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 . . . 11

5.1.4 Function Documentation . . . 11

5.1.4.1 compareTaskByName() . . . 11

5.1.4.2 uvSVCTaskManager() . . . 11

5.1.4.3 uvTaskPeriodEnd() . . . 12

5.1.5 Variable Documentation . . . 12

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 . . . 13

5.1.5.5 error_bitfield . . . 13

5.1.5.6 last_task_end_times . . . 13

5.1.5.7 last_task_start_times . . . 13

5.1.5.8 nc_throttling . . . 13

5.1.5.9 os_settings . . . 14

5.1.5.10 previous_state . . . 14

5.1.5.11 SCD_active . . . 14

5.1.5.12 scd_handle_ptr . . . 14

5.1.5.13 state_change_queue . . . 14

5.1.5.14 svc_task_manager . . . 15

5.1.5.15 task_manager . . . 15

5.1.5.16 task_name_lut . . . 15

5.1.5.17 task_name_tree . . . 15

5.1.5.18 task_tardiness . . . 15

5.1.5.19 throttle_daq . . . 15

5.1.5.20 vehicle_state . . . 15

5.2 State Engine API . . . 16

5.2.1 Detailed Description . . . 18
5.2.2 Macro Definition Documentation . . . 18
5.2.2.1 UV_TASK_AWAITING_DELETION . . . 18
5.2.2.2 UV_TASK_DEADLINE_FIRM . . . 18
5.2.2.3 UV_TASK_DEADLINE_HARD . . . 18
5.2.2.4 UV_TASK_DEADLINE_MASK . . . 19
5.2.2.5 UV_TASK_DEADLINE_NOT_ENFORCED . . . 19
5.2.2.6 UV_TASK_DEFER_DELETION . . . 19
5.2.2.7 UV_TASK_DELAYING . . . 19
5.2.2.8 UV_TASK_DORMANT_SVC . . . 19
5.2.2.9 UV_TASK_ERR_IN_CHILD . . . 19
5.2.2.10 UV_TASK_GENERIC_SVC . . . 20
5.2.2.11 UV_TASK_IS_CHILD . . . 20
5.2.2.12 UV_TASK_IS_ORPHAN . . . 20
5.2.2.13 UV_TASK_IS_PARENT . . . 20
5.2.2.14 UV_TASK_LOG_MEM_USAGE . . . 20
5.2.2.15 UV_TASK_LOG_START_STOP_TIME . . . 20
5.2.2.16 UV_TASK_MANAGER_MASK . . . 21
5.2.2.17 UV_TASK_MISSION_CRITICAL . . . 21
5.2.2.18 UV_TASK_PERIODIC_SVC . . . 21
5.2.2.19 UV_TASK_PRIO_INCREMENTATION . . . 21
5.2.2.20 UV_TASK_SCD_IGNORE . . . 21
5.2.2.21 UV_TASK_VEHICLE_APPLICATION . . . 21
5.2.2.22 uvTaskDelay . . . 21
5.2.2.23 uvTaskDelayUntil . . . 22
5.2.2.24 uvTaskIsDelaying . . . 22
5.2.2.25 uvTaskResetDelayBit . . . 22
5.2.2.26 uvTaskResetDeletionBit . . . 23
5.2.2.27 uvTaskSetDelayBit . . . 23
5.2.2.28 uvTaskSetDeletionBit . . . 23
5.2.3 Typedef Documentation . . . 23
5.2.3.1 task_management_info . . . 23
5.2.3.2 task_priority . . . 23
5.2.3.3 task_status_block . . . 24
5.2.3.4 uv_os_settings . . . 24
5.2.3.5 uv_scd_response . . . 24
5.2.3.6 uv_task_cmd . . . 24
5.2.3.7 uv_task_info . . . 24
5.2.3.8 uv_task_status . . . 24
5.2.3.9 uv_vehicle_state . . . 25
5.2.4 Enumeration Type Documentation . . . 25
5.2.4.1 os_flag . . . 25

5.2.4.2 task_priority . . . 25
5.2.4.3 uv_scd_response_e . . . 26
5.2.4.4 uv_task_cmd_e . . . 26
5.2.4.5 uv_task_state_t . . . 26
5.2.4.6 uv_vehicle_state_t . . . 27
5.2.5 Function Documentation . . . 27
5.2.5.1 changeVehicleState() . . . 27
5.2.5.2 uvCreateTask() . . . 28
5.2.5.3 uvDeInitStateEngine() . . . 28
5.2.5.4 uvInitStateEngine() . . . 29
5.2.5.5 uvStartStateMachine() . . . 29
5.2.5.6 uvTaskPeriodEnd() . . . 29
5.3 State Engine Internals . . . 30
5.3.1 Detailed Description . . . 31
5.3.2 Function Documentation . . . 31
5.3.2.1 __uvPanic() . . . 31
5.3.2.2 _stateChangeDaemon() . . . 31
5.3.2.3 _uvValidateSpecificTask() . . . 33
5.3.2.4 addTaskToTaskRegister() . . . 33
5.3.2.5 killEmAll() . . . 33
5.3.2.6 killSelf() . . . 34
5.3.2.7 proccessSCDMsg() . . . 34
5.3.2.8 suspendSelf() . . . 35
5.3.2.9 uvAbortTaskDeletion() . . . 35
5.3.2.10 uvCreateServiceTask() . . . 35
5.3.2.11 uvDeleteSVCTask() . . . 36
5.3.2.12 uvDeleteTask() . . . 36
5.3.2.13 uvGetTaskFromName() . . . 36
5.3.2.14 uvGetTaskFromRTOSHandle() . . . 36
5.3.2.15 uvKillTaskViolently() . . . 37
5.3.2.16 uvLateTaskHandler() . . . 37
5.3.2.17 uvRestartSVCTask() . . . 38
5.3.2.18 uvScheduleTaskDeletion() . . . 38
5.3.2.19 uvSendTaskStatusReport() . . . 38
5.3.2.20 uvStartSVCTask() . . . 38
5.3.2.21 uvStartTask() . . . 39
5.3.2.22 uvSuspendSVCTask() . . . 39
5.3.2.23 uvSuspendTask() . . . 39
5.3.2.24 uvTaskCrashHandler() . . . 40
5.3.2.25 uvTaskManager() . . . 40
5.3.2.26 uvThrottleNonCritTasks() . . . 41
5.3.2.27 uvValidateManagedTasks() . . . 41

5.4 UVFR Utilities . . . 42
5.4.1 Detailed Description . . . 42
5.5 Utility Macros . . . 43
5.5.1 Detailed Description . . . 43
5.5.2 Macro Definition Documentation . . . 43
5.5.2.1 _BV . . . 43
5.5.2.2 _BV_16 . . . 44
5.5.2.3 _BV_32 . . . 44
5.5.2.4 _BV_8 . . . 44
5.5.2.5 deserializeBigE16 . . . 44
5.5.2.6 deserializeBigE32 . . . 44
5.5.2.7 deserializeSmallE16 . . . 45
5.5.2.8 deserializeSmallE32 . . . 45
5.5.2.9 endianSwap . . . 45
5.5.2.10 endianSwap16 . . . 45
5.5.2.11 endianSwap32 . . . 45
5.5.2.12 endianSwap8 . . . 46
5.5.2.13 false . . . 46
5.5.2.14 isPowerOfTwo . . . 46
5.5.2.15 safePtrRead . . . 46
5.5.2.16 safePtrWrite . . . 46
5.5.2.17 serializeBigE16 . . . 47
5.5.2.18 serializeBigE32 . . . 47
5.5.2.19 serializeSmallE16 . . . 47
5.5.2.20 serializeSmallE32 . . . 47
5.5.2.21 setBits . . . 47
5.5.2.22 true . . . 48
5.6 UVFR Vehicle Commands . . . 49
5.7 UVFR CANbus API . . . 50
5.7.1 Detailed Description . . . 50
5.7.2 Function Documentation . . . 50
5.7.2.1 insertCANMessageHandler() . . . 50
5.7.2.2 uvSendCanMSG() . . . 50
5.8 CMSIS . . . 51
5.8.1 Detailed Description . . . 51
5.9 Stm32f4xx_system . . . 52
5.9.1 Detailed Description . . . 52
5.10 STM32F4xx_System_Private_Includes . . . 53
5.10.1 Detailed Description . . . 53
5.10.2 Macro Definition Documentation . . . 53
5.10.2.1 HSE_VALUE . . . 53
5.10.2.2 HSI_VALUE . . . 53

5.11 STM32F4xx_System_Private_TypesDefinitions . . . 54
5.12 STM32F4xx_System_Private_Defines . . . 55
5.13 STM32F4xx_System_Private_Macros . . . 56
5.14 STM32F4xx_System_Private_Variables . . . 57
5.14.1 Detailed Description . . . 57
5.14.2 Variable Documentation . . . 57
5.14.2.1 AHBPrescTable . . . 57
5.14.2.2 APBPrescTable . . . 57
5.14.2.3 SystemCoreClock . . . 57
5.15 STM32F4xx_System_Private_FunctionPrototypes . . . 58
5.16 STM32F4xx_System_Private_Functions . . . 59
5.16.1 Detailed Description . . . 59
5.16.2 Function Documentation . . . 59
5.16.2.1 SystemCoreClockUpdate() . . . 59
5.16.2.2 SystemInit() . . . 59

6 Data Structure Documentation **61**

6.1 access_control_info Union Reference . . . 61
6.1.1 Detailed Description . . . 61
6.1.2 Field Documentation . . . 61
6.1.2.1 bin_semaphore . . . 61
6.1.2.2 mutex . . . 61
6.1.2.3 semaphore . . . 62
6.2 bms_settings_t Struct Reference . . . 62
6.2.1 Detailed Description . . . 62
6.2.2 Field Documentation . . . 62
6.2.2.1 BMS_CAN_timeout . . . 62
6.2.2.2 max_temp . . . 62
6.3 CAN_Callback Struct Reference . . . 63
6.3.1 Detailed Description . . . 63
6.3.2 Field Documentation . . . 63
6.3.2.1 CAN_id . . . 63
6.3.2.2 function . . . 63
6.3.2.3 next . . . 63
6.4 daq_child_task Struct Reference . . . 64
6.4.1 Detailed Description . . . 64
6.4.2 Field Documentation . . . 64
6.4.2.1 meta_task_handle . . . 64
6.4.2.2 next_task . . . 64
6.4.2.3 param_list . . . 64
6.4.2.4 period . . . 65
6.5 daq_datapoint Struct Reference . . . 65

6.5.1 Detailed Description . . . 65
6.5.2 Field Documentation . . . 65
6.5.2.1 can_id . . . 65
6.5.2.2 param . . . 66
6.5.2.3 period . . . 66
6.5.2.4 type . . . 66
6.6 daq_loop_args Struct Reference . . . 66
6.6.1 Detailed Description . . . 67
6.6.2 Field Documentation . . . 67
6.6.2.1 can_channel . . . 67
6.6.2.2 daq_child_priority . . . 67
6.6.2.3 minimum_daq_period . . . 67
6.6.2.4 throttle_daq_to_preserve_performance . . . 67
6.6.2.5 total_params_logged . . . 68
6.7 daq_param_list_node Struct Reference . . . 68
6.7.1 Detailed Description . . . 68
6.7.2 Field Documentation . . . 68
6.7.2.1 can_id . . . 68
6.7.2.2 next . . . 68
6.7.2.3 param . . . 69
6.7.2.4 size . . . 69
6.8 driving_loop_args Struct Reference . . . 69
6.8.1 Detailed Description . . . 70
6.8.2 Field Documentation . . . 70
6.8.2.1 absolute_max_acc_pwr . . . 70
6.8.2.2 absolute_max_accum_current . . . 70
6.8.2.3 absolute_max_motor_rpm . . . 70
6.8.2.4 absolute_max_motor_torque . . . 70
6.8.2.5 accum_regen_soc_threshold . . . 71
6.8.2.6 apps_bottom . . . 71
6.8.2.7 apps_implausibility_recovery_threshold . . . 71
6.8.2.8 apps_plausibility_check_threshold . . . 71
6.8.2.9 apps_top . . . 71
6.8.2.10 bps_implausibility_recovery_threshold . . . 72
6.8.2.11 bps_plausibility_check_threshold . . . 72
6.8.2.12 dmodes . . . 72
6.8.2.13 max_accum_current_5s . . . 72
6.8.2.14 max_apps1_value . . . 72
6.8.2.15 max_apps2_value . . . 73
6.8.2.16 max_apps_offset . . . 73
6.8.2.17 max_BPS_value . . . 73
6.8.2.18 min_apps1_value . . . 73

6.8.2.19 min_apps2_value . . . 73
6.8.2.20 min_apps_offset . . . 74
6.8.2.21 min_apps_value . . . 74
6.8.2.22 min_BPS_value . . . 74
6.8.2.23 num_driving_modes . . . 74
6.8.2.24 period . . . 74
6.8.2.25 regen_rpm_cutoff . . . 75
6.9 drivingLoopArgs Struct Reference . . . 75
6.9.1 Detailed Description . . . 75
6.10 drivingMode Struct Reference . . . 75
6.10.1 Detailed Description . . . 76
6.10.2 Field Documentation . . . 76
6.10.2.1 control_map_fn . . . 76
6.10.2.2 dm_name . . . 76
6.10.2.3 flags . . . 76
6.10.2.4 map_fn_params . . . 76
6.10.2.5 max_acc_pwr . . . 77
6.10.2.6 max_current . . . 77
6.10.2.7 max_motor_torque . . . 77
6.11 drivingModeParams Union Reference . . . 77
6.11.1 Detailed Description . . . 77
6.12 exp_torque_map_args Struct Reference . . . 77
6.12.1 Detailed Description . . . 78
6.12.2 Field Documentation . . . 78
6.12.2.1 gamma . . . 78
6.12.2.2 offset . . . 78
6.13 helper_task_args Union Reference . . . 78
6.13.1 Detailed Description . . . 78
6.13.2 Field Documentation . . . 79
6.13.2.1 journal_tx_args . . . 79
6.13.2.2 setting_tx_args . . . 79
6.14 linear_torque_map_args Struct Reference . . . 79
6.14.1 Detailed Description . . . 79
6.14.2 Field Documentation . . . 79
6.14.2.1 offset . . . 80
6.14.2.2 slope . . . 80
6.15 motor_controller_settings Struct Reference . . . 80
6.15.1 Detailed Description . . . 80
6.15.2 Field Documentation . . . 80
6.15.2.1 can_id_rx . . . 80
6.15.2.2 can_id_tx . . . 81
6.15.2.3 integral_memory_max . . . 81

6.15.2.4 integral_time_constant . . . 81
6.15.2.5 mc_CAN_timeout . . . 81
6.15.2.6 proportional_gain . . . 81
6.16 p_status Struct Reference . . . 81
6.16.1 Detailed Description . . . 82
6.16.2 Field Documentation . . . 82
6.16.2.1 activation_time . . . 82
6.16.2.2 peripheral_status . . . 82
6.17 rbnode Struct Reference . . . 82
6.17.1 Detailed Description . . . 83
6.17.2 Field Documentation . . . 83
6.17.2.1 color . . . 83
6.17.2.2 data . . . 83
6.17.2.3 left . . . 83
6.17.2.4 parent . . . 84
6.17.2.5 right . . . 84
6.18 rbtree Struct Reference . . . 84
6.18.1 Detailed Description . . . 84
6.18.2 Field Documentation . . . 85
6.18.2.1 compare . . . 85
6.18.2.2 count . . . 85
6.18.2.3 destroy . . . 85
6.18.2.4 min . . . 85
6.18.2.5 nil . . . 86
6.18.2.6 print . . . 86
6.18.2.7 root . . . 86
6.19 s_curve_torque_map_args Struct Reference . . . 86
6.19.1 Detailed Description . . . 87
6.19.2 Field Documentation . . . 87
6.19.2.1 a . . . 87
6.19.2.2 b . . . 87
6.19.2.3 c . . . 87
6.20 setting_helper_task Struct Reference . . . 87
6.20.1 Detailed Description . . . 88
6.20.2 Field Documentation . . . 88
6.20.2.1 args . . . 88
6.20.2.2 meta_task_handle . . . 88
6.20.2.3 status . . . 88
6.21 state_change_daemon_args Struct Reference . . . 88
6.21.1 Detailed Description . . . 89
6.21.2 Field Documentation . . . 89
6.21.2.1 meta_task_handle . . . 89

6.22 task_management_info Struct Reference . . . 89
6.22.1 Detailed Description . . . 89
6.22.2 Field Documentation . . . 89
6.22.2.1 parent_msg_queue . . . 90
6.22.2.2 task_handle . . . 90
6.23 task_status_block Struct Reference . . . 90
6.23.1 Detailed Description . . . 90
6.23.2 Field Documentation . . . 90
6.23.2.1 task_high_water_mark . . . 91
6.23.2.2 task_report_time . . . 91
6.24 tx_all_settings_args Struct Reference . . . 91
6.24.1 Detailed Description . . . 91
6.24.2 Field Documentation . . . 91
6.24.2.1 sblock_origin . . . 91
6.25 tx_journal_args Struct Reference . . . 92
6.25.1 Detailed Description . . . 92
6.25.2 Field Documentation . . . 92
6.25.2.1 end_time . . . 92
6.25.2.2 start_time . . . 92
6.26 uv19_pdu_settings Struct Reference . . . 92
6.26.1 Detailed Description . . . 93
6.26.2 Field Documentation . . . 93
6.26.2.1 brake_light_channel . . . 93
6.26.2.2 buzzer_channel . . . 93
6.26.2.3 coolant_pump_channel . . . 93
6.26.2.4 fan1_channel . . . 93
6.26.2.5 fan2_channel . . . 93
6.26.2.6 mc_rfe_channel . . . 94
6.26.2.7 PDU_rx_addr . . . 94
6.26.2.8 PDU_tx_addr . . . 94
6.26.2.9 sdc_channel . . . 94
6.27 uv_binary_semaphore_info Struct Reference . . . 94
6.27.1 Detailed Description . . . 94
6.27.2 Field Documentation . . . 95
6.27.2.1 handle . . . 95
6.28 uv_CAN_msg Struct Reference . . . 95
6.28.1 Detailed Description . . . 95
6.28.2 Field Documentation . . . 95
6.28.2.1 data . . . 95
6.28.2.2 dlc . . . 96
6.28.2.3 flags . . . 96
6.28.2.4 msg_id . . . 96

