conf.h, 183
PHY_CONFIG_DELAY
stm32f4xx_hal_conf.h, 183

PHY_DUPLEX_STATUS
stm32f4xx_hal_conf.h, 183
PHY_FULLDUPLEX_100M
stm32f4xx_hal_conf.h, 183
PHY_FULLDUPLEX_10M
stm32f4xx_hal_conf.h, 183
PHY_HALFDUPLEX_100M
stm32f4xx_hal_conf.h, 184
PHY_HALFDUPLEX_10M
stm32f4xx_hal_conf.h, 184
PHY_ISOLATE
stm32f4xx_hal_conf.h, 184
PHY_JABBER_DETECTION
stm32f4xx_hal_conf.h, 184
PHY_LINKED_STATUS
stm32f4xx_hal_conf.h, 184
PHY_LOOPBACK
stm32f4xx_hal_conf.h, 185
PHY_POWERDOWN
stm32f4xx_hal_conf.h, 185
PHY_READ_TO
stm32f4xx_hal_conf.h, 185
PHY_RESET
stm32f4xx_hal_conf.h, 185
PHY_RESET_DELAY
stm32f4xx_hal_conf.h, 185
PHY_RESTART_AUTONEGOTIATION
stm32f4xx_hal_conf.h, 186
PHY_SPEED_STATUS
stm32f4xx_hal_conf.h, 186
PHY_SR
stm32f4xx_hal_conf.h, 186
PHY_WRITE_TO
stm32f4xx_hal_conf.h, 186
Plausible
driving_loop.h, 127
POSTORDER
rb_tree.h, 169
POWER_DERATE_FACTOR
daq.h, 120
power_factor
motor_controller.h, 156
PREFETCH_ENABLE
stm32f4xx_hal_conf.h, 186
PREORDER
rb_tree.h, 169
previous_state
State Engine, 12
print
rb_tree.c, 278
rbtree, 81
proccessSCDMsg
State Engine Internals, 33
PROGRAMMING
State Engine API, 26
proportional_gain
motor_controller.h, 156
motor_controllor_settings, 76
proportional_gain_2
motor_controller.h, 156

race_away_detected
motor_controller.h, 158
ramp_set_current
motor_controller.h, 156
rated_motor_speed
motor_controller.h, 156
RB_APPLY
rb_tree.h, 167
rb_apply
rb_tree.c, 278
RB_DUP
rb_tree.h, 167
RB_FIRST
rb_tree.h, 167
RB_ISEMPTY
rb_tree.h, 168
RB_MIN
rb_tree.h, 168
RB_MINIMAL
rb_tree.h, 168
RB_NIL
rb_tree.h, 168
RB_ROOT
rb_tree.h, 168
rb_tree.c
checkBlackHeight, 276
checkOrder, 277
deleteRepair, 277
destroyAllNodes, 277
insertRepair, 277
print, 278
rb_apply, 278
rbCheckBlackHeight, 278
rbCheckOrder, 278
rbCreate, 279
rbDelete, 279
rbDestroy, 279
rbFind, 279
rbInsert, 280
rbPrint, 280
rbSuccessor, 280
rotateLeft, 280
rotateRight, 281
rb_tree.h
BLACK, 167
INORDER, 169
POSTORDER, 169
PREORDER, 169
RB_APPLY, 167
RB_DUP, 167
RB_FIRST, 167
RB_ISEMPTY, 168
RB_MIN, 168
RB_MINIMAL, 168
RB_NIL, 168
RB_ROOT, 168

rbApplyNode, 169
rbCheckBlackHeight, 169
rbCheckOrder, 170
rbCreate, 170
rbDelete, 170
rbDestroy, 170
rbFind, 171
rbInsert, 171
rbnode, 169
rbPrint, 171
rbSuccessor, 171
rbtraversal, 169
RED, 168
rbApplyNode
rb_tree.h, 169
rbCheckBlackHeight
rb_tree.c, 278
rb_tree.h, 169
rbCheckOrder
rb_tree.c, 278
rb_tree.h, 170
rbCreate
rb_tree.c, 279
rb_tree.h, 170
rbDelete
rb_tree.c, 279
rb_tree.h, 170
rbDestroy
rb_tree.c, 279
rb_tree.h, 170
rbFind
rb_tree.c, 279
rb_tree.h, 171
rbInsert
rb_tree.c, 280
rb_tree.h, 171
rbnode, 78
color, 78
data, 78
left, 78
parent, 79
rb_tree.h, 169
right, 79
rbPrint
rb_tree.c, 280
rb_tree.h, 171
rbSuccessor
rb_tree.c, 280
rb_tree.h, 171
rbtraversal
rb_tree.h, 169
rbtree, 79
compare, 80
count, 80
destroy, 80
min, 81
nil, 81
print, 81
root, 81
REALTIME_PRIORITY
State Engine API, 24
recuperation_ramp
motor_controller.h, 156
RED
rb_tree.h, 168
Red_LED_GPIO_Port
main.h, 151
Red_LED_Pin
main.h, 151
refresh_OLED
oled.h, 161
regen_rpm_cutoff
driving_loop_args, 71
reset_handle
uvfr_utils.c, 313
response_val
uv_scd_response, 95
right
rbnode, 79
root
rbtree, 81
rotate_field_enable_not_present_norun
motor_controller.h, 158
rotate_field_enable_not_present_run
motor_controller.h, 158
rotateLeft
rb_tree.c, 280
rotateRight
rb_tree.c, 281
rotor_resistance
motor_controller.h, 156
Rx_msg_queue
can.c, 234
RxData
constants.c, 235
constants.h, 117
RxHeader
constants.c, 235
constants.h, 117

s_curve_speed_map
driving_loop.h, 127
s_curve_torque_map
driving_loop.h, 127
s_curve_torque_map_args, 82
a, 82
b, 82
c, 83
driving_loop.h, 126
safePtrRead
Utility Macros, 45
safePtrWrite
Utility Macros, 45
safety_touch_current
imd.h, 144
safety_touch_energy
imd.h, 144

SCD_active
State Engine, 13
scd_handle_ptr
State Engine, 13
semaphore
access_control_info, 59
sender
uv_task_msg_t, 103
Serial_number_0
imd.h, 143
Serial_number_1
imd.h, 143
Serial_number_2
imd.h, 144
Serial_number_3
imd.h, 144
SERIAL_NUMBER_REGISTER
motor_controller.h, 154
serializeBigE16
Utility Macros, 45
serializeBigE32
Utility Macros, 46
serializeSmallE16
Utility Macros, 46
serializeSmallE32
Utility Macros, 46
setBits
Utility Macros, 46
setup_extern_devices
uvfr_utils.c, 309
setupDefaultSettings
uvfr_settings.c, 303
slope
linear_torque_map_args, 75
special_CPU_fault
motor_controller.h, 158
specific_args
uv_init_task_args, 89
speed_actual_resolution_limit
motor_controller.h, 158
spi.c
HAL_SPI_MspDeInit, 282
HAL_SPI_MspInit, 282
hspi1, 283
MX_SPI1_Init, 282
spi.h
hspi1, 173
MX_SPI1_Init, 172
SRC_UVFR_SETTINGS_C_
uvfr_settings.c, 303
stack_size
uv_task_info, 98
Start_Button_Input_EXTI_IRQn
main.h, 151
Start_Button_Input_GPIO_Port
main.h, 151
Start_Button_Input_Pin
main.h, 151
startDaqSubTasks
daq.c, 238
StartDefaultTask
freertos.c, 245
StartDrivingLoop
driving_loop.c, 242
driving_loop.h, 128
State Engine, 9
_next_svc_task_id, 11
_next_task_id, 12
_task_register, 12
compareTaskByName, 11
default_os_settings, 12
MAX_NUM_MANAGED_TASKS, 10
previous_state, 12
SCD_active, 13
scd_handle_ptr, 13
state_change_daemon_args, 10
state_change_queue, 13
svc_task_manager, 13
task_manager, 13
task_name_lut, 14
task_name_tree, 14
uvSVCTaskManager, 11
vehicle_state, 14
State Engine API, 15
ABOVE_NORMAL, 24
BELOW_NORMAL, 24
changeVehicleState, 26
HIGH_PRIORITY, 24
IDLE_TASK_PRIORITY, 24
LOW_PRIORITY, 24
MEDIUM_PRIORITY, 24
PROGRAMMING, 26
REALTIME_PRIORITY, 24
task_management_info, 22
task_priority, 22, 24
task_status_block, 22
UV_BOOT, 26
UV_COULDNT_DELETE, 24
UV_COULDNT_SUSPEND, 24
UV_DRIVING, 26
UV_ERROR_STATE, 26
UV_HALT, 26
UV_INIT, 26
UV_KILL_CMD, 25
UV_LAUNCH_CONTROL, 26
UV_NO_CMD, 25
uv_os_settings, 23
UV_READY, 26
uv_scd_response, 23
uv_scd_response_e, 24
UV_SUCCESSFUL_DELETION, 24
UV_SUCCESSFUL_SUSPENSION, 24
UV_SUSPEND_CMD, 25
UV_SUSPENDED, 26
UV_TASK_AWAITING_DELETION, 17
uv_task_cmd, 23