6.29 uv_imd_settings Struct Reference . . . 96
6.29.1 Detailed Description . . . 97
6.29.2 Field Documentation . . . 97
6.29.2.1 expected_isolation_capacitances . . . 97
6.29.2.2 max_imd_temperature . . . 97
6.29.2.3 min_isolation_resistances . . . 97
6.30 uv_init_struct Struct Reference . . . 97
6.30.1 Detailed Description . . . 98
6.30.2 Field Documentation . . . 98
6.30.2.1 use_default_settings . . . 98
6.31 uv_init_task_args Struct Reference . . . 98
6.31.1 Detailed Description . . . 98
6.31.2 Field Documentation . . . 99
6.31.2.1 init_info_queue . . . 99
6.31.2.2 meta_task_handle . . . 99
6.31.2.3 specific_args . . . 99
6.32 uv_init_task_response Struct Reference . . . 99
6.32.1 Detailed Description . . . 100
6.32.2 Field Documentation . . . 100
6.32.2.1 device . . . 100
6.32.2.2 errmsg . . . 100
6.32.2.3 nchar . . . 100
6.32.2.4 status . . . 100
6.33 uv_internal_params Struct Reference . . . 101
6.33.1 Detailed Description . . . 101
6.33.2 Field Documentation . . . 101
6.33.2.1 e_code . . . 101
6.33.2.2 init_params . . . 101
6.33.2.3 peripheral_status . . . 101
6.33.2.4 vehicle_settings . . . 102
6.34 uv_mutex_info Struct Reference . . . 102
6.34.1 Detailed Description . . . 102
6.34.2 Field Documentation . . . 102
6.34.2.1 handle . . . 102
6.35 uv_os_settings Struct Reference . . . 102
6.35.1 Detailed Description . . . 103
6.35.2 Field Documentation . . . 103
6.35.2.1 max_svc_task_period . . . 103
6.35.2.2 max_task_period . . . 103
6.35.2.3 min_task_period . . . 103
6.35.2.4 os_flags . . . 104
6.35.2.5 svc_task_manager_period . . . 104

6.35.2.6 task_manager_period . . . 104
6.35.2.7 task_overshoot_margin_crit . . . 104
6.35.2.8 task_overshoot_margin_noncrit . . . 104
6.35.2.9 task_throttle_increment . . . 104
6.36 uv_persistant_data_frame Struct Reference . . . 105
6.36.1 Detailed Description . . . 105
6.36.2 Field Documentation . . . 105
6.36.2.1 total_distance_cm . . . 105
6.36.2.2 total_time_driving . . . 105
6.36.2.3 total_vehicle_uptime . . . 105
6.37 uv_scd_response Struct Reference . . . 106
6.37.1 Detailed Description . . . 106
6.37.2 Field Documentation . . . 106
6.37.2.1 meta_id . . . 106
6.37.2.2 response_val . . . 106
6.38 uv_semaphore_info Struct Reference . . . 106
6.38.1 Detailed Description . . . 107
6.38.2 Field Documentation . . . 107
6.38.2.1 handle . . . 107
6.39 uv_task_info Struct Reference . . . 107
6.39.1 Detailed Description . . . 108
6.39.2 Field Documentation . . . 108
6.39.2.1 active_states . . . 108
6.39.2.2 cmd_data . . . 108
6.39.2.3 deletion_delay . . . 108
6.39.2.4 deletion_states . . . 109
6.39.2.5 last_execution_time . . . 109
6.39.2.6 parent . . . 109
6.39.2.7 stack_size . . . 109
6.39.2.8 suspension_states . . . 110
6.39.2.9 task_args . . . 110
6.39.2.10 task_flags . . . 110
6.39.2.11 task_function . . . 111
6.39.2.12 task_handle . . . 111
6.39.2.13 task_id . . . 111
6.39.2.14 task_name . . . 111
6.39.2.15 task_period . . . 112
6.39.2.16 task_priority . . . 112
6.39.2.17 task_rx_mailbox . . . 112
6.39.2.18 task_state . . . 112
6.39.2.19 throttle_factor . . . 113
6.39.2.20 tmi . . . 113

6.40 uv_task_msg_t Struct Reference . . . 113
6.40.1 Detailed Description . . . 113
6.40.2 Field Documentation . . . 114
6.40.2.1 intended_recipient . . . 114
6.40.2.2 message_size . . . 114
6.40.2.3 message_type . . . 114
6.40.2.4 msg_contents . . . 114
6.40.2.5 sender . . . 114
6.40.2.6 time_sent . . . 115
6.41 uv_vehicle_settings Struct Reference . . . 115
6.41.1 Detailed Description . . . 115
6.41.2 Field Documentation . . . 115
6.41.2.1 bms_settings . . . 115
6.41.2.2 daq_param_list . . . 116
6.41.2.3 daq_settings . . . 116
6.41.2.4 driving_loop_settings . . . 116
6.41.2.5 flags . . . 116
6.41.2.6 imd_settings . . . 116
6.41.2.7 mc_settings . . . 117
6.41.2.8 os_settings . . . 117
6.41.2.9 pdu_settings . . . 117
6.41.2.10 veh_info . . . 117
6.42 veh_gen_info Struct Reference . . . 117
6.42.1 Detailed Description . . . 118
6.42.2 Field Documentation . . . 118
6.42.2.1 drive_ratio . . . 118
6.42.2.2 test1 . . . 118
6.42.2.3 test2 . . . 118
6.42.2.4 test3 . . . 118
6.42.2.5 test4 . . . 119
6.42.2.6 test5 . . . 119
6.42.2.7 test6 . . . 119
6.42.2.8 wheel_size . . . 119

7 File Documentation **121**

7.1 Core/Inc/adc.h File Reference . . . 121
7.1.1 Detailed Description . . . 122
7.1.2 Macro Definition Documentation . . . 122
7.1.2.1 ADC1_BUF_LEN . . . 122
7.1.2.2 ADC1_CHNL_CNT . . . 122
7.1.2.3 ADC1_MAX_VOLT . . . 122
7.1.2.4 ADC1_MIN_VOLT . . . 122

7.1.2.5 ADC1_SAMPLES . . . 123
7.1.2.6 ADC2_BUF_LEN . . . 123
7.1.2.7 ADC2_CHNL_CNT . . . 123
7.1.2.8 ADC2_MAX_VOLT . . . 123
7.1.2.9 ADC2_MIN_VOLT . . . 123
7.1.2.10 ADC2_SAMPLES . . . 123
7.1.3 Function Documentation . . . 124
7.1.3.1 initADCTask() . . . 124
7.1.3.2 MX_ADC1_Init() . . . 124
7.1.3.3 MX_ADC2_Init() . . . 124
7.1.3.4 processADCBuffer() . . . 125
7.1.4 Variable Documentation . . . 125
7.1.4.1 hadc1 . . . 125
7.1.4.2 hadc2 . . . 125
7.2 Core/Inc/bms.h File Reference . . . 125
7.2.1 Macro Definition Documentation . . . 126
7.2.1.1 DEFAULT_BMS_CAN_TIMEOUT . . . 126
7.2.2 Typedef Documentation . . . 126
7.2.2.1 bms_settings_t . . . 126
7.2.3 Function Documentation . . . 126
7.2.3.1 BMS_Init() . . . 126
7.3 Core/Inc/can.h File Reference . . . 127
7.3.1 Detailed Description . . . 127
7.3.2 Macro Definition Documentation . . . 128
7.3.2.1 CAN_RX_DAEMON_NAME . . . 128
7.3.2.2 CAN_TX_DAEMON_NAME . . . 128
7.3.3 Typedef Documentation . . . 128
7.3.3.1 uv_CAN_msg . . . 128
7.3.3.2 uv_status . . . 128
7.3.4 Function Documentation . . . 128
7.3.4.1 CANbusRxSvcDaemon() . . . 129
7.3.4.2 CANbusTxSvcDaemon() . . . 129
7.3.4.3 HAL_CAN_RxFifo0MsgPendingCallback() . . . 129
7.3.4.4 HAL_CAN_RxFifo1MsgPendingCallback() . . . 130
7.3.4.5 MX_CAN2_Init() . . . 130
7.3.4.6 nuke_hash_table() . . . 130
7.3.5 Variable Documentation . . . 130
7.3.5.1 hcan2 . . . 130
7.4 Core/Inc/constants.h File Reference . . . 131
7.4.1 Enumeration Type Documentation . . . 131
7.4.1.1 CAN_IDs . . . 131
7.4.2 Variable Documentation . . . 131

7.4.2.1 RxData . . . 131
7.4.2.2 RxHeader . . . 132
7.4.2.3 TxData . . . 132
7.4.2.4 TxHeader . . . 132
7.4.2.5 TxMailbox . . . 132
7.5 Core/Inc/daq.h File Reference . . . 133
7.5.1 Macro Definition Documentation . . . 134
7.5.1.1 _NUM_LOGGABLE_PARAMS . . . 134
7.5.2 Typedef Documentation . . . 134
7.5.2.1 daq_datapoint . . . 134
7.5.2.2 daq_loop_args . . . 134
7.5.2.3 uv_status . . . 134
7.5.3 Enumeration Type Documentation . . . 134
7.5.3.1 loggable_params . . . 134
7.5.4 Function Documentation . . . 136
7.5.4.1 associateDaqParamWithVar() . . . 136
7.5.4.2 daqMasterTask() . . . 136
7.5.4.3 initDaqTask() . . . 137
7.6 Core/Inc/dash.h File Reference . . . 137
7.6.1 Enumeration Type Documentation . . . 137
7.6.1.1 dash_can_ids . . . 137
7.6.2 Function Documentation . . . 138
7.6.2.1 Update_Batt_Temp() . . . 138
7.6.2.2 Update_RPM() . . . 138
7.6.2.3 Update_State_Of_Charge() . . . 138
7.7 Core/Inc/dma.h File Reference . . . 139
7.7.1 Detailed Description . . . 139
7.7.2 Function Documentation . . . 139
7.7.2.1 MX_DMA_Init() . . . 139
7.8 Core/Inc/driving_loop.h File Reference . . . 139
7.8.1 Typedef Documentation . . . 140
7.8.1.1 driving_loop_args . . . 141
7.8.1.2 drivingMode . . . 141
7.8.1.3 drivingModeParams . . . 141
7.8.1.4 exp_torque_map_args . . . 141
7.8.1.5 linear_torque_map_args . . . 141
7.8.1.6 MC_POWER . . . 141
7.8.1.7 MC_RPM . . . 142
7.8.1.8 MC_Torque . . . 142
7.8.1.9 s_curve_torque_map_args . . . 142
7.8.2 Enumeration Type Documentation . . . 142
7.8.2.1 DL_internal_state . . . 142

7.8.2.2 map_mode . . . 142
7.8.3 Function Documentation . . . 143
7.8.3.1 initDrivingLoop() . . . 143
7.8.3.2 StartDrivingLoop() . . . 143
7.9 Core/Inc/errorLUT.h File Reference . . . 145
7.9.1 Macro Definition Documentation . . . 145
7.9.1.1 _NUM_ERRORS_ . . . 145
7.10 Core/Inc/FreeRTOSConfig.h File Reference . . . 145
7.10.1 Macro Definition Documentation . . . 146
7.10.1.1 configASSERT . . . 147
7.10.1.2 configCHECK_FOR_STACK_OVERFLOW [1/2] . . . 147
7.10.1.3 configCHECK_FOR_STACK_OVERFLOW [2/2] . . . 147
7.10.1.4 configCPU_CLOCK_HZ . . . 147
7.10.1.5 configENABLE_BACKWARD_COMPATIBILITY . . . 147
7.10.1.6 configENABLE_FPU . . . 147
7.10.1.7 configENABLE_MPU . . . 148
7.10.1.8 configKERNEL_INTERRUPT_PRIORITY . . . 148
7.10.1.9 configLIBRARY_LOWEST_INTERRUPT_PRIORITY . . . 148
7.10.1.10 configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY . . . 148
7.10.1.11 configMAX_CO_ROUTINE_PRIORITIES . . . 148
7.10.1.12 configMAX_PRIORITIES . . . 148
7.10.1.13 configMAX_SYSCALL_INTERRUPT_PRIORITY . . . 149
7.10.1.14 configMAX_TASK_NAME_LEN . . . 149
7.10.1.15 configMESSAGE_BUFFER_LENGTH_TYPE . . . 149
7.10.1.16 configMINIMAL_STACK_SIZE . . . 149
7.10.1.17 configPRIO_BITS . . . 149
7.10.1.18 configQUEUE_REGISTRY_SIZE . . . 149
7.10.1.19 configRECORD_STACK_HIGH_ADDRESS . . . 150
7.10.1.20 configSUPPORT_DYNAMIC_ALLOCATION . . . 150
7.10.1.21 configSUPPORT_STATIC_ALLOCATION . . . 150
7.10.1.22 configTICK_RATE_HZ . . . 150
7.10.1.23 configTIMER_QUEUE_LENGTH . . . 150
7.10.1.24 configTIMER_TASK_PRIORITY . . . 150
7.10.1.25 configTIMER_TASK_STACK_DEPTH . . . 151
7.10.1.26 configTOTAL_HEAP_SIZE . . . 151
7.10.1.27 configUSE_16_BIT_TICKS . . . 151
7.10.1.28 configUSE_APPLICATION_TASK_TAG . . . 151
7.10.1.29 configUSE_CO_ROUTINES . . . 151
7.10.1.30 configUSE_COUNTING_SEMAPHORES . . . 151
7.10.1.31 configUSE_IDLE_HOOK . . . 152
7.10.1.32 configUSE_MALLOC_FAILED_HOOK [1/2] . . . 152
7.10.1.33 configUSE_MALLOC_FAILED_HOOK [2/2] . . . 152

7.10.1.34 configUSE_MUTEXES . . . 152
7.10.1.35 configUSE_PORT_OPTIMISED_TASK_SELECTION . . . 152
7.10.1.36 configUSE_PREEMPTION . . . 152
7.10.1.37 configUSE_TICK_HOOK . . . 153
7.10.1.38 configUSE_TIMERS . . . 153
7.10.1.39 INCLUDE_eTaskGetState . . . 153
7.10.1.40 INCLUDE_pcTaskGetTaskName . . . 153
7.10.1.41 INCLUDE_uxTaskGetStackHighWaterMark . . . 153
7.10.1.42 INCLUDE_uxTaskGetStackHighWaterMark2 . . . 153
7.10.1.43 INCLUDE_uxTaskPriorityGet . . . 154
7.10.1.44 INCLUDE_vTaskCleanUpResources . . . 154
7.10.1.45 INCLUDE_vTaskDelay . . . 154
7.10.1.46 INCLUDE_vTaskDelayUntil . . . 154
7.10.1.47 INCLUDE_vTaskDelete . . . 154
7.10.1.48 INCLUDE_vTaskPrioritySet . . . 154
7.10.1.49 INCLUDE_vTaskSuspend . . . 155
7.10.1.50 INCLUDE_xEventGroupSetBitFromISR . . . 155
7.10.1.51 INCLUDE_xQueueGetMutexHolder . . . 155
7.10.1.52 INCLUDE_xSemaphoreGetMutexHolder . . . 155
7.10.1.53 INCLUDE_xTaskAbortDelay . . . 155
7.10.1.54 INCLUDE_xTaskDelayUntil . . . 155
7.10.1.55 INCLUDE_xTaskGetCurrentTaskHandle . . . 156
7.10.1.56 INCLUDE_xTaskGetHandle . . . 156
7.10.1.57 INCLUDE_xTaskGetSchedulerState . . . 156
7.10.1.58 INCLUDE_xTimerPendFunctionCall . . . 156
7.10.1.59 vPortSVCHandler . . . 156
7.10.1.60 xPortPendSVHandler . . . 156
7.10.1.61 xPortSysTickHandler . . . 157
7.11 Core/Inc/gpio.h File Reference . . . 157
7.11.1 Detailed Description . . . 157
7.11.2 Function Documentation . . . 157
7.11.2.1 MX_GPIO_Init() . . . 157
7.12 Core/Inc/imd.h File Reference . . . 158
7.12.1 Typedef Documentation . . . 159
7.12.1.1 uv_imd_settings . . . 159
7.12.2 Enumeration Type Documentation . . . 159
7.12.2.1 imd_error_flags . . . 159
7.12.2.2 imd_high_resolution_measurements . . . 159
7.12.2.3 imd_manufacturer_requests . . . 160
7.12.2.4 imd_status_bits . . . 160
7.12.2.5 imd_status_requests . . . 161
7.12.3 Function Documentation . . . 161

7.12.3.1 IMD_Check_Battery_Voltage() . . . 161
7.12.3.2 IMD_Check_Error_Flags() . . . 161
7.12.3.3 IMD_Check_Isolation_Capacitances() . . . 162
7.12.3.4 IMD_Check_Isolation_Resistances() . . . 162
7.12.3.5 IMD_Check_Isolation_State() . . . 162
7.12.3.6 IMD_Check_Max_Battery_Working_Voltage() . . . 162
7.12.3.7 IMD_Check_Part_Name() . . . 163
7.12.3.8 IMD_Check_Safety_Touch_Current() . . . 163
7.12.3.9 IMD_Check_Safety_Touch_Energy() . . . 163
7.12.3.10 IMD_Check_Serial_Number() . . . 163
7.12.3.11 IMD_Check_Status_Bits() . . . 164
7.12.3.12 IMD_Check_Temperature() . . . 164
7.12.3.13 IMD_Check_Uptime() . . . 164
7.12.3.14 IMD_Check_Version() . . . 164
7.12.3.15 IMD_Check_Voltages_Vp_and_Vn() . . . 165
7.12.3.16 IMD_Parse_Message() . . . 165
7.12.3.17 IMD_Request_Status() . . . 165
7.12.3.18 IMD_Startup() . . . 165
7.12.3.19 initIMD() . . . 166
7.13 Core/Inc/main.h File Reference . . . 166
7.13.1 Detailed Description . . . 166
7.13.2 Macro Definition Documentation . . . 167
7.13.2.1 Blue_LED_GPIO_Port . . . 167
7.13.2.2 Blue_LED_Pin . . . 167
7.13.2.3 Orange_LED_GPIO_Port . . . 167
7.13.2.4 Orange_LED_Pin . . . 167
7.13.2.5 Red_LED_GPIO_Port . . . 167
7.13.2.6 Red_LED_Pin . . . 168
7.13.2.7 Start_Button_Input_EXTI_IRQn . . . 168
7.13.2.8 Start_Button_Input_GPIO_Port . . . 168
7.13.2.9 Start_Button_Input_Pin . . . 168
7.13.3 Function Documentation . . . 168
7.13.3.1 Error_Handler() . . . 168
7.14 Core/Inc/motor_controller.h File Reference . . . 169
7.14.1 Macro Definition Documentation . . . 170
7.14.1.1 FIRMWARE_VERSION_REGISTER . . . 171
7.14.1.2 N_set_cmd . . . 171
7.14.1.3 SERIAL_NUMBER_REGISTER . . . 171
7.14.2 Typedef Documentation . . . 171
7.14.2.1 motor_controller_settings . . . 171
7.14.2.2 uv_CAN_msg . . . 171
7.14.3 Enumeration Type Documentation . . . 171

7.14.3.1 motor_controller_current_parameters . . . 171
7.14.3.2 motor_controller_io . . . 172
7.14.3.3 motor_controller_limp_mode . . . 172
7.14.3.4 motor_controller_measurements . . . 172
7.14.3.5 motor_controller_motor_constants . . . 173
7.14.3.6 motor_controller_PI_values . . . 173
7.14.3.7 motor_controller_repeating_time . . . 174
7.14.3.8 motor_controller_speed_parameters . . . 174
7.14.3.9 motor_controller_startup . . . 174
7.14.3.10 motor_controller_status . . . 174
7.14.3.11 motor_controller_status_information_errors_warnings . . . 175
7.14.3.12 motor_controller_temperatures . . . 176
7.14.4 Function Documentation . . . 176
7.14.4.1 MC_Request_Data() . . . 176
7.14.4.2 MC_Startup() . . . 176
7.14.4.3 MotorControllerSpinTest() . . . 177
7.14.4.4 Parse_Bamocar_Response() . . . 177
7.14.4.5 ProcessMotorControllerResponse() . . . 178
7.14.5 Variable Documentation . . . 178
7.14.5.1 mc_default_settings . . . 178
7.15 Core/Inc/odometer.h File Reference . . . 178
7.15.1 Typedef Documentation . . . 179
7.15.1.1 uv_persistant_data_frame . . . 179
7.15.2 Function Documentation . . . 179
7.15.2.1 initOdometer() . . . 179
7.15.2.2 odometerTask() . . . 179
7.16 Core/Inc/oled.h File Reference . . . 180
7.16.1 Function Documentation . . . 180
7.16.1.1 amogus() . . . 180
7.16.1.2 oled_config() . . . 180
7.16.1.3 oled_Write() . . . 180
7.16.1.4 oled_Write_Cmd() . . . 180
7.16.1.5 oled_Write_Data() . . . 181
7.16.1.6 refresh_OLED() . . . 181
7.16.1.7 wait() . . . 181
7.17 Core/Inc/pdu.h File Reference . . . 181
7.17.1 Typedef Documentation . . . 182
7.17.1.1 uv19_pdu_settings . . . 182
7.17.2 Enumeration Type Documentation . . . 182
7.17.2.1 pdu_messages_20A . . . 182
7.17.2.2 pdu_messages_5A . . . 182
7.17.3 Function Documentation . . . 183

7.17.3.1 initPDU() . . . 183
7.17.3.2 PDU_disable_brake_light() . . . 183
7.17.3.3 PDU_disable_coolant_pump() . . . 183
7.17.3.4 PDU_disable_cooling_fans() . . . 184
7.17.3.5 PDU_disable_motor_controller() . . . 184
7.17.3.6 PDU_disable_shutdown_circuit() . . . 184
7.17.3.7 PDU_enable_brake_light() . . . 184
7.17.3.8 PDU_enable_coolant_pump() . . . 184
7.17.3.9 PDU_enable_cooling_fans() . . . 185
7.17.3.10 PDU_enable_motor_controller() . . . 185
7.17.3.11 PDU_enable_shutdown_circuit() . . . 185
7.17.3.12 PDU_speaker_chirp() . . . 185
7.18 Core/Inc/rb_tree.h File Reference . . . 185
7.18.1 Macro Definition Documentation . . . 186
7.18.1.1 BLACK . . . 187
7.18.1.2 RB_APPLY . . . 187
7.18.1.3 RB_DUP . . . 187
7.18.1.4 RB_FIRST . . . 187
7.18.1.5 RB_ISEMPTY . . . 187
7.18.1.6 RB_MIN . . . 188
7.18.1.7 RB_MINIMAL . . . 188
7.18.1.8 RB_NIL . . . 188
7.18.1.9 RB_ROOT . . . 188
7.18.1.10 RED . . . 188
7.18.2 Typedef Documentation . . . 188
7.18.2.1 rbnode . . . 189
7.18.3 Enumeration Type Documentation . . . 189
7.18.3.1 rbtraversal . . . 189
7.18.4 Function Documentation . . . 190
7.18.4.1 rbApplyNode() . . . 190
7.18.4.2 rbCheckBlackHeight() . . . 191
7.18.4.3 rbCheckOrder() . . . 191
7.18.4.4 rbCreate() . . . 191
7.18.4.5 rbDelete() . . . 192
7.18.4.6 rbDestroy() . . . 192
7.18.4.7 rbFind() . . . 193
7.18.4.8 rbInsert() . . . 193
7.18.4.9 rbPrint() . . . 194
7.18.4.10 rbSuccessor() . . . 194
7.19 Core/Inc/spi.h File Reference . . . 194
7.19.1 Detailed Description . . . 195
7.19.2 Function Documentation . . . 195

7.19.2.1 MX_SPI1_Init() . . . 195
7.19.3 Variable Documentation . . . 195
7.19.3.1 hspi1 . . . 195
7.20 Core/Inc/stm32f4xx_hal_conf.h File Reference . . . 196
7.20.1 Detailed Description . . . 198
7.20.2 Macro Definition Documentation . . . 198
7.20.2.1 assert_param . . . 199
7.20.2.2 DATA_CACHE_ENABLE . . . 199
7.20.2.3 DP83848_PHY_ADDRESS . . . 199
7.20.2.4 ETH_RX_BUF_SIZE . . . 199
7.20.2.5 ETH_RXBUFNB . . . 199
7.20.2.6 ETH_TX_BUF_SIZE . . . 199
7.20.2.7 ETH_TXBUFNB . . . 200
7.20.2.8 EXTERNAL_CLOCK_VALUE . . . 200
7.20.2.9 HAL_ADC_MODULE_ENABLED . . . 200
7.20.2.10 HAL_CAN_MODULE_ENABLED . . . 200
7.20.2.11 HAL_CORTEX_MODULE_ENABLED . . . 200
7.20.2.12 HAL_DMA_MODULE_ENABLED . . . 200
7.20.2.13 HAL_EXTI_MODULE_ENABLED . . . 201
7.20.2.14 HAL_FLASH_MODULE_ENABLED . . . 201
7.20.2.15 HAL_GPIO_MODULE_ENABLED . . . 201
7.20.2.16 HAL_MODULE_ENABLED . . . 201
7.20.2.17 HAL_PWR_MODULE_ENABLED . . . 201
7.20.2.18 HAL_RCC_MODULE_ENABLED . . . 201
7.20.2.19 HAL_SPI_MODULE_ENABLED . . . 202
7.20.2.20 HAL_TIM_MODULE_ENABLED . . . 202
7.20.2.21 HSE_STARTUP_TIMEOUT . . . 202
7.20.2.22 HSE_VALUE . . . 202
7.20.2.23 HSI_VALUE . . . 202
7.20.2.24 INSTRUCTION_CACHE_ENABLE . . . 203
7.20.2.25 LSE_STARTUP_TIMEOUT . . . 203
7.20.2.26 LSE_VALUE . . . 203
7.20.2.27 LSI_VALUE . . . 203
7.20.2.28 MAC_ADDR0 . . . 203
7.20.2.29 MAC_ADDR1 . . . 204
7.20.2.30 MAC_ADDR2 . . . 204
7.20.2.31 MAC_ADDR3 . . . 204
7.20.2.32 MAC_ADDR4 . . . 204
7.20.2.33 MAC_ADDR5 . . . 204
7.20.2.34 PHY_AUTONEGO_COMPLETE . . . 204
7.20.2.35 PHY_AUTONEGOTIATION . . . 205
7.20.2.36 PHY_BCR . . . 205

7.20.2.37 PHY_BSR . . . 205
7.20.2.38 PHY_CONFIG_DELAY . . . 205
7.20.2.39 PHY_DUPLEX_STATUS . . . 205
7.20.2.40 PHY_FULLDUPLEX_100M . . . 206
7.20.2.41 PHY_FULLDUPLEX_10M . . . 206
7.20.2.42 PHY_HALFDUPLEX_100M . . . 206
7.20.2.43 PHY_HALFDUPLEX_10M . . . 206
7.20.2.44 PHY_ISOLATE . . . 206
7.20.2.45 PHY_JABBER_DETECTION . . . 207
7.20.2.46 PHY_LINKED_STATUS . . . 207
7.20.2.47 PHY_LOOPBACK . . . 207
7.20.2.48 PHY_POWERDOWN . . . 207
7.20.2.49 PHY_READ_TO . . . 207
7.20.2.50 PHY_RESET . . . 208
7.20.2.51 PHY_RESET_DELAY . . . 208
7.20.2.52 PHY_RESTART_AUTONEGOTIATION . . . 208
7.20.2.53 PHY_SPEED_STATUS . . . 208
7.20.2.54 PHY_SR . . . 208
7.20.2.55 PHY_WRITE_TO . . . 209
7.20.2.56 PREFETCH_ENABLE . . . 209
7.20.2.57 TICK_INT_PRIORITY . . . 209
7.20.2.58 USE_HAL_ADC_REGISTER_CALLBACKS . . . 209
7.20.2.59 USE_HAL_CAN_REGISTER_CALLBACKS . . . 209
7.20.2.60 USE_HAL_CEC_REGISTER_CALLBACKS . . . 209
7.20.2.61 USE_HAL_CRYP_REGISTER_CALLBACKS . . . 210
7.20.2.62 USE_HAL_DAC_REGISTER_CALLBACKS . . . 210
7.20.2.63 USE_HAL_DCMI_REGISTER_CALLBACKS . . . 210
7.20.2.64 USE_HAL_DFSDM_REGISTER_CALLBACKS . . . 210
7.20.2.65 USE_HAL_DMA2D_REGISTER_CALLBACKS . . . 210
7.20.2.66 USE_HAL_DSI_REGISTER_CALLBACKS . . . 210
7.20.2.67 USE_HAL_ETH_REGISTER_CALLBACKS . . . 211
7.20.2.68 USE_HAL_FMPI2C_REGISTER_CALLBACKS . . . 211
7.20.2.69 USE_HAL_FMPSMBUS_REGISTER_CALLBACKS . . . 211
7.20.2.70 USE_HAL_HASH_REGISTER_CALLBACKS . . . 211
7.20.2.71 USE_HAL_HCD_REGISTER_CALLBACKS . . . 211
7.20.2.72 USE_HAL_I2C_REGISTER_CALLBACKS . . . 211
7.20.2.73 USE_HAL_I2S_REGISTER_CALLBACKS . . . 212
7.20.2.74 USE_HAL_IRDA_REGISTER_CALLBACKS . . . 212
7.20.2.75 USE_HAL_LPTIM_REGISTER_CALLBACKS . . . 212
7.20.2.76 USE_HAL_LTDC_REGISTER_CALLBACKS . . . 212
7.20.2.77 USE_HAL_MMC_REGISTER_CALLBACKS . . . 212
7.20.2.78 USE_HAL_NAND_REGISTER_CALLBACKS . . . 212

7.20.2.79 USE_HAL_NOR_REGISTER_CALLBACKS . . . 213
7.20.2.80 USE_HAL_PCCARD_REGISTER_CALLBACKS . . . 213
7.20.2.81 USE_HAL_PCD_REGISTER_CALLBACKS . . . 213
7.20.2.82 USE_HAL_QSPI_REGISTER_CALLBACKS . . . 213
7.20.2.83 USE_HAL_RNG_REGISTER_CALLBACKS . . . 213
7.20.2.84 USE_HAL_RTC_REGISTER_CALLBACKS . . . 213
7.20.2.85 USE_HAL_SAI_REGISTER_CALLBACKS . . . 214
7.20.2.86 USE_HAL_SD_REGISTER_CALLBACKS . . . 214
7.20.2.87 USE_HAL_SDRAM_REGISTER_CALLBACKS . . . 214
7.20.2.88 USE_HAL_SMARTCARD_REGISTER_CALLBACKS . . . 214
7.20.2.89 USE_HAL_SMBUS_REGISTER_CALLBACKS . . . 214
7.20.2.90 USE_HAL_SPDIFRX_REGISTER_CALLBACKS . . . 214
7.20.2.91 USE_HAL_SPI_REGISTER_CALLBACKS . . . 215
7.20.2.92 USE_HAL_SRAM_REGISTER_CALLBACKS . . . 215
7.20.2.93 USE_HAL_TIM_REGISTER_CALLBACKS . . . 215
7.20.2.94 USE_HAL_UART_REGISTER_CALLBACKS . . . 215
7.20.2.95 USE_HAL_USART_REGISTER_CALLBACKS . . . 215
7.20.2.96 USE_HAL_WWDG_REGISTER_CALLBACKS . . . 215
7.20.2.97 USE_RTOS . . . 216
7.20.2.98 USE_SPI_CRC . . . 216
7.20.2.99 VDD_VALUE . . . 216
7.21 Core/Inc/stm32f4xx_it.h File Reference . . . 216
7.21.1 Detailed Description . . . 217
7.21.2 Function Documentation . . . 217
7.21.2.1 ADC_IRQHandler() . . . 217
7.21.2.2 BusFault_Handler() . . . 217
7.21.2.3 CAN2_RX0_IRQHandler() . . . 217
7.21.2.4 CAN2_RX1_IRQHandler() . . . 218
7.21.2.5 CAN2_TX_IRQHandler() . . . 218
7.21.2.6 DebugMon_Handler() . . . 218
7.21.2.7 DMA2_Stream0_IRQHandler() . . . 218
7.21.2.8 EXTI0_IRQHandler() . . . 219
7.21.2.9 HardFault_Handler() . . . 219
7.21.2.10 MemManage_Handler() . . . 219
7.21.2.11 NMI_Handler() . . . 219
7.21.2.12 TIM1_UP_TIM10_IRQHandler() . . . 220
7.21.2.13 UsageFault_Handler() . . . 220
7.22 Core/Inc/temp_monitoring.h File Reference . . . 220
7.22.1 Function Documentation . . . 220
7.22.1.1 initTempMonitor() . . . 220
7.22.1.2 tempMonitorTask() . . . 221
7.23 Core/Inc/tim.h File Reference . . . 221

7.23.1 Detailed Description . . . 221
7.23.2 Function Documentation . . . 222
7.23.2.1 MX_TIM3_Init() . . . 222
7.23.3 Variable Documentation . . . 222
7.23.3.1 htim3 . . . 222
7.24 Core/Inc/uvfr_global_config.h File Reference . . . 222
7.24.1 Macro Definition Documentation . . . 222
7.24.1.1 ECUMASTER_PMU . . . 223
7.24.1.2 FIRMWARE_MAJOR_RELEASE . . . 223
7.24.1.3 FIRMWARE_MINOR_RELEASE . . . 223
7.24.1.4 FIRMWARE_PATCH_NUM . . . 223
7.24.1.5 STM32_F407 . . . 223
7.24.1.6 STM32_H7xx . . . 223
7.24.1.7 USE_OS_MEM_MGMT . . . 224
7.24.1.8 UV19_PDU . . . 224
7.24.1.9 UV_MALLOC_LIMIT . . . 224
7.25 Core/Inc/uvfr_settings.h File Reference . . . 224
7.25.1 Macro Definition Documentation . . . 226
7.25.1.1 BMS_ADDR . . . 226
7.25.1.2 BMS_MGROUP . . . 227
7.25.1.3 BMS_OFFSET . . . 227
7.25.1.4 CRC_MGROUP1 . . . 227
7.25.1.5 CRC_MGROUP2 . . . 227
7.25.1.6 CRC_POLY . . . 227
7.25.1.7 DAQ_HEAD_ADDR . . . 227
7.25.1.8 DAQ_HEAD_MGROUP . . . 228
7.25.1.9 DAQ_HEAD_OFFSET . . . 228
7.25.1.10 DAQ_PARAMS1_ADDR . . . 228
7.25.1.11 DAQ_PARAMS1_MGROUP . . . 228
7.25.1.12 DAQ_PARAMS1_OFFSET . . . 228
7.25.1.13 DAQ_PARAMS2_ADDR . . . 228
7.25.1.14 DAQ_PARAMS2_MGROUP . . . 229
7.25.1.15 DAQ_PARAMS2_OFFSET . . . 229
7.25.1.16 DAQ_PARAMS3_ADDR . . . 229
7.25.1.17 DAQ_PARAMS3_MGROUP . . . 229
7.25.1.18 DAQ_PARAMS3_OFFSET . . . 229
7.25.1.19 DRIVING_ADDR . . . 229
7.25.1.20 DRIVING_MGROUP . . . 230
7.25.1.21 DRIVING_OFFSET . . . 230
7.25.1.22 ENABLE_FLASH_SETTINGS . . . 230
7.25.1.23 FLASH_SBLOCK_START . . . 230
7.25.1.24 GENERAL_VEH_INFO_MGROUP . . . 230

7.25.1.25 GENERAL_VEH_INFO_OFFSET . . . 230
7.25.1.26 IMD_ADDR . . . 231
7.25.1.27 IMD_MGROUP . . . 231
7.25.1.28 IMD_OFFSET . . . 231
7.25.1.29 isPtrToFlash . . . 231
7.25.1.30 MAGIC_NUMBER . . . 231
7.25.1.31 MOTOR_ADDR . . . 231
7.25.1.32 MOTOR_MGROUP . . . 232
7.25.1.33 MOTOR_OFFSET . . . 232
7.25.1.34 OS_SETTINGS_ADDR . . . 232
7.25.1.35 OS_SETTINGS_MGROUP . . . 232
7.25.1.36 OS_SETTINGS_OFFSET . . . 232
7.25.1.37 PDU_ADDR . . . 232
7.25.1.38 PDU_MGROUP . . . 233
7.25.1.39 PDU_OFFSET . . . 233
7.25.1.40 PERSISTENT_DATA_MGROUP . . . 233
7.25.1.41 SBLOCK_CRC_REGION_OFFSET . . . 233
7.25.1.42 SBLOCK_CSR_OFFSET . . . 233
7.25.1.43 SETTING_BRANCH_SIZE . . . 233
7.25.1.44 SIZE_OF_MGROUP . . . 234
7.25.1.45 SIZEOF_SBLOCK . . . 234
7.25.1.46 SIZEOF_USER_FLASH . . . 234
7.25.1.47 START_OF_USER_FLASH . . . 234
7.25.1.48 SYS_DATA_OFFSET . . . 234
7.25.1.49 TOP_OF_FLASH_SBLOCK . . . 234
7.25.1.50 TOP_OF_USER_FLASH . . . 235
7.25.2 Typedef Documentation . . . 235
7.25.2.1 motor_controller_settings . . . 235
7.25.2.2 uv_status . . . 235
7.25.2.3 uv_vehicle_settings . . . 235
7.25.2.4 veh_gen_info . . . 235
7.25.3 Enumeration Type Documentation . . . 235
7.25.3.1 laptop_CMD . . . 235
7.25.4 Function Documentation . . . 236
7.25.4.1 nukeSettings() . . . 236
7.25.4.2 setupDefaultSettings() . . . 236
7.25.4.3 uvComputeMemRegionChecksum() . . . 237
7.25.4.4 uvComputeSettingsCheckSums() . . . 237
7.25.4.5 uvConfigSettingTask() . . . 237
7.25.4.6 uvSaveSettingsToFlash() . . . 237
7.25.4.7 uvSettingsInit() . . . 238
7.25.4.8 uvValidateChecksums() . . . 238