uv_task_cmd_e, 25
UV_TASK_DEADLINE_FIRM, 17
UV_TASK_DEADLINE_HARD, 17
UV_TASK_DEADLINE_MASK, 17
UV_TASK_DEADLINE_NOT_ENFORCED, 17
UV_TASK_DEFER_DELETION, 18
UV_TASK_DELAYING, 18
UV_TASK_DELETED, 25
UV_TASK_DORMANT_SVC, 18
UV_TASK_ERR_IN_CHILD, 18
UV_TASK_GENERIC_SVC, 18
uv_task_info, 23
UV_TASK_IS_CHILD, 18
UV_TASK_IS_ORPHAN, 19
UV_TASK_IS_PARENT, 19
UV_TASK_LOG_MEM_USAGE, 19
UV_TASK_LOG_START_STOP_TIME, 19
UV_TASK_MANAGER_MASK, 19
UV_TASK_MISSION_CRITICAL, 19
UV_TASK_NOT_STARTED, 25
UV_TASK_PERIODIC_SVC, 20
UV_TASK_PRIO_INCREMENTATION, 20
UV_TASK_RUNNING, 25
UV_TASK_SCD_IGNORE, 20
UV_TASK_START_CMD, 25
uv_task_state_t, 25
uv_task_status, 23
UV_TASK_SUSPENDED, 25
UV_TASK_VEHICLE_APPLICATION, 20
UV_UNSAFE_STATE, 24
uv_vehicle_state, 23
uv_vehicle_state_t, 25
uvCreateTask, 27
uvDeInitStateEngine, 27
uvInitStateEngine, 27
uvStartStateMachine, 27
uvTaskDelay, 20
uvTaskDelayUntil, 21
uvTaskIsDelaying, 21
uvTaskResetDelayBit, 21
uvTaskResetDeletionBit, 21
uvTaskSetDelayBit, 22
uvTaskSetDeletionBit, 22
State Engine Internals, 29
__uvPanic, 30
_stateChangeDaemon, 30
_uvValidateSpecificTask, 32
addTaskToTaskRegister, 32
killEmAll, 32
killSelf, 32
proccessSCDMsg, 33
suspendSelf, 33
uvAbortTaskDeletion, 34
uvCreateServiceTask, 34
uvDeleteSVCTask, 34
uvDeleteTask, 35
uvGetTaskFromName, 35
uvGetTaskFromRTOSHandle, 35
uvInvokeSCD, 36
uvKillTaskViolently, 36
uvRestartSVCTask, 36
uvScheduleTaskDeletion, 36
uvSendTaskStatusReport, 37
uvStartSVCTask, 37
uvStartTask, 37
uvSuspendSVCTask, 38
uvSuspendTask, 38
uvTaskCrashHandler, 39
uvTaskManager, 39
uvValidateManagedTasks, 39
state_change_daemon_args, 83
meta_task_handle, 83
State Engine, 10
state_change_queue
State Engine, 13
stator_leakage_inductance
motor_controller.h, 156
stator_resistance_ph_ph
motor_controller.h, 156
status
uv_init_task_response, 90
STM32_F407
uvfr_global_config.h, 200
STM32_H7xx
uvfr_global_config.h, 200
stm32f4xx_hal_conf.h
assert_param, 176
DATA_CACHE_ENABLE, 176
DP83848_PHY_ADDRESS, 176
ETH_RX_BUF_SIZE, 177
ETH_RXBUFNB, 177
ETH_TX_BUF_SIZE, 177
ETH_TXBUFNB, 177
EXTERNAL_CLOCK_VALUE, 177
HAL_ADC_MODULE_ENABLED, 177
HAL_CAN_MODULE_ENABLED, 178
HAL_CORTEX_MODULE_ENABLED, 178
HAL_DMA_MODULE_ENABLED, 178
HAL_EXTI_MODULE_ENABLED, 178
HAL_FLASH_MODULE_ENABLED, 178
HAL_GPIO_MODULE_ENABLED, 178
HAL_MODULE_ENABLED, 179
HAL_PWR_MODULE_ENABLED, 179
HAL_RCC_MODULE_ENABLED, 179
HAL_SPI_MODULE_ENABLED, 179
HAL_TIM_MODULE_ENABLED, 179
HSE_STARTUP_TIMEOUT, 179
HSE_VALUE, 180
HSI_VALUE, 180
INSTRUCTION_CACHE_ENABLE, 180
LSE_STARTUP_TIMEOUT, 180
LSE_VALUE, 180
LSI_VALUE, 181
MAC_ADDR0, 181
MAC_ADDR1, 181
MAC_ADDR2, 181

MAC_ADDR3, 181
MAC_ADDR4, 182
MAC_ADDR5, 182
PHY_AUTONEGO_COMPLETE, 182
PHY_AUTONEGOTIATION, 182
PHY_BCR, 182
PHY_BSR, 183
PHY_CONFIG_DELAY, 183
PHY_DUPLEX_STATUS, 183
PHY_FULLDUPLEX_100M, 183
PHY_FULLDUPLEX_10M, 183
PHY_HALFDUPLEX_100M, 184
PHY_HALFDUPLEX_10M, 184
PHY_ISOLATE, 184
PHY_JABBER_DETECTION, 184
PHY_LINKED_STATUS, 184
PHY_LOOPBACK, 185
PHY_POWERDOWN, 185
PHY_READ_TO, 185
PHY_RESET, 185
PHY_RESET_DELAY, 185
PHY_RESTART_AUTONEGOTIATION, 186
PHY_SPEED_STATUS, 186
PHY_SR, 186
PHY_WRITE_TO, 186
PREFETCH_ENABLE, 186
TICK_INT_PRIORITY, 187
USE_HAL_ADC_REGISTER_CALLBACKS, 187
USE_HAL_CAN_REGISTER_CALLBACKS, 187
USE_HAL_CEC_REGISTER_CALLBACKS, 187
USE_HAL_CRYP_REGISTER_CALLBACKS, 187
USE_HAL_DAC_REGISTER_CALLBACKS, 187
USE_HAL_DCMI_REGISTER_CALLBACKS, 188
USE_HAL_DFSDM_REGISTER_CALLBACKS, 188
USE_HAL_DMA2D_REGISTER_CALLBACKS, 188
USE_HAL_DSI_REGISTER_CALLBACKS, 188
USE_HAL_ETH_REGISTER_CALLBACKS, 188
USE_HAL_FMPI2C_REGISTER_CALLBACKS, 188
USE_HAL_FMPSMBUS_REGISTER_CALLBACKS, 189
USE_HAL_HASH_REGISTER_CALLBACKS, 189
USE_HAL_HCD_REGISTER_CALLBACKS, 189
USE_HAL_I2C_REGISTER_CALLBACKS, 189
USE_HAL_I2S_REGISTER_CALLBACKS, 189
USE_HAL_IRDA_REGISTER_CALLBACKS, 189
USE_HAL_LPTIM_REGISTER_CALLBACKS, 190
USE_HAL_LTDC_REGISTER_CALLBACKS, 190
USE_HAL_MMC_REGISTER_CALLBACKS, 190
USE_HAL_NAND_REGISTER_CALLBACKS, 190
USE_HAL_NOR_REGISTER_CALLBACKS, 190
USE_HAL_PCCARD_REGISTER_CALLBACKS, 190
USE_HAL_PCD_REGISTER_CALLBACKS, 191
USE_HAL_QSPI_REGISTER_CALLBACKS, 191
USE_HAL_RNG_REGISTER_CALLBACKS, 191
USE_HAL_RTC_REGISTER_CALLBACKS, 191
USE_HAL_SAI_REGISTER_CALLBACKS, 191
USE_HAL_SD_REGISTER_CALLBACKS, 191
USE_HAL_SDRAM_REGISTER_CALLBACKS, 192
USE_HAL_SMARTCARD_REGISTER_CALLBACKS, 192
USE_HAL_SMBUS_REGISTER_CALLBACKS, 192
USE_HAL_SPDIFRX_REGISTER_CALLBACKS, 192
USE_HAL_SPI_REGISTER_CALLBACKS, 192
USE_HAL_SRAM_REGISTER_CALLBACKS, 192
USE_HAL_TIM_REGISTER_CALLBACKS, 193
USE_HAL_UART_REGISTER_CALLBACKS, 193
USE_HAL_USART_REGISTER_CALLBACKS, 193
USE_HAL_WWDG_REGISTER_CALLBACKS, 193
USE_RTOS, 193
USE_SPI_CRC, 193
VDD_VALUE, 194
stm32f4xx_hal_msp.c
HAL_MspInit, 283
stm32f4xx_hal_timebase_tim.c
HAL_InitTick, 284
HAL_ResumeTick, 285
HAL_SuspendTick, 285
htim1, 286
stm32f4xx_it.c
BusFault_Handler, 287
CAN2_RX0_IRQHandler, 287
CAN2_RX1_IRQHandler, 288
CAN2_TX_IRQHandler, 288
DebugMon_Handler, 288
DMA2_Stream0_IRQHandler, 288
EXTI0_IRQHandler, 289
HardFault_Handler, 289
hcan2, 290
hdma_adc1, 290
htim1, 291
MemManage_Handler, 289
NMI_Handler, 289
TIM1_UP_TIM10_IRQHandler, 290
UsageFault_Handler, 290
stm32f4xx_it.h
BusFault_Handler, 195
CAN2_RX0_IRQHandler, 195
CAN2_RX1_IRQHandler, 195
CAN2_TX_IRQHandler, 195
DebugMon_Handler, 196
DMA2_Stream0_IRQHandler, 196
EXTI0_IRQHandler, 196
HardFault_Handler, 196
MemManage_Handler, 197
NMI_Handler, 197
TIM1_UP_TIM10_IRQHandler, 197
UsageFault_Handler, 197

Stm32f4xx_system, 51
STM32F4xx_System_Private_Defines, 54
STM32F4xx_System_Private_FunctionPrototypes, 57
STM32F4xx_System_Private_Functions, 58
SystemCoreClockUpdate, 58
SystemInit, 58
STM32F4xx_System_Private_Includes, 52
HSE_VALUE, 52
HSI_VALUE, 52
STM32F4xx_System_Private_Macros, 55
STM32F4xx_System_Private_TypesDefinitions, 53
STM32F4xx_System_Private_Variables, 56
AHBPrescTable, 56
APBPrescTable, 56
SystemCoreClock, 56
stopDaqSubTasks
daq.c, 238
suspendSelf
State Engine Internals, 33
suspension_states
uv_task_info, 98
svc_task_manager
State Engine, 13
svc_task_manager_period
uv_os_settings, 94
SVC_TASK_MAX_CHECKIN_PERIOD
uvfr_state_engine.h, 207
syscalls.c
__attribute__, 292
__io_getchar, 292
__io_putchar, 292
_close, 293
_execve, 293
_exit, 293
_fork, 293
_fstat, 293
_getpid, 294
_isatty, 294
_kill, 294
_link, 294
_lseek, 294
_open, 295
_stat, 295
_times, 295
_unlink, 295
_wait, 295
environ, 296
initialise_monitor_handles, 296
sysmem.c
__sbrk_heap_end, 298
_sbrk, 297
SystemClock_Config
main.c, 264
SystemCoreClock
STM32F4xx_System_Private_Variables, 56
SystemCoreClockUpdate
STM32F4xx_System_Private_Functions, 58
SystemInit
STM32F4xx_System_Private_Functions, 58
table_size
can.c, 230
task_args
uv_task_info, 98
task_flags
uv_task_info, 98
task_function
uv_task_info, 99
task_handle
task_management_info, 84
uv_task_info, 99
task_high_water_mark
task_status_block, 85
task_id
uv_task_info, 100
task_management_info, 84
parent_msg_queue, 84
State Engine API, 22
task_handle, 84
task_manager
State Engine, 13
task_manager_period
uv_os_settings, 94
task_name
uv_task_info, 100
task_name_lut
State Engine, 14
task_name_tree
State Engine, 14
task_period
uv_task_info, 100
task_priority
State Engine API, 22, 24
uv_task_info, 100
task_report_time
task_status_block, 85
task_rx_mailbox
uv_task_info, 101
task_state
uv_task_info, 101
task_status_block, 85
State Engine API, 22
task_high_water_mark, 85
task_report_time, 85
temp_monitoring.c
initTempMonitor, 299
tempMonitorTask, 300
testfunc, 300
testfunc2, 300
temp_monitoring.h
initTempMonitor, 198
tempMonitorTask, 198
temp_sensor_pt1
motor_controller.h, 159
temp_sensor_pt2
motor_controller.h, 159
temp_sensor_pt3