7.25.4.9 uvValidateSettingsFromFlash() . . . 238
7.25.5 Variable Documentation . . . 238
7.25.5.1 current_vehicle_settings . . . 238
7.26 Core/Inc/uvfr_state_engine.h File Reference . . . 238
7.26.1 Macro Definition Documentation . . . 241
7.26.1.1 _LONGEST_SC_TIME . . . 242
7.26.1.2 _SC_DAEMON_PERIOD . . . 242
7.26.1.3 _UV_DEFAULT_TASK_INSTANCES . . . 242
7.26.1.4 _UV_DEFAULT_TASK_PERIOD . . . 242
7.26.1.5 _UV_DEFAULT_TASK_STACK_SIZE . . . 242
7.26.1.6 _UV_MIN_TASK_PERIOD . . . 242
7.26.1.7 SVC_TASK_MAX_CHECKIN_PERIOD . . . 243
7.26.2 Typedef Documentation . . . 243
7.26.2.1 uv_status . . . 243
7.26.2.2 uv_task_id . . . 243
7.26.2.3 uv_timespan_ms . . . 243
7.26.3 Function Documentation . . . 243
7.26.3.1 getSVCTaskID() . . . 243
7.26.3.2 updateRunningTasks() . . . 244
7.26.3.3 uvGetTaskById() . . . 244
7.26.3.4 uvRegisterTask() . . . 244
7.27 Core/Inc/uvfr_utils.h File Reference . . . 244
7.27.1 Detailed Description . . . 247
7.27.2 Macro Definition Documentation . . . 247
7.27.2.1 INIT_CHECK_PERIOD . . . 247
7.27.2.2 MAX_INIT_TIME . . . 247
7.27.2.3 USE_OLED_DEBUG . . . 247
7.27.2.4 UV_CAN1 . . . 248
7.27.2.5 UV_CAN2 . . . 248
7.27.2.6 UV_CAN_CHANNEL_MASK . . . 248
7.27.2.7 UV_CAN_DYNAMIC_MEM . . . 248
7.27.2.8 UV_CAN_EXTENDED_ID . . . 248
7.27.3 Typedef Documentation . . . 248
7.27.3.1 access_control_info . . . 248
7.27.3.2 access_control_type . . . 249
7.27.3.3 bool . . . 249
7.27.3.4 p_status . . . 249
7.27.3.5 uv_CAN_msg . . . 249
7.27.3.6 uv_ext_device_id . . . 249
7.27.3.7 uv_init_struct . . . 249
7.27.3.8 uv_init_task_args . . . 249
7.27.3.9 uv_init_task_response . . . 250

7.27.3.10 uv_internal_params . . . 250
7.27.3.11 uv_msg_type . . . 250
7.27.3.12 uv_status . . . 250
7.27.3.13 uv_task_cmd . . . 250
7.27.3.14 uv_task_id . . . 250
7.27.3.15 uv_task_msg . . . 251
7.27.3.16 uv_timespan_ms . . . 251
7.27.4 Enumeration Type Documentation . . . 251
7.27.4.1 access_control_t . . . 251
7.27.4.2 data_type . . . 251
7.27.4.3 uv_driving_mode_t . . . 252
7.27.4.4 uv_external_device . . . 252
7.27.4.5 uv_msg_type_t . . . 252
7.27.4.6 uv_status_t . . . 253
7.27.5 Function Documentation . . . 253
7.27.5.1 __uvInitPanic() . . . 253
7.27.5.2 uvInit() . . . 254
7.27.5.3 uvIsPTRValid() . . . 256
7.27.6 Variable Documentation . . . 256
7.27.6.1 data_size . . . 256
7.27.6.2 global_context . . . 256
7.28 Core/Inc/uvfr_vehicle_commands.h File Reference . . . 257
7.28.1 Macro Definition Documentation . . . 257
7.28.1.1 uvHonkHorn . . . 257
7.28.1.2 uvOpenSDC [1/2] . . . 257
7.28.1.3 uvOpenSDC [2/2] . . . 258
7.28.1.4 uvSilenceHorn [1/2] . . . 258
7.28.1.5 uvSilenceHorn [2/2] . . . 258
7.28.1.6 uvStartCoolantPump . . . 258
7.28.1.7 uvStartFans . . . 258
7.28.1.8 uvStopCoolantPump . . . 258
7.28.1.9 uvStopFans . . . 259
7.28.2 Function Documentation . . . 259
7.28.2.1 _uvCloseSDC_canBased() . . . 259
7.28.2.2 _uvHonkHorn_canBased() . . . 259
7.28.2.3 _uvOpenSDC_canBased() . . . 259
7.28.2.4 _uvSilenceHorn_canBased() . . . 259
7.28.2.5 _uvStartCoolantPump_canBased() . . . 259
7.28.2.6 _uvStopCoolantPump_canBased() . . . 259
7.28.2.7 uvSecureVehicle() . . . 260
7.29 Core/Src/adc.c File Reference . . . 260
7.29.1 Detailed Description . . . 261

7.29.2 Macro Definition Documentation . . . 261
7.29.2.1 DEBUG_TOGGLE_LED . . . 261
7.29.3 Function Documentation . . . 261
7.29.3.1 HAL_ADC_MspDeInit() . . . 261
7.29.3.2 HAL_ADC_MspInit() . . . 262
7.29.3.3 initADCTask() . . . 262
7.29.3.4 MX_ADC1_Init() . . . 262
7.29.3.5 MX_ADC2_Init() . . . 263
7.29.3.6 processADCBuffer() . . . 263
7.29.3.7 StartADCTask() . . . 263
7.29.4 Variable Documentation . . . 263
7.29.4.1 adc1_APPS1 . . . 263
7.29.4.2 adc1_APPS2 . . . 264
7.29.4.3 adc1_BPS1 . . . 264
7.29.4.4 adc1_BPS2 . . . 264
7.29.4.5 adc_buf1 . . . 264
7.29.4.6 hadc1 . . . 264
7.29.4.7 hadc2 . . . 265
7.29.4.8 hdma_adc1 . . . 265
7.30 Core/Src/bms.c File Reference . . . 265
7.30.1 Function Documentation . . . 265
7.30.1.1 BMS_Init() . . . 266
7.30.1.2 handleBMSVoltageMsg() . . . 266
7.30.2 Variable Documentation . . . 266
7.30.2.1 default_bms_settings . . . 266
7.31 Core/Src/can.c File Reference . . . 266
7.31.1 Detailed Description . . . 267
7.31.2 Macro Definition Documentation . . . 268
7.31.2.1 HAL_CAN_ERROR_INVALID_CALLBACK . . . 268
7.31.2.2 table_size . . . 268
7.31.3 Typedef Documentation . . . 268
7.31.3.1 CAN_Callback . . . 268
7.31.4 Function Documentation . . . 268
7.31.4.1 __uvCANtxCritSection() . . . 268
7.31.4.2 callFunctionFromCANid() . . . 269
7.31.4.3 CANbusRxSvcDaemon() . . . 269
7.31.4.4 CANbusTxSvcDaemon() . . . 269
7.31.4.5 generateHash() . . . 270
7.31.4.6 HAL_CAN_MspDeInit() . . . 270
7.31.4.7 HAL_CAN_MspInit() . . . 270
7.31.4.8 HAL_CAN_RxFifo0MsgPendingCallback() . . . 270
7.31.4.9 HAL_CAN_RxFifo1MsgPendingCallback() . . . 271

7.31.4.10 handleCANbusError() . . . 271
7.31.4.11 MX_CAN2_Init() . . . 271
7.31.4.12 nuke_hash_table() . . . 271
7.31.5 Variable Documentation . . . 271
7.31.5.1 callback_table_mutex . . . 272
7.31.5.2 CAN_callback_table . . . 272
7.31.5.3 hcan2 . . . 272
7.31.5.4 Rx_msg_queue . . . 272
7.31.5.5 Tx_msg_queue . . . 273
7.32 Core/Src/constants.c File Reference . . . 273
7.32.1 Variable Documentation . . . 273
7.32.1.1 RxData . . . 273
7.32.1.2 RxHeader . . . 273
7.32.1.3 TxData . . . 274
7.32.1.4 TxHeader . . . 274
7.32.1.5 TxMailbox . . . 274
7.33 Core/Src/daq.c File Reference . . . 274
7.33.1 Macro Definition Documentation . . . 275
7.33.1.1 _SRC_UVFR_DAQ . . . 275
7.33.2 Typedef Documentation . . . 276
7.33.2.1 daq_child_task . . . 276
7.33.2.2 daq_param_list_node . . . 276
7.33.3 Function Documentation . . . 276
7.33.3.1 associateDaqParamWithVar() . . . 276
7.33.3.2 configureDaqSubTasks() . . . 276
7.33.3.3 daqMasterTask() . . . 277
7.33.3.4 daqSubTask() . . . 277
7.33.3.5 initDaqTask() . . . 277
7.33.3.6 insertParamToParamList() . . . 278
7.33.3.7 insertParamToRegister() . . . 278
7.33.3.8 sendAllParamsFromList() . . . 278
7.33.3.9 sendDaqMsg() . . . 278
7.33.3.10 startDaqSubTasks() . . . 279
7.33.3.11 stopDaqSubTasks() . . . 279
7.33.4 Variable Documentation . . . 279
7.33.4.1 curr_daq_settings . . . 279
7.33.4.2 daq_tlist . . . 279
7.33.4.3 datapoints . . . 280
7.33.4.4 default_daq_settings . . . 280
7.33.4.5 default_datapoints . . . 280
7.33.4.6 param_bank . . . 280
7.33.4.7 param_ptrs . . . 281

7.33.4.8 tmp_daq_msg . . . 281
7.34 Core/Src/dash.c File Reference . . . 281
7.34.1 Function Documentation . . . 281
7.34.1.1 Update_Batt_Temp() . . . 281
7.34.1.2 Update_RPM() . . . 282
7.34.1.3 Update_State_Of_Charge() . . . 282
7.35 Core/Src/dma.c File Reference . . . 282
7.35.1 Detailed Description . . . 282
7.35.2 Function Documentation . . . 283
7.35.2.1 MX_DMA_Init() . . . 283
7.36 Core/Src/driving_loop.c File Reference . . . 283
7.36.1 Detailed Description . . . 284
7.36.2 Macro Definition Documentation . . . 284
7.36.2.1 ACCELERATION . . . 284
7.36.2.2 AUTOCROSS . . . 284
7.36.2.3 ENDURANCE . . . 284
7.36.3 Function Documentation . . . 284
7.36.3.1 applyTorqueFilter() . . . 284
7.36.3.2 calculateThrottlePercentage() . . . 285
7.36.3.3 getKValue() . . . 285
7.36.3.4 initDrivingLoop() . . . 285
7.36.3.5 mapThrottleToTorque() . . . 286
7.36.3.6 performSafetyChecks() . . . 286
7.36.3.7 sendTorqueToMotorController() . . . 287
7.36.3.8 StartDrivingLoop() . . . 287
7.36.4 Variable Documentation . . . 288
7.36.4.1 adc1_APPS1 . . . 289
7.36.4.2 adc1_APPS2 . . . 289
7.36.4.3 adc1_BPS1 . . . 289
7.36.4.4 adc1_BPS2 . . . 289
7.36.4.5 default_dl_settings . . . 289
7.36.4.6 driving_args . . . 290
7.36.4.7 is_accelerating . . . 290
7.36.4.8 T_PREV . . . 290
7.36.4.9 T_REQ . . . 290
7.37 Core/Src/freertos.c File Reference . . . 290
7.37.1 Function Documentation . . . 291
7.37.1.1 MX_FREERTOS_Init() . . . 291
7.37.1.2 StartDefaultTask() . . . 292
7.37.1.3 vApplicationGetIdleTaskMemory() . . . 292
7.37.1.4 vApplicationGetTimerTaskMemory() . . . 292
7.37.1.5 vApplicationIdleHook() . . . 293

7.37.1.6 vApplicationMallocFailedHook() . . . 293
7.37.1.7 vApplicationStackOverflowHook() . . . 293
7.37.1.8 vApplicationTickHook() . . . 293
7.37.2 Variable Documentation . . . 293
7.37.2.1 defaultTaskHandle . . . 293
7.37.2.2 init_settings . . . 294
7.37.2.3 init_task_handle . . . 294
7.37.2.4 xIdleStack . . . 294
7.37.2.5 xIdleTaskTCBBuffer . . . 294
7.37.2.6 xTimerStack . . . 295
7.37.2.7 xTimerTaskTCBBuffer . . . 295
7.38 Core/Src/gpio.c File Reference . . . 295
7.38.1 Detailed Description . . . 295
7.38.2 Function Documentation . . . 295
7.38.2.1 MX_GPIO_Init() . . . 296
7.39 Core/Src/imd.c File Reference . . . 296
7.39.1 Function Documentation . . . 297
7.39.1.1 IMD_Check_Battery_Voltage() . . . 297
7.39.1.2 IMD_Check_Error_Flags() . . . 297
7.39.1.3 IMD_Check_Isolation_Capacitances() . . . 298
7.39.1.4 IMD_Check_Isolation_Resistances() . . . 298
7.39.1.5 IMD_Check_Isolation_State() . . . 298
7.39.1.6 IMD_Check_Max_Battery_Working_Voltage() . . . 298
7.39.1.7 IMD_Check_Part_Name() . . . 299
7.39.1.8 IMD_Check_Safety_Touch_Current() . . . 299
7.39.1.9 IMD_Check_Safety_Touch_Energy() . . . 299
7.39.1.10 IMD_Check_Serial_Number() . . . 299
7.39.1.11 IMD_Check_Status_Bits() . . . 300
7.39.1.12 IMD_Check_Temperature() . . . 300
7.39.1.13 IMD_Check_Uptime() . . . 300
7.39.1.14 IMD_Check_Version() . . . 300
7.39.1.15 IMD_Check_Voltages_Vp_and_Vn() . . . 301
7.39.1.16 IMD_Parse_Message() . . . 301
7.39.1.17 IMD_Request_Status() . . . 301
7.39.1.18 IMD_Startup() . . . 301
7.39.1.19 initIMD() . . . 302
7.39.2 Variable Documentation . . . 302
7.39.2.1 default_imd_settings . . . 302
7.39.2.2 IMD_error_flags_requested . . . 302
7.39.2.3 IMD_Expected_Part_Name . . . 302
7.39.2.4 IMD_Expected_Serial_Number . . . 303
7.39.2.5 IMD_Expected_Version . . . 303

7.39.2.6 IMD_High_Uncertainty . . . 303
7.39.2.7 IMD_Part_Name_0_Set . . . 303
7.39.2.8 IMD_Part_Name_1_Set . . . 303
7.39.2.9 IMD_Part_Name_2_Set . . . 304
7.39.2.10 IMD_Part_Name_3_Set . . . 304
7.39.2.11 IMD_Part_Name_Set . . . 304
7.39.2.12 IMD_Read_Part_Name . . . 304
7.39.2.13 IMD_Read_Serial_Number . . . 304
7.39.2.14 IMD_Read_Version . . . 305
7.39.2.15 IMD_Serial_Number_0_Set . . . 305
7.39.2.16 IMD_Serial_Number_1_Set . . . 305
7.39.2.17 IMD_Serial_Number_2_Set . . . 305
7.39.2.18 IMD_Serial_Number_3_Set . . . 305
7.39.2.19 IMD_Serial_Number_Set . . . 306
7.39.2.20 IMD_status_bits . . . 306
7.39.2.21 IMD_Temperature . . . 306
7.39.2.22 IMD_Version_0_Set . . . 306
7.39.2.23 IMD_Version_1_Set . . . 306
7.39.2.24 IMD_Version_2_Set . . . 307
7.39.2.25 IMD_Version_Set . . . 307
7.40 Core/Src/main.c File Reference . . . 307
7.40.1 Detailed Description . . . 308
7.40.2 Macro Definition Documentation . . . 308
7.40.2.1 DEBUG_CAN_IN_MAIN . . . 308
7.40.3 Function Documentation . . . 308
7.40.3.1 Error_Handler() . . . 308
7.40.3.2 HAL_ADC_ConvCpltCallback() . . . 309
7.40.3.3 HAL_ADC_LevelOutOfWindowCallback() . . . 309
7.40.3.4 HAL_GPIO_EXTI_Callback() . . . 309
7.40.3.5 HAL_TIM_PeriodElapsedCallback() . . . 309
7.40.3.6 main() . . . 310
7.40.3.7 MX_FREERTOS_Init() . . . 310
7.40.3.8 SystemClock_Config() . . . 311
7.40.4 Variable Documentation . . . 311
7.40.4.1 adc1_APPS1 . . . 311
7.40.4.2 adc1_APPS2 . . . 312
7.40.4.3 adc1_BPS1 . . . 312
7.40.4.4 adc1_BPS2 . . . 312
7.40.4.5 adc2_CoolantFlow . . . 312
7.40.4.6 adc2_CoolantTemp . . . 312
7.40.4.7 adc_buf1 . . . 313
7.40.4.8 adc_buf2 . . . 313

7.41 Core/Src/motor_controller.c File Reference . . . 313
7.41.1 Function Documentation . . . 314
7.41.1.1 MC_Request_Data() . . . 314
7.41.1.2 MC_Set_Param() . . . 314
7.41.1.3 MC_Startup() . . . 314
7.41.1.4 MotorControllerErrorHandler_16bitLE() . . . 315
7.41.1.5 MotorControllerSpinTest() . . . 315
7.41.1.6 Parse_Bamocar_Response() . . . 315
7.41.1.7 ProcessMotorControllerResponse() . . . 316
7.41.2 Variable Documentation . . . 316
7.41.2.1 mc_default_settings . . . 316
7.42 Core/Src/odometer.c File Reference . . . 316
7.42.1 Function Documentation . . . 317
7.42.1.1 initOdometer() . . . 317
7.42.1.2 odometerTask() . . . 317
7.43 Core/Src/oled.c File Reference . . . 317
7.44 Core/Src/pdu.c File Reference . . . 318
7.44.1 Function Documentation . . . 318
7.44.1.1 initPDU() . . . 318
7.44.1.2 PDU_disable_brake_light() . . . 319
7.44.1.3 PDU_disable_coolant_pump() . . . 319
7.44.1.4 PDU_disable_cooling_fans() . . . 319
7.44.1.5 PDU_disable_motor_controller() . . . 319
7.44.1.6 PDU_disable_shutdown_circuit() . . . 319
7.44.1.7 PDU_enable_brake_light() . . . 320
7.44.1.8 PDU_enable_coolant_pump() . . . 320
7.44.1.9 PDU_enable_cooling_fans() . . . 320
7.44.1.10 PDU_enable_motor_controller() . . . 320
7.44.1.11 PDU_enable_shutdown_circuit() . . . 320
7.44.1.12 PDU_speaker_chirp() . . . 321
7.44.2 Variable Documentation . . . 321
7.44.2.1 default_pdu_settings . . . 321
7.45 Core/Src/rb_tree.c File Reference . . . 321
7.45.1 Function Documentation . . . 322
7.45.1.1 checkBlackHeight() . . . 322
7.45.1.2 checkOrder() . . . 322
7.45.1.3 deleteRepair() . . . 323
7.45.1.4 destroyAllNodes() . . . 323
7.45.1.5 insertRepair() . . . 323
7.45.1.6 print() . . . 323
7.45.1.7 rbApplyNode() . . . 324
7.45.1.8 rbCheckBlackHeight() . . . 324