motor_controller.h, 159
temp_sensor_pt4
motor_controller.h, 159
Temperature
imd.h, 144
tempMonitorTask
temp_monitoring.c, 300
temp_monitoring.h, 198
testfunc
temp_monitoring.c, 300
testfunc2
temp_monitoring.c, 300
throttle_daq_to_preserve_performance
daq_loop_args, 65
TICK_INT_PRIORITY
stm32f4xx_hal_conf.h, 187
tim.c
HAL_TIM_Base_MspDeInit, 301
HAL_TIM_Base_MspInit, 302
htim3, 302
MX_TIM3_Init, 302
tim.h
htim3, 199
MX_TIM3_Init, 199
TIM1_UP_TIM10_IRQHandler
stm32f4xx_it.c, 290
stm32f4xx_it.h, 197
time_constant_rotor
motor_controller.h, 156
time_constant_stator
motor_controller.h, 156
time_sent
uv_task_msg_t, 103
tmi
uv_task_info, 101
todo1
motor_controller.h, 155
todo6969
motor_controller.h, 155
Touch_energy_fault
imd.h, 144
treenode
daq_child_task, 62
tripzone_glitch_detected
motor_controller.h, 158
true
Utility Macros, 47
Tx_msg_queue
can.c, 234
TxData
constants.c, 235
constants.h, 117
uvfr_utils.c, 313
TxHeader
constants.c, 236
constants.h, 117
TxMailbox
constants.c, 236
constants.h, 118
type
daq_datapoint, 63

Update_Batt_Temp
dash.c, 239
dash.h, 122
Update_RPM
dash.c, 239
dash.h, 122
Update_State_Of_Charge
dash.c, 240
dash.h, 123
updateRunningTasks
uvfr_state_engine.h, 208
Uptime_counter
imd.h, 144
UsageFault_Handler
stm32f4xx_it.c, 290
stm32f4xx_it.h, 197
use_default_settings
uv_init_struct, 88
USE_HAL_ADC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 187
USE_HAL_CAN_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 187
USE_HAL_CEC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 187
USE_HAL_CRYP_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 187
USE_HAL_DAC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 187
USE_HAL_DCMI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 188
USE_HAL_DFSDM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 188
USE_HAL_DMA2D_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 188
USE_HAL_DSI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 188
USE_HAL_ETH_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 188
USE_HAL_FMPI2C_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 188
USE_HAL_FMPSMBUS_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 189
USE_HAL_HASH_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 189
USE_HAL_HCD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 189
USE_HAL_I2C_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 189
USE_HAL_I2S_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 189
USE_HAL_IRDA_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 189
USE_HAL_LPTIM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 190
USE_HAL_LTDC_REGISTER_CALLBACKS

stm32f4xx_hal_conf.h, 190
USE_HAL_MMC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 190
USE_HAL_NAND_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 190
USE_HAL_NOR_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 190
USE_HAL_PCCARD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 190
USE_HAL_PCD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 191
USE_HAL_QSPI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 191
USE_HAL_RNG_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 191
USE_HAL_RTC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 191
USE_HAL_SAI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 191
USE_HAL_SD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 191
USE_HAL_SDRAM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 192
USE_HAL_SMARTCARD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 192
USE_HAL_SMBUS_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 192
USE_HAL_SPDIFRX_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 192
USE_HAL_SPI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 192
USE_HAL_SRAM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 192
USE_HAL_TIM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 193
USE_HAL_UART_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 193
USE_HAL_USART_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 193
USE_HAL_WWDG_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 193
USE_OLED_DEBUG
uvfr_utils.h, 212
USE_OS_MEM_MGMT
uvfr_global_config.h, 200
USE_RTOS
stm32f4xx_hal_conf.h, 193
USE_SPI_CRC
stm32f4xx_hal_conf.h, 193
Utility Macros, 42
_BV, 42
_BV_16, 42
_BV_32, 43
_BV_8, 43
deserializeBigE16, 43
deserializeBigE32, 43
deserializeSmallE16, 43
deserializeSmallE32, 44
endianSwap, 44
endianSwap16, 44
endianSwap32, 44
endianSwap8, 44
false, 45
isPowerOfTwo, 45
safePtrRead, 45
safePtrWrite, 45
serializeBigE16, 45
serializeBigE32, 46
serializeSmallE16, 46
serializeSmallE32, 46
setBits, 46
true, 47
UV19_PDU
uvfr_global_config.h, 201
UV_ABORTED
uvfr_utils.h, 218
UV_ASSIGN_TASK
uvfr_utils.h, 218
UV_BINARY_SEMAPHORE
uvfr_utils.h, 216
uv_binary_semaphore_info, 85
handle, 86
UV_BOOT
State Engine API, 26
UV_CAN1
uvfr_utils.h, 212
UV_CAN2
uvfr_utils.h, 212
UV_CAN_CHANNEL_MASK
uvfr_utils.h, 212
UV_CAN_DYNAMIC_MEM
uvfr_utils.h, 213
UV_CAN_EXTENDED_ID
uvfr_utils.h, 213
uv_CAN_msg, 86
can.h, 114
data, 86
dlc, 87
flags, 87
msg_id, 87
uvfr_utils.h, 214
UV_COMMAND_ACKNOWLEDGEMENT
uvfr_utils.h, 217
UV_COULDNT_DELETE
State Engine API, 24
UV_COULDNT_SUSPEND
State Engine API, 24
UV_DOUBLE
uvfr_utils.h, 216
UV_DRIVING
State Engine API, 26
uv_driving_mode_t
uvfr_utils.h, 217
UV_DUMB_FLAG
uvfr_utils.h, 216
UV_ERROR

uvfr_utils.h, 218
UV_ERROR_REPORT
uvfr_utils.h, 217
UV_ERROR_STATE
State Engine API, 26
uv_ext_device_id
uvfr_utils.h, 214
uv_external_device
uvfr_utils.h, 217
UV_FLOAT
uvfr_utils.h, 216
UV_HALT
State Engine API, 26
UV_INIT
State Engine API, 26
uv_init_struct, 88
use_default_settings, 88
uvfr_utils.h, 214
uv_init_task_args, 88
init_info_queue, 89
meta_task_handle, 89
specific_args, 89
uvfr_utils.h, 214
uv_init_task_response, 89
device, 90
errmsg, 90
nchar, 90
status, 90
uvfr_utils.h, 214
UV_INT16
uvfr_utils.h, 216
UV_INT32
uvfr_utils.h, 216
UV_INT64
uvfr_utils.h, 216
UV_INT8
uvfr_utils.h, 216
uv_internal_params, 91
e_code, 91
init_params, 91
peripheral_status, 92
uvfr_utils.h, 214
vehicle_settings, 92
UV_INVALID_MSG
uvfr_utils.h, 218
UV_KILL_CMD
State Engine API, 25
UV_LAUNCH_CONTROL
State Engine API, 26
UV_MALLOC_LIMIT
uvfr_global_config.h, 201
uv_msg_type
uvfr_utils.h, 215
uv_msg_type_t
uvfr_utils.h, 217
UV_MUTEX
uvfr_utils.h, 216
uv_mutex_info, 92
handle, 92
UV_NO_CMD
State Engine API, 25
UV_NONE
uvfr_utils.h, 216
UV_OK
uvfr_utils.h, 218
uv_os_settings, 93
max_svc_task_period, 93
max_task_period, 93
min_task_period, 94
State Engine API, 23
svc_task_manager_period, 94
task_manager_period, 94
UV_PARAM_READY
uvfr_utils.h, 218
UV_PARAM_REQUEST
uvfr_utils.h, 218
UV_RAW_DATA_TRANSFER
uvfr_utils.h, 218
UV_READY
State Engine API, 26
UV_SC_COMMAND
uvfr_utils.h, 218
uv_scd_response, 94
meta_id, 95
response_val, 95
State Engine API, 23
uv_scd_response_e
State Engine API, 24
UV_SEMAPHORE
uvfr_utils.h, 216
uv_semaphore_info, 95
handle, 95
uv_status
can.h, 114
uvfr_state_engine.h, 207
uvfr_utils.h, 215
uv_status_t
uvfr_utils.h, 218
UV_STRING
uvfr_utils.h, 216
UV_SUCCESSFUL_DELETION
State Engine API, 24
UV_SUCCESSFUL_SUSPENSION
State Engine API, 24
UV_SUSPEND_CMD
State Engine API, 25
UV_SUSPENDED
State Engine API, 26
UV_TASK_AWAITING_DELETION
State Engine API, 17
uv_task_cmd
State Engine API, 23
uvfr_utils.h, 215
uv_task_cmd_e
State Engine API, 25
UV_TASK_DEADLINE_FIRM

State Engine API, 17
UV_TASK_DEADLINE_HARD
State Engine API, 17
UV_TASK_DEADLINE_MASK
State Engine API, 17
UV_TASK_DEADLINE_NOT_ENFORCED
State Engine API, 17
UV_TASK_DEFER_DELETION
State Engine API, 18
UV_TASK_DELAYING
State Engine API, 18
UV_TASK_DELETE_COMMAND
uvfr_utils.h, 217
UV_TASK_DELETED
State Engine API, 25
UV_TASK_DORMANT_SVC
State Engine API, 18
UV_TASK_ERR_IN_CHILD
State Engine API, 18
UV_TASK_GENERIC_SVC
State Engine API, 18
uv_task_id
uvfr_state_engine.h, 207
uvfr_utils.h, 215
uv_task_info, 96
active_states, 97
cmd_data, 97
deletion_delay, 97
deletion_states, 97
parent, 97
stack_size, 98
State Engine API, 23
suspension_states, 98
task_args, 98
task_flags, 98
task_function, 99
task_handle, 99
task_id, 100
task_name, 100
task_period, 100
task_priority, 100
task_rx_mailbox, 101
task_state, 101
tmi, 101
UV_TASK_IS_CHILD
State Engine API, 18
UV_TASK_IS_ORPHAN
State Engine API, 19
UV_TASK_IS_PARENT
State Engine API, 19
UV_TASK_LOG_MEM_USAGE
State Engine API, 19
UV_TASK_LOG_START_STOP_TIME
State Engine API, 19
UV_TASK_MANAGER_MASK
State Engine API, 19
UV_TASK_MISSION_CRITICAL
State Engine API, 19
uv_task_msg
uvfr_utils.h, 215
uv_task_msg_t, 101
intended_recipient, 102
message_size, 102
message_type, 102
msg_contents, 102
sender, 103
time_sent, 103
UV_TASK_NOT_STARTED
State Engine API, 25
UV_TASK_PERIODIC_SVC
State Engine API, 20
UV_TASK_PRIO_INCREMENTATION
State Engine API, 20
UV_TASK_RUNNING
State Engine API, 25
UV_TASK_SCD_IGNORE
State Engine API, 20
UV_TASK_START_CMD
State Engine API, 25
UV_TASK_START_COMMAND
uvfr_utils.h, 217
uv_task_state_t
State Engine API, 25
uv_task_status
State Engine API, 23
UV_TASK_STATUS_REPORT
uvfr_utils.h, 217
UV_TASK_SUSPEND_COMMAND
uvfr_utils.h, 217
UV_TASK_SUSPENDED
State Engine API, 25
UV_TASK_VEHICLE_APPLICATION
State Engine API, 20
uv_timespan_ms
uvfr_state_engine.h, 207
uvfr_utils.h, 215
UV_UINT16
uvfr_utils.h, 216
UV_UINT32
uvfr_utils.h, 216
UV_UINT64
uvfr_utils.h, 216
UV_UINT8
uvfr_utils.h, 216
UV_UNSAFE_STATE
State Engine API, 24
UV_UTILS_SRC_IMPLIMENTATION
uvfr_utils.c, 308
uv_vehicle_settings, 103
bms_settings, 104
daq_settings, 104
driving_loop_settings, 104
imd_settings, 104
is_default, 104
mc_settings, 104
os_settings, 105