7.45.1.9 rbCheckOrder() . . . 324
7.45.1.10 rbCreate() . . . 324
7.45.1.11 rbDelete() . . . 325
7.45.1.12 rbDestroy() . . . 325
7.45.1.13 rbFind() . . . 326
7.45.1.14 rbInsert() . . . 326
7.45.1.15 rbPrint() . . . 327
7.45.1.16 rbSuccessor() . . . 327
7.45.1.17 rotateLeft() . . . 327
7.45.1.18 rotateRight() . . . 328
7.46 Core/Src/spi.c File Reference . . . 328
7.46.1 Detailed Description . . . 328
7.46.2 Function Documentation . . . 328
7.46.2.1 HAL_SPI_MspDeInit() . . . 329
7.46.2.2 HAL_SPI_MspInit() . . . 329
7.46.2.3 MX_SPI1_Init() . . . 329
7.46.3 Variable Documentation . . . 329
7.46.3.1 hspi1 . . . 329
7.47 Core/Src/stm32f4xx_hal_msp.c File Reference . . . 329
7.47.1 Detailed Description . . . 330
7.47.2 Function Documentation . . . 330
7.47.2.1 HAL_MspInit() . . . 330
7.48 Core/Src/stm32f4xx_hal_timebase_tim.c File Reference . . . 330
7.48.1 Detailed Description . . . 331
7.48.2 Function Documentation . . . 331
7.48.2.1 HAL_InitTick() . . . 331
7.48.2.2 HAL_ResumeTick() . . . 332
7.48.2.3 HAL_SuspendTick() . . . 332
7.48.3 Variable Documentation . . . 333
7.48.3.1 htim1 . . . 333
7.49 Core/Src/stm32f4xx_it.c File Reference . . . 333
7.49.1 Detailed Description . . . 334
7.49.2 Function Documentation . . . 334
7.49.2.1 ADC_IRQHandler() . . . 334
7.49.2.2 BusFault_Handler() . . . 334
7.49.2.3 CAN2_RX0_IRQHandler() . . . 335
7.49.2.4 CAN2_RX1_IRQHandler() . . . 335
7.49.2.5 CAN2_TX_IRQHandler() . . . 335
7.49.2.6 DebugMon_Handler() . . . 335
7.49.2.7 DMA2_Stream0_IRQHandler() . . . 336
7.49.2.8 EXTI0_IRQHandler() . . . 336
7.49.2.9 HardFault_Handler() . . . 336

7.49.2.10 MemManage_Handler() . . . 336
7.49.2.11 NMI_Handler() . . . 337
7.49.2.12 TIM1_UP_TIM10_IRQHandler() . . . 337
7.49.2.13 UsageFault_Handler() . . . 337
7.49.3 Variable Documentation . . . 337
7.49.3.1 hadc1 . . . 337
7.49.3.2 hadc2 . . . 338
7.49.3.3 hcan2 . . . 338
7.49.3.4 hdma_adc1 . . . 338
7.49.3.5 htim1 . . . 338
7.50 Core/Src/syscalls.c File Reference . . . 338
7.50.1 Detailed Description . . . 339
7.50.2 Function Documentation . . . 339
7.50.2.1 __attribute__() . . . 340
7.50.2.2 __io_getchar() . . . 340
7.50.2.3 __io_putchar() . . . 340
7.50.2.4 _close() . . . 340
7.50.2.5 _execve() . . . 340
7.50.2.6 _exit() . . . 341
7.50.2.7 _fork() . . . 341
7.50.2.8 _fstat() . . . 341
7.50.2.9 _getpid() . . . 341
7.50.2.10 _isatty() . . . 341
7.50.2.11 _kill() . . . 342
7.50.2.12 _link() . . . 342
7.50.2.13 _lseek() . . . 342
7.50.2.14 _open() . . . 342
7.50.2.15 _stat() . . . 342
7.50.2.16 _times() . . . 343
7.50.2.17 _unlink() . . . 343
7.50.2.18 _wait() . . . 343
7.50.2.19 initialise_monitor_handles() . . . 343
7.50.3 Variable Documentation . . . 343
7.50.3.1 environ . . . 343
7.51 Core/Src/sysmem.c File Reference . . . 344
7.51.1 Detailed Description . . . 344
7.51.2 Function Documentation . . . 344
7.51.2.1 _sbrk() . . . 344
7.51.3 Variable Documentation . . . 345
7.51.3.1 __sbrk_heap_end . . . 345
7.52 Core/Src/system_stm32f4xx.c File Reference . . . 345
7.52.1 Detailed Description . . . 346

7.53 Core/Src/temp_monitoring.c File Reference . . . 346
7.53.1 Function Documentation . . . 346
7.53.1.1 initTempMonitor() . . . 347
7.53.1.2 tempMonitorTask() . . . 347
7.53.1.3 testfunc() . . . 347
7.53.1.4 testfunc2() . . . 348
7.54 Core/Src/tim.c File Reference . . . 348
7.54.1 Detailed Description . . . 348
7.54.2 Function Documentation . . . 348
7.54.2.1 HAL_TIM_Base_MspDeInit() . . . 349
7.54.2.2 HAL_TIM_Base_MspInit() . . . 349
7.54.2.3 MX_TIM3_Init() . . . 349
7.54.3 Variable Documentation . . . 349
7.54.3.1 htim3 . . . 349
7.55 Core/Src/uvfr_settings.c File Reference . . . 349
7.55.1 Macro Definition Documentation . . . 352
7.55.1.1 DATA_MISMATCH . . . 352
7.55.1.2 DID_NOT_FINISH_PROGRAMMING . . . 352
7.55.1.3 FLASH_NOT_UNLOCKED . . . 352
7.55.1.4 FLASH_OK . . . 352
7.55.1.5 INVALID_SBLOCK . . . 352
7.55.1.6 LAPTOP_TO_VCU_ID . . . 353
7.55.1.7 POST_CHECKSUM . . . 353
7.55.1.8 PRE_CHECKSUM . . . 353
7.55.1.9 SRC_UVFR_SETTINGS_C_ . . . 353
7.55.1.10 VCU_TO_LAPTOP_ID . . . 353
7.55.2 Typedef Documentation . . . 353
7.55.2.1 helper_task_args . . . 353
7.55.2.2 setting_helper_task . . . 354
7.55.2.3 tx_all_settings_args . . . 354
7.55.2.4 tx_journal_args . . . 354
7.55.3 Function Documentation . . . 354
7.55.3.1 handleIncomingLaptopMsg() . . . 354
7.55.3.2 nukeSettings() . . . 354
7.55.3.3 sendAllSettingsWorker() . . . 355
7.55.3.4 sendJournalWorker() . . . 355
7.55.3.5 settingCopy() . . . 355
7.55.3.6 setupDefaultSettings() . . . 356
7.55.3.7 uvAwaitLaptopAckMsg() . . . 356
7.55.3.8 uvComputeChunkChecksum() . . . 356
7.55.3.9 uvComputeMemRegionChecksum() . . . 356
7.55.3.10 uvComputeSettingsChecksums() . . . 357

7.55.3.11 uvConfigSettingTask() . . . 357
7.55.3.12 uvCreateTmpSettingsCopy() . . . 357
7.55.3.13 uvForceDefaultReversionUponDeviceReset() . . . 357
7.55.3.14 uvLoadSettingsFromFlash() . . . 358
7.55.3.15 uvOverwriteCsr() . . . 358
7.55.3.16 uvResetFlashToDefault() [1/2] . . . 358
7.55.3.17 uvResetFlashToDefault() [2/2] . . . 358
7.55.3.18 uvSaveSettingsToFlash() . . . 359
7.55.3.19 uvSBlockCorruptionHandler() . . . 359
7.55.3.20 uvSendSettingGroup() . . . 359
7.55.3.21 uvSendSpecificParam() . . . 360
7.55.3.22 uvSetSettingResponseReminder() . . . 360
7.55.3.23 uvSettingsInit() . . . 360
7.55.3.24 uvSettingsProgrammerTask() . . . 361
7.55.3.25 uvTransmitVehicleStatus() . . . 361
7.55.3.26 uvUpdateTmpSettings() . . . 361
7.55.3.27 uvValidateChecksums() . . . 362
7.55.3.28 uvValidateFlashSettings() . . . 362
7.55.4 Variable Documentation . . . 362
7.55.4.1 _e_uvdata . . . 362
7.55.4.2 _s_uvdata . . . 362
7.55.4.3 child_task_active . . . 363
7.55.4.4 child_task_handle . . . 363
7.55.4.5 current_vehicle_settings . . . 363
7.55.4.6 default_bms_settings . . . 363
7.55.4.7 default_daq_settings . . . 363
7.55.4.8 default_datapoints . . . 364
7.55.4.9 default_dl_settings . . . 364
7.55.4.10 default_imd_settings . . . 364
7.55.4.11 default_pdu_settings . . . 364
7.55.4.12 default_vehicle . . . 364
7.55.4.13 is_laptop_connected . . . 365
7.55.4.14 last_contact_with_laptop . . . 365
7.55.4.15 mc_default_settings . . . 365
7.55.4.16 ptask_handle . . . 365
7.55.4.17 settings_queue . . . 365
7.55.4.18 vcu_ack_failed_msg . . . 366
7.55.4.19 vcu_ack_msg . . . 366
7.56 Core/Src/uvfr_state_engine.c File Reference . . . 366
7.56.1 Detailed Description . . . 368
7.56.2 Macro Definition Documentation . . . 369
7.56.2.1 UVFR_STATE_MACHINE_IMPLIMENTATION . . . 369

7.57 Core/Src/uvfr_utils.c File Reference 369
7.57.1 Macro Definition Documentation 370
7.57.1.1 UV_UTILS_SRC_IMPLIMENTATION 370
7.57.2 Function Documentation 370
7.57.2.1 __uvFreeCritSection() 370
7.57.2.2 __uvFreeOS() 370
7.57.2.3 __uvInitPanic() 371
7.57.2.4 __uvMallocCritSection() 371
7.57.2.5 __uvMallocOS() 371
7.57.2.6 setup_extern_devices() 371
7.57.2.7 uvInit() 372
7.57.2.8 uvIsPTRValid() 374
7.57.2.9 uvSysResetDaemon() 374
7.57.2.10 uvUtilsReset() 374
7.57.3 Variable Documentation 375
7.57.3.1 data_size 375
7.57.3.2 init_task_handle 375
7.57.3.3 reset_handle 375
7.57.3.4 TxData 375
7.58 Core/Src/uvfr_vehicle_commands.c File Reference 375
7.58.1 Function Documentation 376
7.58.1.1 uvSecureVehicle() 376

Index **377**

Chapter 1

Deprecated List

Global rbCheckBlackHeight (rbtree *rbt)

Leftovers from laptop unit tests.

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

Leftovers from laptop unit tests.

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

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

Global setup_extern_devices (void *argument)

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

Global uvForceDefaultReversionUponDeviceReset ()

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

State Engine . . . 9
State Engine API . . . 16
State Engine Internals . . . 30
UVFR Utilities . . . 42
Utility Macros . . . 43
UVFR Vehicle Commands . . . 49
UVFR CANbus API . . . 50
CMSIS . . . 51
Stm32f4xx_system . . . 52
STM32F4xx_System_Private_Includes . . . 53
STM32F4xx_System_Private_TypesDefinitions . . . 54
STM32F4xx_System_Private_Defines . . . 55
STM32F4xx_System_Private_Macros . . . 56
STM32F4xx_System_Private_Variables . . . 57
STM32F4xx_System_Private_FunctionPrototypes . . . 58
STM32F4xx_System_Private_Functions . . . 59

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

access_control_info . . . 61
bms_settings_t . . . 62
CAN_Callback . . . 63
daq_child_task . . . 64
daq_datapoint . . . 65
daq_loop_args
This struct holds info of what needs to be logged . . . 66
daq_param_list_node . . . 68
driving_loop_args . . . 69
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 75
drivingMode
This is where the driving mode and the drivingModeParams are at . . . 75
drivingModeParams
This struct is designed to hold information about each drivingmode's map params . . . 77
exp_torque_map_args
Struct to hold parameters used in an exponential torque map . . . 77
helper_task_args . . . 78
linear_torque_map_args . . . 79
motor_controller_settings . . . 80
p_status . . . 81
rbnode
Node of a Red-Black binary search tree . . . 82
rbtree
Struct representing a binary search tree . . . 84
s_curve_torque_map_args
Struct for s-curve parameters for torque . . . 86
setting_helper_task . . . 87
state_change_daemon_args . . . 88
task_management_info
Struct to contain data about a parent task . . . 89
task_status_block
Information about the task . . . 90

tx_all_settings_args . . . 91
tx_journal_args . . . 92
uv19_pdu_settings . . . 92
uv_binary_semaphore_info . . . 94
uv_CAN_msg
Representative of a CAN message . . . 95
uv_imd_settings . . . 96
uv_init_struct . . . 97
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 . . . 98
uv_init_task_response
Struct representing the response of one of the initialization tasks . . . 99
uv_internal_params
Data used by the uvfr_utils library to do what it needs to do :) . . . 101
uv_mutex_info . . . 102
uv_os_settings
Settings that dictate state engine behavior . . . 102
uv_persistant_data_frame . . . 105
uv_scd_response . . . 106
uv_semaphore_info . . . 106
uv_task_info
This struct is designed to hold neccessary information about an RTOS task that will be managed by uvfr_state_engine . . . 107
uv_task_msg_t
Struct containing a message between two tasks . . . 113
uv_vehicle_settings . . . 115
veh_gen_info . . . 117

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 . . . 121
Core/Inc/bms.h . . . 125
Core/Inc/can.h
: This file contains all the function prototypes for the can.c file . . . 127
Core/Inc/constants.h . . . 131
Core/Inc/daq.h . . . 133
Core/Inc/dash.h . . . 137
Core/Inc/dma.h
: This file contains all the function prototypes for the dma.c file . . . 139
Core/Inc/driving_loop.h . . . 139
Core/Inc/errorLUT.h . . . 145
Core/Inc/FreeRTOSConfig.h . . . 145
Core/Inc/gpio.h
: This file contains all the function prototypes for the gpio.c file . . . 157
Core/Inc/imd.h . . . 158
Core/Inc/main.h
: Header for main.c file. This file contains the common defines of the application . . . 166
Core/Inc/motor_controller.h . . . 169
Core/Inc/odometer.h . . . 178
Core/Inc/oled.h . . . 180
Core/Inc/pdu.h . . . 181
Core/Inc/rb_tree.h . . . 185
Core/Inc/spi.h
: This file contains all the function prototypes for the spi.c file . . . 194
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 . . . 196
Core/Inc/stm32f4xx_it.h
: This file contains the headers of the interrupt handlers . . . 216
Core/Inc/temp_monitoring.h . . . 220
Core/Inc/tim.h
: This file contains all the function prototypes for the tim.c file . . . 221
Core/Inc/uvfr_global_config.h . . . 222
Core/Inc/uvfr_settings.h . . . 224

Core/Inc/uvfr_state_engine.h . . . 238
Core/Inc/uvfr_utils.h . . . 244
Core/Inc/uvfr_vehicle_commands.h . . . 257
Core/Src/adc.c
This file provides code for the configuration of the ADC instances . . . 260
Core/Src/bms.c . . . 265
Core/Src/can.c
This file provides code for the configuration of the CAN instances . . . 266
Core/Src/constants.c . . . 273
Core/Src/daq.c . . . 274
Core/Src/dash.c . . . 281
Core/Src/dma.c
This file provides code for the configuration of all the requested memory to memory DMA transfers . . . 282
Core/Src/driving_loop.c
File containing the meat and potatoes driving loop thread, and all supporting functions . . . 283
Core/Src/freertos.c . . . 290
Core/Src/gpio.c
This file provides code for the configuration of all used GPIO pins . . . 295
Core/Src/imd.c . . . 296
Core/Src/main.c
: Main program body . . . 307
Core/Src/motor_controller.c . . . 313
Core/Src/odometer.c . . . 316
Core/Src/oled.c . . . 317
Core/Src/pdu.c . . . 318
Core/Src/rb_tree.c . . . 321
Core/Src/spi.c
This file provides code for the configuration of the SPI instances . . . 328
Core/Src/stm32f4xx_hal_msp.c
This file provides code for the MSP Initialization and de-Initialization codes . . . 329
Core/Src/stm32f4xx_hal_timebase_tim.c
HAL time base based on the hardware TIM . . . 330
Core/Src/stm32f4xx_it.c
Interrupt Service Routines . . . 333
Core/Src/syscalls.c
STM32CubeIDE Minimal System calls file . . . 338
Core/Src/sysmem.c
STM32CubeIDE System Memory calls file . . . 344
Core/Src/system_stm32f4xx.c
CMSIS Cortex-M4 Device Peripheral Access Layer System Source File . . . 345
Core/Src/temp_monitoring.c . . . 346
Core/Src/tim.c
This file provides code for the configuration of the TIM instances . . . 348
Core/Src/uvfr_settings.c . . . 349
Core/Src/uvfr_state_engine.c
File containing the implementation of the vehicle's state engine and error handling infrastructure 366
Core/Src/uvfr_utils.c . . . 369
Core/Src/uvfr_vehicle_commands.c . . . 375

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)
- void uvTaskPeriodEnd (uv_task_info *t)

  Function called at the end of the task period.

Variables

- static uv_task_id _next_task_id = 0
- static uv_task_info * _task_register = NULL
- static TickType_t * last_task_start_times = NULL
- static TickType_t * last_task_end_times = NULL
- uint8_t * task_tardiness
- static uv_task_id _next_svc_task_id = 0
- TaskHandle_t * scd_handle_ptr
- static volatile bool SCD_active = false
- static volatile bool throttle_daq = false
- static volatile bool nc_throttling = 1
- static QueueHandle_t state_change_queue = NULL
- rbtree * task_name_lut = NULL
- uint32_t error_bitfield [4]
- enum uv_vehicle_state_t vehicle_state = UV_BOOT
- enum uv_vehicle_state_t previous_state = UV_BOOT
- uv_task_info * task_manager = NULL
- uv_task_info * svc_task_manager = NULL
- rbtree * task_name_tree
- uv_os_settings * os_settings = NULL
- uv_os_settings default_os_settings

5.1.1 Detailed Description

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

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

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

5.1.2 Macro Definition Documentation

5.1.2.1 MAX_NUM_MANAGED_TASKS

```
#define MAX_NUM_MANAGED_TASKS 16
```

Definition at line 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 1397 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 1350 of file uvfr_state_engine.c.