pdu_settings, 105
uvfr_settings.h, 202
uv_vehicle_state
State Engine API, 23
uv_vehicle_state_t
State Engine API, 25
UV_WAKEUP
uvfr_utils.h, 217
UV_WARNING
uvfr_utils.h, 218
uvAbortTaskDeletion
State Engine Internals, 34
uvCreateServiceTask
State Engine Internals, 34
uvCreateTask
State Engine API, 27
uvDeInitStateEngine
State Engine API, 27
uvDeleteSVCTask
State Engine Internals, 34
uvDeleteTask
State Engine Internals, 35
UVFR CANbus API, 49
insertCANMessageHandler, 49
uvSendCanMSG, 49
UVFR Utilities, 41
UVFR Vehicle Commands, 48
uvfr_global_config.h
ECUMASTER_PMU, 200
STM32_F407, 200
STM32_H7xx, 200
USE_OS_MEM_MGMT, 200
UV19_PDU, 201
UV_MALLOC_LIMIT, 201
uvfr_settings.c
current_vehicle_settings, 304
nukeSettings, 303
setupDefaultSettings, 303
SRC_UVFR_SETTINGS_C_, 303
uvSettingsInit, 304
uvSettingsProgrammerTask, 304
uvfr_settings.h
current_vehicle_settings, 203
ENABLE_FLASH_SETTINGS, 202
nukeSettings, 202
uv_vehicle_settings, 202
uvSettingsInit, 202
veh_gen_info, 202
uvfr_state_engine.c
UVFR_STATE_MACHINE_IMPLIMENTATION, 307
uvfr_state_engine.h
_LONGEST_SC_TIME, 206
_SC_DAEMON_PERIOD, 206
_UV_DEFAULT_TASK_INSTANCES, 206
_UV_DEFAULT_TASK_PERIOD, 206
_UV_DEFAULT_TASK_STACK_SIZE, 207
_UV_MIN_TASK_PERIOD, 207
getSVCTaskID, 208
SVC_TASK_MAX_CHECKIN_PERIOD, 207
updateRunningTasks, 208
uv_status, 207
uv_task_id, 207
uv_timespan_ms, 207
uvGetTaskById, 208
uvRegisterTask, 208
UVFR_STATE_MACHINE_IMPLIMENTATION
uvfr_state_engine.c, 307
uvfr_utils.c
__uvFreeCritSection, 308
__uvFreeOS, 308
__uvInitPanic, 308
__uvMallocCritSection, 309
__uvMallocOS, 309
init_task_handle, 313
reset_handle, 313
setup_extern_devices, 309
TxData, 313
UV_UTILS_SRC_IMPLIMENTATION, 308
uvInit, 309
uvIsPTRValid, 311
uvSysResetDaemon, 312
uvUtilsReset, 312
uvfr_utils.h
__uvInitPanic, 218
accel, 217
access_control_info, 213
access_control_t, 216
access_control_type, 213
BMS, 217
bool, 213
data_type, 216
econ, 217
global_context, 221
IMD, 217
INIT_CHECK_PERIOD, 212
limp, 217
MAX_INIT_TIME, 212
MOTOR_CONTROLLER, 217
normal, 217
p_status, 213
PDU, 217
USE_OLED_DEBUG, 212
UV_ABORTED, 218
UV_ASSIGN_TASK, 218
UV_BINARY_SEMAPHORE, 216
UV_CAN1, 212
UV_CAN2, 212
UV_CAN_CHANNEL_MASK, 212
UV_CAN_DYNAMIC_MEM, 213
UV_CAN_EXTENDED_ID, 213
uv_CAN_msg, 214
UV_COMMAND_ACKNOWLEDGEMENT, 217
UV_DOUBLE, 216
uv_driving_mode_t, 217
UV_DUMB_FLAG, 216
UV_ERROR, 218

UV_ERROR_REPORT, 217
uv_ext_device_id, 214
uv_external_device, 217
UV_FLOAT, 216
uv_init_struct, 214
uv_init_task_args, 214
uv_init_task_response, 214
UV_INT16, 216
UV_INT32, 216
UV_INT64, 216
UV_INT8, 216
uv_internal_params, 214
UV_INVALID_MSG, 218
uv_msg_type, 215
uv_msg_type_t, 217
UV_MUTEX, 216
UV_NONE, 216
UV_OK, 218
UV_PARAM_READY, 218
UV_PARAM_REQUEST, 218
UV_RAW_DATA_TRANSFER, 218
UV_SC_COMMAND, 218
UV_SEMAPHORE, 216
uv_status, 215
uv_status_t, 218
UV_STRING, 216
uv_task_cmd, 215
UV_TASK_DELETE_COMMAND, 217
uv_task_id, 215
uv_task_msg, 215
UV_TASK_START_COMMAND, 217
UV_TASK_STATUS_REPORT, 217
UV_TASK_SUSPEND_COMMAND, 217
uv_timespan_ms, 215
UV_UINT16, 216
UV_UINT32, 216
UV_UINT64, 216
UV_UINT8, 216
UV_WAKEUP, 217
UV_WARNING, 218
uvInit, 218
uvIsPTRValid, 220
uvfr_vehicle_commands.c
uvSecureVehicle, 313
uvfr_vehicle_commands.h
_uvCloseSDC_canBased, 223
_uvHonkHorn_canBased, 223
_uvOpenSDC_canBased, 224
_uvSilenceHorn_canBased, 224
_uvStartCoolantPump_canBased, 224
_uvStopCoolantPump_canBased, 224
uvHonkHorn, 222
uvOpenSDC, 222
uvSecureVehicle, 224
uvSilenceHorn, 222
uvStartCoolantPump, 223
uvStartFans, 223
uvStopCoolantPump, 223
uvStopFans, 223
uvGetTaskById
uvfr_state_engine.h, 208
uvGetTaskFromName
State Engine Internals, 35
uvGetTaskFromRTOSHandle
State Engine Internals, 35
uvHonkHorn
uvfr_vehicle_commands.h, 222
uvInit
uvfr_utils.c, 309
uvfr_utils.h, 218
uvInitStateEngine
State Engine API, 27
uvInvokeSCD
State Engine Internals, 36
uvIsPTRValid
uvfr_utils.c, 311
uvfr_utils.h, 220
uvKillTaskViolently
State Engine Internals, 36
uvOpenSDC
uvfr_vehicle_commands.h, 222
uvRegisterTask
uvfr_state_engine.h, 208
uvRestartSVCTask
State Engine Internals, 36
uvScheduleTaskDeletion
State Engine Internals, 36
uvSecureVehicle
uvfr_vehicle_commands.c, 313
uvfr_vehicle_commands.h, 224
uvSendCanMSG
UVFR CANbus API, 49
uvSendTaskStatusReport
State Engine Internals, 37
uvSettingsInit
uvfr_settings.c, 304
uvfr_settings.h, 202
uvSettingsProgrammerTask
uvfr_settings.c, 304
uvSilenceHorn
uvfr_vehicle_commands.h, 222
uvStartCoolantPump
uvfr_vehicle_commands.h, 223
uvStartFans
uvfr_vehicle_commands.h, 223
uvStartStateMachine
State Engine API, 27
uvStartSVCTask
State Engine Internals, 37
uvStartTask
State Engine Internals, 37
uvStopCoolantPump
uvfr_vehicle_commands.h, 223
uvStopFans
uvfr_vehicle_commands.h, 223
uvSuspendSVCTask

State Engine Internals, 38
uvSuspendTask
State Engine Internals, 38
uvSVCTaskManager
State Engine, 11
uvSysResetDaemon
uvfr_utils.c, 312
uvTaskCrashHandler
State Engine Internals, 39
uvTaskDelay
State Engine API, 20
uvTaskDelayUntil
State Engine API, 21
uvTaskIsDelaying
State Engine API, 21
uvTaskManager
State Engine Internals, 39
uvTaskResetDelayBit
State Engine API, 21
uvTaskResetDeletionBit
State Engine API, 21
uvTaskSetDelayBit
State Engine API, 22
uvTaskSetDeletionBit
State Engine API, 22
uvUtilsReset
uvfr_utils.c, 312
uvValidateManagedTasks
State Engine Internals, 39

vApplicationGetIdleTaskMemory
freertos.c, 245
vApplicationGetTimerTaskMemory
freertos.c, 245
vApplicationIdleHook
freertos.c, 246
vApplicationMallocFailedHook
freertos.c, 246
vApplicationStackOverflowHook
freertos.c, 246
vApplicationTickHook
freertos.c, 246
Vb_hi_res
imd.h, 143
VDD_VALUE
stm32f4xx_hal_conf.h, 194
veh_gen_info, 105
uvfr_settings.h, 202
vehicle_settings
uv_internal_params, 92
vehicle_state
State Engine, 14
Version_0
imd.h, 143
Version_1
imd.h, 143
Version_2
imd.h, 143
Vexc_hi_res
imd.h, 143
Vn_hi_res
imd.h, 143
voltages_Vp_and_Vn
imd.h, 144
Vout_saturation_max_limit
motor_controller.h, 158
Vp_hi_res
imd.h, 143
vPortSVCHandler
FreeRTOSConfig.h, 140
Vpwr_hi_res
imd.h, 143

wait
oled.h, 162
WaitFor_CAN_Response
motor_controller.c, 270
warning_5
motor_controller.h, 158
warning_9
motor_controller.h, 158
watchdog_reset
motor_controller.h, 158

xIdleStack
freertos.c, 247
xIdleTaskTCBBuffer
freertos.c, 247
xPortPendSVHandler
FreeRTOSConfig.h, 140
xPortSysTickHandler
FreeRTOSConfig.h, 140
xTimerStack
freertos.c, 248
xTimerTaskTCBBuffer
freertos.c, 248