References __uvInitPanic(), _task_register, and task_management_info::task_handle.

Referenced by uvStartStateMachine().

5.1.4.3 uvTaskPeriodEnd()

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

Function called at the end of the task period.

@

Definition at line 1454 of file uvfr_state_engine.c.

References uv_task_info::last_execution_time, nc_throttling, uv_task_info::task_id, uv_task_info::task_period, task_tardiness, uvTaskResetDelayBit, and uvTaskSetDelayBit.

Referenced by uvTaskManager().

5.1.5 Variable Documentation

5.1.5.1 _next_svc_task_id

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

Definition at line 32 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(), uvTaskManager(), and uvValidateManagedTasks().

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(), uvSVCTaskManager(), and uvTaskManager().

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,
        .min_task_period = 3,
        .task_overshoot_margin_noncrit = 1.5F,
        .task_overshoot_margin_crit = 1.1F
}
```

Definition at line 57 of file uvfr_state_engine.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

5.1.5.5 error_bitfield

```
uint32_t error_bitfield[4]
```

Definition at line 45 of file uvfr_state_engine.c.

5.1.5.6 last_task_end_times

```
TickType_t* last_task_end_times = NULL  [static]
```

Definition at line 29 of file uvfr_state_engine.c.

Referenced by uvTaskManager().

5.1.5.7 last_task_start_times

```
TickType_t* last_task_start_times = NULL  [static]
```

Definition at line 28 of file uvfr_state_engine.c.

Referenced by uvTaskManager().

5.1.5.8 nc_throttling

```
volatile bool nc_throttling = 1  [static]
```

Definition at line 40 of file uvfr_state_engine.c.

Referenced by uvTaskManager(), and uvTaskPeriodEnd().

5.1.5.9 os_settings

```
uv_os_settings* os_settings = NULL
```

Definition at line 55 of file uvfr_state_engine.c.

Referenced by uvStartStateMachine(), and uvTaskManager().

5.1.5.10 previous_state

```
enum uv_vehicle_state_t previous_state = UV_BOOT
```

Definition at line 48 of file uvfr_state_engine.c.

Referenced by changeVehicleState(), and uvStartStateMachine().

5.1.5.11 SCD_active

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

Definition at line 38 of file uvfr_state_engine.c.

Referenced by _stateChangeDaemon().

5.1.5.12 scd_handle_ptr

```
TaskHandle_t* scd_handle_ptr
```

Definition at line 35 of file uvfr_state_engine.c.

5.1.5.13 state_change_queue

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

Definition at line 41 of file uvfr_state_engine.c.

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

5.1.5.14 svc_task_manager

```
uv_task_info* svc_task_manager = NULL
```

Definition at line 51 of file uvfr_state_engine.c.

Referenced by uvInitStateEngine(), and uvStartStateMachine().

5.1.5.15 task_manager

```
uv_task_info* task_manager = NULL
```

Definition at line 50 of file uvfr_state_engine.c.

Referenced by uvInitStateEngine(), and uvStartStateMachine().

5.1.5.16 task_name_lut

```
rbtree* task_name_lut = NULL
```

Definition at line 43 of file uvfr_state_engine.c.

5.1.5.17 task_name_tree

```
rbtree* task_name_tree
```

Definition at line 53 of file uvfr_state_engine.c.

5.1.5.18 task_tardiness

```
uint8_t* task_tardiness
```

Definition at line 30 of file uvfr_state_engine.c.

Referenced by uvLateTaskHandler(), uvTaskManager(), and uvTaskPeriodEnd().

5.1.5.19 throttle_daq

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

Definition at line 39 of file uvfr_state_engine.c.

5.1.5.20 vehicle_state

```
enum uv_vehicle_state_t vehicle_state = UV_BOOT
```

Definition at line 47 of file uvfr_state_engine.c.

Referenced by _stateChangeDaemon(), changeVehicleState(), handleIncomingLaptopMsg(), testfunc(), and uv↩ StartStateMachine().

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.
- enum os_flag { UV_OS_LOG_MEM = 0x01, UV_OS_LOG_TASK_END_TIME = 0x02, UV_OS_ATTEMPT_RESTART_NC_TAS
  = 0x04, UV_OS_ENABLE_NONCRIT_TASK_THROTTLE = 0x08 }

Functions

- void uvTaskPeriodEnd (uv_task_info *t)

  Function called at the end of the task period.
- uv_status changeVehicleState (uint16_t state)

  Function for changing the state of the vehicle, as well as the list of active + inactive tasks.
- uv_status 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 204 of file uvfr_state_engine.h.

5.2.2.2 UV_TASK_DEADLINE_FIRM

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

Definition at line 208 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 209 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 210 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 206 of file uvfr_state_engine.h.

5.2.2.6 UV_TASK_DEFER_DELETION

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

Definition at line 205 of file uvfr_state_engine.h.

5.2.2.7 UV_TASK_DELAYING

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

Definition at line 212 of file uvfr_state_engine.h.

5.2.2.8 UV_TASK_DORMANT_SVC

```
#define UV_TASK_DORMANT_SVC 0b0000000000000011
```

Definition at line 194 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 203 of file uvfr_state_engine.h.

5.2.2.10 UV_TASK_GENERIC_SVC

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

Definition at line 195 of file uvfr_state_engine.h.

5.2.2.11 UV_TASK_IS_CHILD

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

Definition at line 201 of file uvfr_state_engine.h.

5.2.2.12 UV_TASK_IS_ORPHAN

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

Definition at line 202 of file uvfr_state_engine.h.

5.2.2.13 UV_TASK_IS_PARENT

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

Definition at line 200 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 198 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 197 of file uvfr_state_engine.h.

5.2.2.16 UV_TASK_MANAGER_MASK

```
#define UV_TASK_MANAGER_MASK 0b0000000000000011
```

Definition at line 196 of file uvfr_state_engine.h.

5.2.2.17 UV_TASK_MISSION_CRITICAL

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

Definition at line 211 of file uvfr_state_engine.h.

5.2.2.18 UV_TASK_PERIODIC_SVC

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

Definition at line 193 of file uvfr_state_engine.h.

5.2.2.19 UV_TASK_PRIO_INCREMENTATION

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

Definition at line 207 of file uvfr_state_engine.h.

5.2.2.20 UV_TASK_SCD_IGNORE

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

Definition at line 199 of file uvfr_state_engine.h.

5.2.2.21 UV_TASK_VEHICLE_APPLICATION

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

Definition at line 192 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 288 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 309 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 281 of file uvfr_state_engine.h.

5.2.2.25 uvTaskResetDelayBit

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

Definition at line 279 of file uvfr_state_engine.h.

5.2.2.26 uvTaskResetDeletionBit

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

Definition at line 275 of file uvfr_state_engine.h.

5.2.2.27 uvTaskSetDelayBit

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

Definition at line 277 of file uvfr_state_engine.h.

5.2.2.28 uvTaskSetDeletionBit

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

Definition at line 274 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 os_flag

```
enum os_flag
```

Enumerator

UV_OS_LOG_MEM	
UV_OS_LOG_TASK_END_TIME	
UV_OS_ATTEMPT_RESTART_NC_TASK	
UV_OS_ENABLE_NONCRIT_TASK_THROTTLE	

Definition at line 167 of file uvfr_state_engine.h.

5.2.4.2 task_priority

```
enum task_priority
```

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

Enumerator

IDLE_TASK_PRIORITY	
LOW_PRIORITY	
BELOW_NORMAL	
MEDIUM_PRIORITY	
ABOVE_NORMAL	
HIGH_PRIORITY	
REALTIME_PRIORITY	

Definition at line 135 of file uvfr_state_engine.h.

5.2.4.3 uv_scd_response_e

enum uv_scd_response_e

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

Enumerator

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

Definition at line 106 of file uvfr_state_engine.h.

5.2.4.4 uv_task_cmd_e

enum uv_task_cmd_e

Special commands used to start and shutdown tasks.

Enumerator

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

Definition at line 96 of file uvfr_state_engine.h.

5.2.4.5 uv_task_state_t

enum uv_task_state_t

Enum representing the state of a managed task.

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

Enumerator

UV_TASK_NOT_STARTED	
UV_TASK_DELETED	
UV_TASK_RUNNING	
UV_TASK_SUSPENDED	

Definition at line 124 of file uvfr_state_engine.h.

5.2.4.6 uv_vehicle_state_t

```
enum uv_vehicle_state_t
```

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

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

Enumerator

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

Definition at line 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 101 of file uvfr_state_engine.c.

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

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

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 272 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(), initTempMonitor(), and uvConfigSettingTask().

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 262 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 171 of file uvfr_state_engine.c.

References __uvInitPanic(), _task_register, initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), MAX_NUM_MANAGED_TASKS, svc_task_manager, task_manager, UV_OK, uvConfigSettingTask(), 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 200 of file uvfr_state_engine.c.

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

Referenced by uvInit().

5.2.5.6 uvTaskPeriodEnd()

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

Function called at the end of the task period.

@

Definition at line 1454 of file uvfr_state_engine.c.

References uv_task_info::last_execution_time, nc_throttling, uv_task_info::task_id, uv_task_info::task_period, task_tardiness, uvTaskResetDelayBit, and uvTaskSetDelayBit.

Referenced by uvTaskManager().

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 uvThrottleNonCritTasks ()
- void uvLateTaskHandler (uv_task_info *t, TickType_t tdiff, uint8_t task_tardiness)
- uv_task_info * uvCreateServiceTask ()

  Create a new service task, because fuck you, thats why.
- uv_status uvStartSVCTask (uv_task_info *t)

  Function to start a service task specifically.
- uv_status uvSuspendSVCTask (uv_task_info *t)

  Function that suspends a service task.
- uv_status uvDeleteSVCTask (uv_task_info *t)

  For when you need to delete a service task... for some reason...
- uv_status uvRestartSVCTask (uv_task_info *t)

  Function that takes a service part that may be messed up and tries to reboot it to recover.
- uv_task_info * uvGetTaskFromName (char *tsk_name)
- uv_task_info * uvGetTaskFromRTOSHandle (TaskHandle_t t_handle)

  Returns the pointer to the task info structure.
- 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 695 of file uvfr_state_engine.c.

References changeVehicleState(), UV_ERROR_STATE, and 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 874 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 331 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 318 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 460 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 717 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(), tempMonitorTask(), uvSettingsProgrammerTask(), and uvTaskManager().

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 801 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 758 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(), tempMonitorTask(), and uvTaskManager().

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 566 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 1204 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 uvInit(), and uvInitStateEngine().

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 1298 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 506 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 1433 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 1445 of file uvfr_state_engine.c.

5.3.2.15 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 481 of file uvfr_state_engine.c.

References UV_OK.

Referenced by uvRestartSVCTask().

5.3.2.16 uvLateTaskHandler()

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

Definition at line 1055 of file uvfr_state_engine.c.

References uv_task_info::task_flags, uv_task_info::task_period, task_tardiness, and UV_TASK_MISSION_CRIT↩ ICAL.

Referenced by uvTaskManager().

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 1326 of file uvfr_state_engine.c.

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

5.3.2.18 uvScheduleTaskDeletion()

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

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

Definition at line 578 of file uvfr_state_engine.c.

References uv_task_info::task_flags, uv_task_info::task_id, uv_task_info::task_state, UV_ABORTED, UV_ERROR, UV_OK, UV_TASK_AWAITING_DELETION, and UV_TASK_DELETED.

Referenced by _stateChangeDaemon().

5.3.2.19 uvSendTaskStatusReport()

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

Definition at line 858 of file uvfr_state_engine.c.

5.3.2.20 uvStartSVCTask()

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

Function to start a service task specifically.

Definition at line 1244 of file uvfr_state_engine.c.

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

Referenced by uvRestartSVCTask().

5.3.2.21 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 388 of file uvfr_state_engine.c.

References _BV_32, uv_task_info::last_execution_time, uv_task_info::stack_size, uv_task_info::task_function, uv_task_info::task_handle, uv_task_info::task_id, uv_task_info::task_name, uv_task_info::task_priority, uv_task_↩ info::task_state, UV_ABORTED, UV_ERROR, UV_OK, UV_TASK_RUNNING, and UV_TASK_SUSPENDED.

Referenced by _stateChangeDaemon(), and uvInit().

5.3.2.22 uvSuspendSVCTask()

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

Function that suspends a service task.

Definition at line 1283 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 605 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 662 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.

Definition at line 1089 of file uvfr_state_engine.c.

References _next_task_id, _task_register, uv_task_info::cmd_data, killSelf(), uv_task_info::last_execution_time, last_task_end_times, last_task_start_times, MAX_NUM_MANAGED_TASKS, nc_throttling, os_settings, task_management_info::parent_msg_queue, suspendSelf(), uv_task_info::task_flags, task_management_info::task_handle, uv_task_info::task_handle, uv_os_settings::task_overshoot_margin_crit, uv_os_settings::task_overshoot_margin_noncrit, uv_task_info::task_period, uv_task_info::task_state, task_tardiness, uv_task_info::tmi, UV_INVALID_MSG, UV_KILL_CMD, UV_SUSPEND_CMD, UV_TASK_MANAGER_MASK, UV_TASK_MISSION_CRITICAL, UV_TASK_RUNNING, UV_TASK_VEHICLE_APPLICATION, uvLateTaskHandler(), and uvTaskPeriodEnd().

Referenced by uvStartStateMachine().

5.3.2.26 uvThrottleNonCritTasks()

```
uv_status uvThrottleNonCritTasks ( )
```

Definition at line 1050 of file uvfr_state_engine.c.

5.3.2.27 uvValidateManagedTasks()

```
uv_status uvValidateManagedTasks ( )
```

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

Definition at line 366 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 MC_Startup(), tempMonitorTask(), and uvSettingsInit().

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 handleIncomingLaptopMsg(), MC_Request_Data(), MC_Set_Param(), MotorControllerSpinTest(), sendDaqMsg(), testfunc2(), uvSendSpecificParam(), uvSettingsInit(), and uvSettingsProgrammerTask().

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 BMS_CAN_timeout
- uint32_t max_temp

6.2.1 Detailed Description

Definition at line 13 of file bms.h.

6.2.2 Field Documentation

6.2.2.1 BMS_CAN_timeout

```
uint32_t bms_settings_t::BMS_CAN_timeout
```

Definition at line 14 of file bms.h.

6.2.2.2 max_temp

```
uint32_t bms_settings_t::max_temp
```

Definition at line 15 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

Data Fields

- struct daq_child_task * next_task
- TaskHandle_t meta_task_handle
- uint32_t period
- daq_param_list_node * param_list

6.4.1 Detailed Description

Definition at line 19 of file daq.c.

6.4.2 Field Documentation

6.4.2.1 meta_task_handle

```
TaskHandle_t daq_child_task::meta_task_handle
```

Definition at line 21 of file daq.c.

Referenced by startDaqSubTasks(), and stopDaqSubTasks().

6.4.2.2 next_task

```
struct daq_child_task* daq_child_task::next_task
```

Definition at line 20 of file daq.c.

Referenced by insertParamToRegister(), and startDaqSubTasks().

6.4.2.3 param_list

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

Definition at line 23 of file daq.c.

Referenced by daqSubTask(), and insertParamToRegister().

6.4.2.4 period

```
uint32_t daq_child_task::period
```

Definition at line 22 of file daq.c.

Referenced by daqSubTask(), and insertParamToRegister().

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

- Core/Src/daq.c

6.5 daq_datapoint Struct Reference

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

Data Fields

- uint32_t can_id
- uint16_t param
- uint8_t period
- uint8_t type

6.5.1 Detailed Description

Definition at line 73 of file daq.h.

6.5.2 Field Documentation

6.5.2.1 can_id

```
uint32_t daq_datapoint::can_id
```

Definition at line 74 of file daq.h.

Referenced by insertParamToRegister().

6.5.2.2 param

`uint16_t daq_datapoint::param`

Which loggable param are we logging boys?

Definition at line 75 of file daq.h.

Referenced by insertParamToRegister().

6.5.2.3 period

`uint8_t daq_datapoint::period`

Time between transmissions in ms

Definition at line 76 of file daq.h.

Referenced by insertParamToRegister().

6.5.2.4 type

`uint8_t daq_datapoint::type`

Datatype of the data

Definition at line 77 of file daq.h.

Referenced by insertParamToRegister().

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

- Core/Inc/daq.h

6.6 daq_loop_args Struct Reference

This struct holds info of what needs to be logged.

`#include <daq.h>`

Data Fields

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

6.6.1 Detailed Description

This struct holds info of what needs to be logged.

Definition at line 86 of file daq.h.

6.6.2 Field Documentation

6.6.2.1 can_channel

```
uint8_t daq_loop_args::can_channel
```

Definition at line 90 of file daq.h.

6.6.2.2 daq_child_priority

```
uint8_t daq_loop_args::daq_child_priority
```

Definition at line 91 of file daq.h.

Referenced by initDaqTask(), and startDaqSubTasks().

6.6.2.3 minimum_daq_period

```
uint8_t daq_loop_args::minimum_daq_period
```

Definition at line 89 of file daq.h.

6.6.2.4 throttle_daq_to_preserve_performance

```
uint8_t daq_loop_args::throttle_daq_to_preserve_performance
```

Definition at line 88 of file daq.h.

6.6.2.5 total_params_logged

```
uint16_t daq_loop_args::total_params_logged
```

Definition at line 87 of file daq.h.

Referenced by configureDaqSubTasks(), and uvResetFlashToDefault().

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

- Core/Inc/daq.h

6.7 daq_param_list_node Struct Reference

Data Fields

- struct daq_param_list_node * next
- uint32_t can_id
- uint16_t param
- uint8_t size

6.7.1 Detailed Description

Definition at line 10 of file daq.c.

6.7.2 Field Documentation

6.7.2.1 can_id

```
uint32_t daq_param_list_node::can_id
```

Definition at line 13 of file daq.c.

Referenced by insertParamToRegister(), and sendDaqMsg().

6.7.2.2 next

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

Definition at line 11 of file daq.c.

Referenced by insertParamToParamList(), insertParamToRegister(), and sendAllParamsFromList().

6.7.2.3 param

```
uint16_t daq_param_list_node::param
```

Definition at line 14 of file daq.c.

Referenced by insertParamToRegister(), and sendDaqMsg().

6.7.2.4 size

```
uint8_t daq_param_list_node::size
```

Definition at line 16 of file daq.c.

Referenced by insertParamToRegister(), and sendDaqMsg().

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

- Core/Src/daq.c

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_apps1_value
- uint16_t min_apps1_value
- uint16_t min_apps2_value
- uint16_t max_apps2_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.

Referenced by mapThrottleToTorque().

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 141 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 131 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 137 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 133 of file driving_loop.h.

6.8.2.9 apps_top

```
uint16_t driving_loop_args::apps_top
```

Max APPS input value, representing 100% throttle

Definition at line 130 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 136 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 134 of file driving_loop.h.

6.8.2.12 dmodes

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

These are various driving modes

Definition at line 144 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_apps1_value

```
uint16_t driving_loop_args::max_apps1_value
```

for detecting disconnects and short circuits

Definition at line 123 of file driving_loop.h.

Referenced by calculateThrottlePercentage().

6.8.2.15 max_apps2_value

```
uint16_t driving_loop_args::max_apps2_value
```

Definition at line 126 of file driving_loop.h.

Referenced by calculateThrottlePercentage().

6.8.2.16 max_apps_offset

```
uint16_t driving_loop_args::max_apps_offset
```

maximum APPS offset

Definition at line 121 of file driving_loop.h.

6.8.2.17 max_BPS_value

```
uint16_t driving_loop_args::max_BPS_value
```

are the brakes valid?

Definition at line 128 of file driving_loop.h.

6.8.2.18 min_apps1_value

```
uint16_t driving_loop_args::min_apps1_value
```

for detecting disconnects and short circuits

Definition at line 124 of file driving_loop.h.

Referenced by calculateThrottlePercentage().

6.8.2.19 min_apps2_value

```
uint16_t driving_loop_args::min_apps2_value
```

for detecting disconnects and short circuits

Definition at line 125 of file driving_loop.h.

6.8.2.20 min_apps_offset

`uint16_t driving_loop_args::min_apps_offset`

minimum APPS offset

Definition at line 120 of file driving_loop.h.

6.8.2.21 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.

6.8.2.22 min_BPS_value

`uint16_t driving_loop_args::min_BPS_value`

are the brakes valid?

Definition at line 127 of file driving_loop.h.

6.8.2.23 num_driving_modes

`uint8_t driving_loop_args::num_driving_modes`

How many modes are actually populated

Definition at line 139 of file driving_loop.h.

6.8.2.24 period

`uint8_t driving_loop_args::period`

how often does the driving loop execute

Definition at line 140 of file driving_loop.h.

6.8.2.25 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 helper_task_args Union Reference

Data Fields

- tx_all_settings_args setting_tx_args
- tx_journal_args journal_tx_args

6.13.1 Detailed Description

Definition at line 51 of file uvfr_settings.c.

6.13.2 Field Documentation

6.13.2.1 journal_tx_args

`tx_journal_args helper_task_args::journal_tx_args`

Definition at line 53 of file uvfr_settings.c.

6.13.2.2 setting_tx_args

`tx_all_settings_args helper_task_args::setting_tx_args`

Definition at line 52 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

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

- Core/Src/uvfr_settings.c

6.14 linear_torque_map_args Struct Reference

`#include <driving_loop.h>`

Data Fields

- int32_t offset
- float slope

6.14.1 Detailed Description

Definition at line 48 of file driving_loop.h.

6.14.2 Field Documentation

6.14.2.1 offset

```
int32_t linear_torque_map_args::offset
```

Definition at line 49 of file driving_loop.h.

6.14.2.2 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

6.15 motor_controller_settings Struct Reference

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

Data Fields

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

6.15.1 Detailed Description

Definition at line 161 of file uvfr_settings.h.

6.15.2 Field Documentation

6.15.2.1 can_id_rx

```
uint32_t motor_controller_settings::can_id_rx
```

Definition at line 165 of file uvfr_settings.h.

6.15.2.2 can_id_tx

```
uint32_t motor_controller_settings::can_id_tx
```

Definition at line 163 of file uvfr_settings.h.

Referenced by MC_Request_Data(), MC_Set_Param(), and MotorControllerSpinTest().

6.15.2.3 integral_memory_max

```
uint8_t motor_controller_settings::integral_memory_max
```

Definition at line 174 of file uvfr_settings.h.

6.15.2.4 integral_time_constant

```
uint32_t motor_controller_settings::integral_time_constant
```

Definition at line 172 of file uvfr_settings.h.

6.15.2.5 mc_CAN_timeout

```
uint32_t motor_controller_settings::mc_CAN_timeout
```

Definition at line 167 of file uvfr_settings.h.

6.15.2.6 proportional_gain

```
uint8_t motor_controller_settings::proportional_gain
```

Definition at line 170 of file uvfr_settings.h.

Referenced by MC_Startup(), and Parse_Bamocar_Response().

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

- Core/Inc/uvfr_settings.h

6.16 p_status Struct Reference

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

Data Fields

- uv_status peripheral_status
- TickType_t activation_time

6.16.1 Detailed Description

Definition at line 317 of file uvfr_utils.h.

6.16.2 Field Documentation

6.16.2.1 activation_time

```
TickType_t p_status::activation_time
```

Definition at line 319 of file uvfr_utils.h.

6.16.2.2 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

6.17 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

6.17.1 Detailed Description

Node of a Red-Black binary search tree.

Definition at line 27 of file rb_tree.h.

6.17.2 Field Documentation

6.17.2.1 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().

6.17.2.2 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(), rbApplyNode(), rbCreate(), rbDelete(), rbFind(), and rbInsert().

6.17.2.3 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(), rbApplyNode(), rbCreate(), rbDelete(), rbFind(), rbInsert(), rbSuccessor(), rotateLeft(), and rotateRight().

6.17.2.4 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(), rbInsert(), rbSuccessor(), rotateLeft(), and rotateRight().

6.17.2.5 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(), rbApplyNode(), rbCreate(), rbDelete(), rbFind(), rbInsert(), rbSuccessor(), rotateLeft(), and rotateRight().

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

- Core/Inc/rb_tree.h

6.18 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

6.18.1 Detailed Description

struct representing a binary search tree

Definition at line 39 of file rb_tree.h.

6.18.2 Field Documentation

6.18.2.1 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().

6.18.2.2 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().

6.18.2.3 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().

6.18.2.4 min

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

Pointer to minimum element

Definition at line 50 of file rb_tree.h.

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

6.18.2.5 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().

6.18.2.6 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.

6.18.2.7 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

6.19 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]

6.19.1 Detailed Description

struct for s-curve parameters for torque

Definition at line 66 of file driving_loop.h.

6.19.2 Field Documentation

6.19.2.1 a

```
int32_t s_curve_torque_map_args::a
```

Definition at line 67 of file driving_loop.h.

6.19.2.2 b

```
int32_t s_curve_torque_map_args::b
```

Definition at line 68 of file driving_loop.h.

6.19.2.3 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

6.20 setting_helper_task Struct Reference

Data Fields

- TaskHandle_t meta_task_handle
- uint32_t status
- helper_task_args args

6.20.1 Detailed Description

Definition at line 59 of file uvfr_settings.c.

6.20.2 Field Documentation

6.20.2.1 args

```
helper_task_args setting_helper_task::args
```

Definition at line 65 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

6.20.2.2 meta_task_handle

```
TaskHandle_t setting_helper_task::meta_task_handle
```

Definition at line 60 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

6.20.2.3 status

```
uint32_t setting_helper_task::status
```

Definition at line 62 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

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

- Core/Src/uvfr_settings.c

6.21 state_change_daemon_args Struct Reference

Data Fields

- TaskHandle_t meta_task_handle

6.21.1 Detailed Description

Definition at line 73 of file uvfr_state_engine.c.

6.21.2 Field Documentation

6.21.2.1 meta_task_handle

```
TaskHandle_t state_change_daemon_args::meta_task_handle
```

Definition at line 74 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

6.22 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

6.22.1 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.

6.22.2 Field Documentation

6.22.2.1 parent_msg_queue

```
QueueHandle_t task_management_info::parent_msg_queue
```

Definition at line 156 of file uvfr_state_engine.h.

Referenced by uvTaskManager().

6.22.2.2 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(), and uvTaskManager().

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

- Core/Inc/uvfr_state_engine.h

6.23 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

6.23.1 Detailed Description

Information about the task.

Definition at line 162 of file uvfr_state_engine.h.

6.23.2 Field Documentation

6.23.2.1 task_high_water_mark

```
uint32_t task_status_block::task_high_water_mark
```

Definition at line 163 of file uvfr_state_engine.h.

6.23.2.2 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

6.24 tx_all_settings_args Struct Reference

Data Fields

- uint8_t * sblock_origin

6.24.1 Detailed Description

These are arguments passed to the "Transmit All Settings Over CANbus" Subroutine.

Definition at line 35 of file uvfr_settings.c.

6.24.2 Field Documentation

6.24.2.1 sblock_origin

```
uint8_t* tx_all_settings_args::sblock_origin
```

Definition at line 36 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

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

- Core/Src/uvfr_settings.c

6.25 tx_journal_args Struct Reference

Data Fields

- uint32_t start_time
- uint32_t end_time

6.25.1 Detailed Description

Definition at line 43 of file uvfr_settings.c.

6.25.2 Field Documentation

6.25.2.1 end_time

```
uint32_t tx_journal_args::end_time
```

Definition at line 45 of file uvfr_settings.c.

6.25.2.2 start_time

```
uint32_t tx_journal_args::start_time
```

Definition at line 44 of file uvfr_settings.c.

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

- Core/Src/uvfr_settings.c

6.26 uv19_pdu_settings Struct Reference

```
#include <pdu.h>
```

Data Fields

- uint32_t PDU_rx_addr
- uint32_t PDU_tx_addr
- uint8_t sdc_channel
- uint8_t brake_light_channel
- uint8_t buzzer_channel
- uint8_t coolant_pump_channel
- uint8_t fan1_channel
- uint8_t fan2_channel
- uint8_t mc_rfe_channel

6.26.1 Detailed Description

Definition at line 14 of file pdu.h.

6.26.2 Field Documentation

6.26.2.1 brake_light_channel

```
uint8_t uv19_pdu_settings::brake_light_channel
```

Definition at line 18 of file pdu.h.

6.26.2.2 buzzer_channel

```
uint8_t uv19_pdu_settings::buzzer_channel
```

Definition at line 19 of file pdu.h.

6.26.2.3 coolant_pump_channel

```
uint8_t uv19_pdu_settings::coolant_pump_channel
```

Definition at line 20 of file pdu.h.

6.26.2.4 fan1_channel

```
uint8_t uv19_pdu_settings::fan1_channel
```

Definition at line 21 of file pdu.h.

6.26.2.5 fan2_channel

```
uint8_t uv19_pdu_settings::fan2_channel
```

Definition at line 22 of file pdu.h.

6.26.2.6 mc_rfe_channel

```
uint8_t uv19_pdu_settings::mc_rfe_channel
```

Definition at line 23 of file pdu.h.

6.26.2.7 PDU_rx_addr

```
uint32_t uv19_pdu_settings::PDU_rx_addr
```

Definition at line 15 of file pdu.h.

6.26.2.8 PDU_tx_addr

```
uint32_t uv19_pdu_settings::PDU_tx_addr
```

Definition at line 16 of file pdu.h.

6.26.2.9 sdc_channel

```
uint8_t uv19_pdu_settings::sdc_channel
```

Definition at line 17 of file pdu.h.

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

- Core/Inc/pdu.h

6.27 uv_binary_semaphore_info Struct Reference

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

Data Fields

- SemaphoreHandle_t handle

6.27.1 Detailed Description

Definition at line 244 of file uvfr_utils.h.

6.27.2 Field Documentation

6.27.2.1 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

6.28 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]

6.28.1 Detailed Description

Representative of a CAN message.

Definition at line 270 of file uvfr_utils.h.

6.28.2 Field Documentation

6.28.2.1 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(), handleIncomingLaptopMsg(), MC_Request_Data(), MC_Set_Param(), MotorControllerSpinTest(), Parse_↩ Bamocar_Response(), ProcessMotorControllerResponse(), sendDaqMsg(), tempMonitorTask(), testfunc2(), uv↩ SendSpecificParam(), and uvSettingsProgrammerTask().

6.28.2.2 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(), handleIncomingLaptopMsg(), MC_Request_Data(), MC_Set_Param(), MotorControllerSpinTest(), Parse_↩Bamocar_Response(), ProcessMotorControllerResponse(), sendDaqMsg(), tempMonitorTask(), and uvSend↩SpecificParam().

6.28.2.3 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 CAN1 if all zeros

Definition at line 271 of file uvfr_utils.h.

Referenced by __uvCANtxCritSection(), CANbusTxSvcDaemon(), MC_Request_Data(), MotorControllerSpinTest(), tempMonitorTask(), testfunc2(), and uvSendSpecificParam().

6.28.2.4 msg_id

```
uint32_t uv_CAN_msg::msg_id
```

The ID of a message

Definition at line 276 of file uvfr_utils.h.

Referenced by __uvCANtxCritSection(), callFunctionFromCANid(), CANbusTxSvcDaemon(), HAL_CAN_RxFifo0↩MsgPendingCallback(), handleIncomingLaptopMsg(), MC_Request_Data(), MC_Set_Param(), MotorController↩SpinTest(), sendDaqMsg(), tempMonitorTask(), testfunc2(), and uvSendSpecificParam().

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

- Core/Inc/uvfr_utils.h

6.29 uv_imd_settings Struct Reference

```
#include <imd.h>
```

Data Fields

- uint16_t min_isolation_resistances
- uint16_t expected_isolation_capacitances
- uint16_t max_imd_temperature

6.29.1 Detailed Description

Definition at line 29 of file imd.h.

6.29.2 Field Documentation

6.29.2.1 expected_isolation_capacitances

```
uint16_t uv_imd_settings::expected_isolation_capacitances
```

Definition at line 31 of file imd.h.

6.29.2.2 max_imd_temperature

```
uint16_t uv_imd_settings::max_imd_temperature
```

Definition at line 32 of file imd.h.

6.29.2.3 min_isolation_resistances

```
uint16_t uv_imd_settings::min_isolation_resistances
```

Definition at line 30 of file imd.h.

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

- Core/Inc/imd.h

6.30 uv_init_struct Struct Reference

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

Data Fields

- bool use_default_settings

6.30.1 Detailed Description

contains info relevant to initializing the vehicle

Definition at line 284 of file uvfr_utils.h.

6.30.2 Field Documentation

6.30.2.1 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

6.31 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

6.31.1 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.

6.31.2 Field Documentation

6.31.2.1 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().

6.31.2.2 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(), and uvInit().

6.31.2.3 specific_args

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

Definition at line 330 of file uvfr_utils.h.

Referenced by uvInit().

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

- Core/Inc/uvfr_utils.h

6.32 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

6.32.1 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.

6.32.2 Field Documentation

6.32.2.1 device

`uv_ext_device_id uv_init_task_response::device`

Definition at line 357 of file uvfr_utils.h.

Referenced by MC_Startup(), and uvInit().

6.32.2.2 errmsg

`char* uv_init_task_response::errmsg`

Definition at line 359 of file uvfr_utils.h.

Referenced by uvInit().

6.32.2.3 nchar

`uint8_t uv_init_task_response::nchar`

Definition at line 358 of file uvfr_utils.h.

Referenced by uvInit().

6.32.2.4 status

`uv_status uv_init_task_response::status`

Definition at line 356 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

6.33 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]

6.33.1 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.

6.33.2 Field Documentation

6.33.2.1 e_code

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

Definition at line 345 of file uvfr_utils.h.

6.33.2.2 init_params

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

Definition at line 342 of file uvfr_utils.h.

6.33.2.3 peripheral_status

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

Definition at line 344 of file uvfr_utils.h.

6.33.2.4 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

6.34 uv_mutex_info Struct Reference

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

Data Fields

- SemaphoreHandle_t handle

6.34.1 Detailed Description

Definition at line 239 of file uvfr_utils.h.

6.34.2 Field Documentation

6.34.2.1 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

6.35 uv_os_settings Struct Reference

Settings that dictate state engine behavior.

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

Data Fields

- TickType_t svc_task_manager_period
- TickType_t task_manager_period
- TickType_t max_svc_task_period
- TickType_t max_task_period
- TickType_t min_task_period
- float task_overshoot_margin_noncrit
- float task_overshoot_margin_crit
- float task_throttle_increment
- uint16_t os_flags

6.35.1 Detailed Description

Settings that dictate state engine behavior.

Definition at line 177 of file uvfr_state_engine.h.

6.35.2 Field Documentation

6.35.2.1 max_svc_task_period

```
TickType_t uv_os_settings::max_svc_task_period
```

Definition at line 180 of file uvfr_state_engine.h.

6.35.2.2 max_task_period

```
TickType_t uv_os_settings::max_task_period
```

Definition at line 181 of file uvfr_state_engine.h.

6.35.2.3 min_task_period

```
TickType_t uv_os_settings::min_task_period
```

Definition at line 182 of file uvfr_state_engine.h.

6.35.2.4 os_flags

```
uint16_t uv_os_settings::os_flags
```

Definition at line 187 of file uvfr_state_engine.h.

6.35.2.5 svc_task_manager_period

```
TickType_t uv_os_settings::svc_task_manager_period
```

Definition at line 178 of file uvfr_state_engine.h.

6.35.2.6 task_manager_period

```
TickType_t uv_os_settings::task_manager_period
```

Definition at line 179 of file uvfr_state_engine.h.

6.35.2.7 task_overshoot_margin_crit

```
float uv_os_settings::task_overshoot_margin_crit
```

Definition at line 184 of file uvfr_state_engine.h.

Referenced by uvTaskManager().

6.35.2.8 task_overshoot_margin_noncrit

```
float uv_os_settings::task_overshoot_margin_noncrit
```

Definition at line 183 of file uvfr_state_engine.h.

Referenced by uvTaskManager().

6.35.2.9 task_throttle_increment

```
float uv_os_settings::task_throttle_increment
```

Definition at line 185 of file uvfr_state_engine.h.

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

- Core/Inc/uvfr_state_engine.h

6.36 uv_persistant_data_frame Struct Reference

`#include <odometer.h>`

Data Fields

- uint64_t total_vehicle_uptime
- uint64_t total_time_driving
- uint64_t total_distance_cm

6.36.1 Detailed Description

Definition at line 13 of file odometer.h.

6.36.2 Field Documentation

6.36.2.1 total_distance_cm

`uint64_t uv_persistant_data_frame::total_distance_cm`

Definition at line 16 of file odometer.h.

6.36.2.2 total_time_driving

`uint64_t uv_persistant_data_frame::total_time_driving`

Definition at line 15 of file odometer.h.

6.36.2.3 total_vehicle_uptime

`uint64_t uv_persistant_data_frame::total_vehicle_uptime`

Definition at line 14 of file odometer.h.

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

- Core/Inc/odometer.h

6.37 uv_scd_response Struct Reference

`#include <uvfr_state_engine.h>`

Data Fields

- enum uv_scd_response_e response_val
- uv_task_id meta_id

6.37.1 Detailed Description

Definition at line 114 of file uvfr_state_engine.h.

6.37.2 Field Documentation

6.37.2.1 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().

6.37.2.2 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

6.38 uv_semaphore_info Struct Reference

`#include <uvfr_utils.h>`

Data Fields

- SemaphoreHandle_t handle

6.38.1 Detailed Description

Definition at line 249 of file uvfr_utils.h.

6.38.2 Field Documentation

6.38.2.1 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

6.39 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
- TickType_t last_execution_time
- uint16_t active_states
- uint16_t deletion_states
- uint16_t suspension_states
- uint16_t task_flags
- uint8_t throttle_factor

6.39.1 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 220 of file uvfr_state_engine.h.

6.39.2 Field Documentation

6.39.2.1 active_states

```
uint16_t uv_task_info::active_states
```

Definition at line 251 of file uvfr_state_engine.h.

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

6.39.2.2 cmd_data

```
uv_task_cmd uv_task_info::cmd_data
```

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

Definition at line 241 of file uvfr_state_engine.h.

Referenced by CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), daqMasterTask(), killSelf(), odometer↩ Task(), StartDrivingLoop(), suspendSelf(), tempMonitorTask(), uvDeleteSVCTask(), uvDeleteTask(), uvSettings↩ ProgrammerTask(), uvSuspendTask(), and uvTaskManager().

6.39.2.3 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 225 of file uvfr_state_engine.h.

6.39.2.4 deletion_states

```
uint16_t uv_task_info::deletion_states
```

Definition at line 252 of file uvfr_state_engine.h.

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

6.39.2.5 last_execution_time

```
TickType_t uv_task_info::last_execution_time
```

Definition at line 249 of file uvfr_state_engine.h.

Referenced by uvStartTask(), uvTaskManager(), and uvTaskPeriodEnd().

6.39.2.6 parent

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

info about the parent of the task

Definition at line 245 of file uvfr_state_engine.h.

Referenced by uvCreateServiceTask(), and uvCreateTask().

6.39.2.7 stack_size

```
uint32_t uv_task_info::stack_size
```

Number of words allocated to the stack of the task

Definition at line 231 of file uvfr_state_engine.h.

Referenced by initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), uvConfigSettingTask(), uv↩CreateServiceTask(), uvCreateTask(), uvStartStateMachine(), uvStartSVCTask(), and uvStartTask().

6.39.2.8 suspension_states

```
uint16_t uv_task_info::suspension_states
```

Definition at line 253 of file uvfr_state_engine.h.

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

6.39.2.9 task_args

```
void* uv_task_info::task_args
```

arguments for the specific task, this is where we will likely pass in task settings

Definition at line 243 of file uvfr_state_engine.h.

Referenced by initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), StartDrivingLoop(), uvConfig↩SettingTask(), and uvStartSVCTask().

6.39.2.10 task_flags

```
uint16_t uv_task_info::task_flags
```

- Bits 0:1 - | Task MGMT | Vehicle Application task - 01 | Periodic SVC Task - 10 | Dormant SVC Task - 11
- Bit 2 - Log task start + stop time
- Bit 3 - Log mem usage
- Bit 4 - SCD ignore flag (only use if task is application layer
- Bit 5 - is parent
- Bit 6 - is child
- Bit 7 - is orphaned
- Bit 8 - error in child task
- Bit 9 - awaiting deferred deletion
- Bit 10 - deferred deletion enabled
- Bits 11:12 - Deadline firmness | No enforcement - 00 | Gradual Priority 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 255 of file uvfr_state_engine.h.

Referenced by _stateChangeDaemon(), uvCreateServiceTask(), uvCreateTask(), uvLateTaskHandler(), uv↩ScheduleTaskDeletion(), uvStartStateMachine(), uvStartSVCTask(), uvTaskCrashHandler(), and uvTaskManager().

6.39.2.11 task_function

```
TaskFunction_t uv_task_info::task_function
```

Pointer to function that implements the task

Definition at line 227 of file uvfr_state_engine.h.

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

6.39.2.12 task_handle

```
TaskHandle_t uv_task_info::task_handle
```

Handle of freeRTOS task control block

Definition at line 239 of file uvfr_state_engine.h.

Referenced by _stateChangeDaemon(), CANbusRxSvcDaemon(), killSelf(), proccessSCDMsg(), suspendSelf(), uvCreateServiceTask(), uvCreateTask(), uvDeleteSVCTask(), uvDeleteTask(), uvSettingsProgrammerTask(), uvStartSVCTask(), uvStartTask(), uvSuspendTask(), and uvTaskManager().

6.39.2.13 task_id

```
uv_task_id uv_task_info::task_id
```

Detailed description after the member

Definition at line 221 of file uvfr_state_engine.h.

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

6.39.2.14 task_name

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

Detailed description after the member

Definition at line 222 of file uvfr_state_engine.h.

Referenced by _uvValidateSpecificTask(), compareTaskByName(), initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), uvConfigSettingTask(), uvCreateServiceTask(), uvCreateTask(), uvInit(), uvStartStateMachine(), uvStartSVCTask(), and uvStartTask().

6.39.2.15 task_period

`uv_timespan_ms uv_task_info::task_period`

Maximum period between task execution

Definition at line 224 of file uvfr_state_engine.h.

Referenced by daqMasterTask(), initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), odometer↩Task(), StartDrivingLoop(), tempMonitorTask(), uvConfigSettingTask(), uvLateTaskHandler(), uvStartState↩Machine(), uvTaskManager(), and uvTaskPeriodEnd().

6.39.2.16 task_priority

`osPriority uv_task_info::task_priority`

Priority of the task. Int between 0 and 7

Definition at line 228 of file uvfr_state_engine.h.

Referenced by initDaqTask(), initDrivingLoop(), initOdometer(), initTempMonitor(), uvConfigSettingTask(), uv↩CreateServiceTask(), uvCreateTask(), uvStartSVCTask(), and uvStartTask().

6.39.2.17 task_rx_mailbox

`MessageBufferHandle_t uv_task_info::task_rx_mailbox`

Incoming messages for this task

Definition at line 248 of file uvfr_state_engine.h.

6.39.2.18 task_state

`uv_task_status uv_task_info::task_state`

Definition at line 236 of file uvfr_state_engine.h.

Referenced by _stateChangeDaemon(), killSelf(), proccessSCDMsg(), suspendSelf(), uvCreateServiceTask(), uv↩CreateTask(), uvDeleteSVCTask(), uvDeleteTask(), uvScheduleTaskDeletion(), uvStartSVCTask(), uvStartTask(), uvSuspendSVCTask(), uvSuspendTask(), and uvTaskManager().

6.39.2.19 throttle_factor

```
uint8_t uv_task_info::throttle_factor
```

How much to throttle the task

Definition at line 271 of file uvfr_state_engine.h.

6.39.2.20 tmi

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

how we will be communicating in the future

Definition at line 247 of file uvfr_state_engine.h.

Referenced by uvTaskManager().

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

- Core/Inc/uvfr_state_engine.h

6.40 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

6.40.1 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.

6.40.2 Field Documentation

6.40.2.1 intended_recipient

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

Definition at line 304 of file uvfr_utils.h.

6.40.2.2 message_size

```
size_t uv_task_msg_t::message_size
```

Definition at line 306 of file uvfr_utils.h.

6.40.2.3 message_type

```
uint32_t uv_task_msg_t::message_type
```

Definition at line 302 of file uvfr_utils.h.

6.40.2.4 msg_contents

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

Definition at line 307 of file uvfr_utils.h.

6.40.2.5 sender

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

Definition at line 303 of file uvfr_utils.h.

6.40.2.6 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

6.41 uv_vehicle_settings Struct Reference

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

Data Fields

- struct veh_gen_info * veh_info
- struct uv_os_settings * os_settings
- struct motor_controller_settings * mc_settings
- driving_loop_args * driving_loop_settings
- struct uv_imd_settings * imd_settings
- bms_settings_t * bms_settings
- daq_loop_args * daq_settings
- daq_datapoint * daq_param_list
- struct uv19_pdu_settings * pdu_settings
- uint16_t flags

6.41.1 Detailed Description

Definition at line 135 of file uvfr_settings.h.

6.41.2 Field Documentation

6.41.2.1 bms_settings

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

Definition at line 144 of file uvfr_settings.h.

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

6.41.2.2 daq_param_list

```
daq_datapoint* uv_vehicle_settings::daq_param_list
```

Definition at line 147 of file uvfr_settings.h.

Referenced by configureDaqSubTasks(), initDaqTask(), setupDefaultSettings(), and uvLoadSettingsFromFlash().

6.41.2.3 daq_settings

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

Definition at line 146 of file uvfr_settings.h.

Referenced by initDaqTask(), setupDefaultSettings(), and uvLoadSettingsFromFlash().

6.41.2.4 driving_loop_settings

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

Definition at line 141 of file uvfr_settings.h.

Referenced by initDrivingLoop(), setupDefaultSettings(), and uvLoadSettingsFromFlash().

6.41.2.5 flags

```
uint16_t uv_vehicle_settings::flags
```

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

Definition at line 152 of file uvfr_settings.h.

Referenced by setupDefaultSettings().

6.41.2.6 imd_settings

```
struct uv_imd_settings* uv_vehicle_settings::imd_settings
```

Definition at line 143 of file uvfr_settings.h.

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

6.41.2.7 mc_settings

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

Definition at line 139 of file uvfr_settings.h.

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

6.41.2.8 os_settings

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

Definition at line 138 of file uvfr_settings.h.

Referenced by setupDefaultSettings(), uvLoadSettingsFromFlash(), and uvStartStateMachine().

6.41.2.9 pdu_settings

```
struct uv19_pdu_settings* uv_vehicle_settings::pdu_settings
```

Definition at line 149 of file uvfr_settings.h.

Referenced by setupDefaultSettings(), and uvLoadSettingsFromFlash().

6.41.2.10 veh_info

```
struct veh_gen_info* uv_vehicle_settings::veh_info
```

Definition at line 136 of file uvfr_settings.h.

Referenced by setupDefaultSettings(), and uvLoadSettingsFromFlash().

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

- Core/Inc/uvfr_settings.h

6.42 veh_gen_info Struct Reference

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

Data Fields

- uint32_t wheel_size
- uint32_t drive_ratio
- uint16_t test1
- uint16_t test2
- uint16_t test3
- uint8_t test4
- uint8_t test5
- uint32_t test6

6.42.1 Detailed Description

Definition at line 99 of file uvfr_settings.h.

6.42.2 Field Documentation

6.42.2.1 drive_ratio

```
uint32_t veh_gen_info::drive_ratio
```

Definition at line 101 of file uvfr_settings.h.

6.42.2.2 test1

```
uint16_t veh_gen_info::test1
```

Definition at line 102 of file uvfr_settings.h.

6.42.2.3 test2

```
uint16_t veh_gen_info::test2
```

Definition at line 103 of file uvfr_settings.h.

6.42.2.4 test3

```
uint16_t veh_gen_info::test3
```

Definition at line 104 of file uvfr_settings.h.

6.42.2.5 test4

```
uint8_t veh_gen_info::test4
```

Definition at line 105 of file uvfr_settings.h.

6.42.2.6 test5

```
uint8_t veh_gen_info::test5
```

Definition at line 106 of file uvfr_settings.h.

6.42.2.7 test6

```
uint32_t veh_gen_info::test6
```

Definition at line 108 of file uvfr_settings.h.

6.42.2.8 wheel_size

```
uint32_t veh_gen_info::wheel_size
```

Definition at line 100 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)
- void initADCTask (void)
- void processADCBuffer ()

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 initADCTask()

```
void initADCTask (
             void  )
```

Definition at line 358 of file adc.c.

References Error_Handler(), and StartADCTask().

Referenced by uvInit().

7.1.3.2 MX_ADC1_Init()

```
void MX_ADC1_Init (
             void  )
```

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

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Definition at line 40 of file adc.c.

References Error_Handler(), and hadc1.

Referenced by main().

7.1.3.3 MX_ADC2_Init()

```
void MX_ADC2_Init (
             void  )
```

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

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Definition at line 114 of file adc.c.

References Error_Handler(), and hadc2.

Referenced by main().

7.1.3.4 processADCBuffer()

```
void processADCBuffer ( )
```

Definition at line 320 of file adc.c.

References adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, and adc_buf1.

Referenced by HAL_ADC_ConvCpltCallback().

7.1.4 Variable Documentation

7.1.4.1 hadc1

```
ADC_HandleTypeDef hadc1
```

Definition at line 35 of file adc.c.

Referenced by HAL_ADC_LevelOutOfWindowCallback().

7.1.4.2 hadc2

```
ADC_HandleTypeDef hadc2
```

Definition at line 36 of file adc.c.

Referenced by HAL_TIM_PeriodElapsedCallback().

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 22 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 632 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 uvInit().

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 563 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 uvInit().

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(), 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().

7.4 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.

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(), 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_datapoint
- struct daq_loop_args

  This struct holds info of what needs to be logged.

Macros

- #define _NUM_LOGGABLE_PARAMS

Typedefs

- typedef struct daq_datapoint daq_datapoint
- typedef struct daq_loop_args daq_loop_args

  This struct holds info of what needs to be logged.
- typedef enum uv_status_t uv_status

Enumerations

- enum loggable_params {
  VCU_VEHICLE_STATE, VCU_ERROR_BITFIELD1, VCU_ERROR_BITFIELD2, VCU_ERROR_BITFIELD3,
  VCU_ERROR_BITFIELD4, VCU_CURRENT_UPTIME, VCU_TOTAL_UPTIME, VEH_DISTANCE_RUN,
  VEH_DISTANCE_TOTAL, VEH_LAPNUM, VEH_SPEED, MOTOR_RPM,
  MOTOR_TEMP, MOTOR_CURRENT, MC_VOLTAGE, MC_CURRENT,
  MC_TEMP, MC_ERRORS, BMS_CURRENT, BMS_VOLTAGE,
  BMS_ERRORS, MAX_CELL_TEMP, MIN_CELL_TEMP, AVG_CELL_TEMP,
  ACCUM_SOC, ACCUM_SOH, ACCUM_POWER, ACCUM_POWER_LIMIT,
  APPS1_ADC_VAL, APPS2_ADC_VAL, BPS1_ADC_VAL, BPS2_ADC_VAL,
  ACCELERATOR_PEDAL_RATIO, BRAKE_PRESSURE_PA, POWER_DERATE_FACTOR, CURRENT_DRIVING_MODE,
  IMD_VOLTAGE, IMD_STATUS, IMD_ERRORS, SUS_DAMPER_FL,
  SUS_DUMPER_FR, SUS_DAMPER_RL, SUS_DAMPER_RR, WSS_FR,
  WSS_FL, WSS_RL, WSS_RR, WSS_F_AVG,
  WSS_R_AVG, WSS_SLIP, MAX_LOGGABLE_PARAMS }

Functions

- uv_status associateDaqParamWithVar (uint16_t paramID, void *var)
- uv_status initDaqTask (void *args)

  initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks
- void daqMasterTask (void *args)

  Controls the Daq.

7.5.1 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_datapoint

```
typedef struct daq_datapoint daq_datapoint
```

7.5.2.2 daq_loop_args

```
typedef struct daq_loop_args daq_loop_args
```

This struct holds info of what needs to be logged.

7.5.2.3 uv_status

```
typedef enum uv_status_t uv_status
```

Definition at line 94 of file daq.h.

7.5.3 Enumeration Type Documentation

7.5.3.1 loggable_params

```
enum loggable_params
```

Enumerator

VCU_VEHICLE_STATE	VCU Current Vehicle State
VCU_ERROR_BITFIELD1	
VCU_ERROR_BITFIELD2	
VCU_ERROR_BITFIELD3	
VCU_ERROR_BITFIELD4	
VCU_CURRENT_UPTIME	
VCU_TOTAL_UPTIME	
VEH_DISTANCE_RUN	
VEH_DISTANCE_TOTAL	
VEH_LAPNUM	
VEH_SPEED	
MOTOR_RPM	RPM as reported by motor controller
MOTOR_TEMP	Motor Temp as reported by motor controller
MOTOR_CURRENT	Motor Phase currents as reported by motor controller
MC_VOLTAGE	Pack voltage as measured by motor_controller
MC_CURRENT	Pack current as measured by motor_controller
MC_TEMP	Motor controller temperature
MC_ERRORS	Motor controller errors bitfield
BMS_CURRENT	Pack current measured by BMS
BMS_VOLTAGE	Pack voltage as measured by BMS
BMS_ERRORS	Error codes in BMS
MAX_CELL_TEMP	Max Temperature of a cell from BMS
MIN_CELL_TEMP	Min Temperature of a cell
AVG_CELL_TEMP	Average Cell Temp
ACCUM_SOC	
ACCUM_SOH	
ACCUM_POWER	
ACCUM_POWER_LIMIT	
APPS1_ADC_VAL	
APPS2_ADC_VAL	
BPS1_ADC_VAL	
BPS2_ADC_VAL	
ACCELERATOR_PEDAL_RATIO	
BRAKE_PRESSURE_PA	
POWER_DERATE_FACTOR	
CURRENT_DRIVING_MODE	
IMD_VOLTAGE	Accumulator voltage as measured by IMD
IMD_STATUS	
IMD_ERRORS	
SUS_DAMPER_FL	
SUS_DUMPER_FR	
SUS_DAMPER_RL	
SUS_DAMPER_RR	
WSS_FR	
WSS_FL	
WSS_RL	
WSS_RR	
WSS_F_AVG	

Enumerator

WSS_R_AVG	
WSS_SLIP	
MAX_LOGGABLE_PARAMS	THIS MUST BE THE FINAL PARAM

Definition at line 18 of file daq.h.

7.5.4 Function Documentation

7.5.4.1 associateDaqParamWithVar()

```
uv_status associateDaqParamWithVar (
            uint16_t paramID,
            void * var )
```

Definition at line 99 of file daq.c.

References MAX_LOGGABLE_PARAMS, param_ptrs, UV_ERROR, UV_OK, and uvIsPTRValid().

Referenced by initDrivingLoop().

7.5.4.2 daqMasterTask()

```
void daqMasterTask (
            void * args )
```

Controls the Daq.

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
 */
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
//TickType_t last_time = xTaskGetTickCount();       /**
```

Definition at line 291 of file daq.c.

References uv_task_info::cmd_data, killSelf(), startDaqSubTasks(), stopDaqSubTasks(), suspendSelf(), uv_task↩_info::task_period, UV_KILL_CMD, UV_OK, UV_SUSPEND_CMD, and uvTaskDelay.

Referenced by initDaqTask().

7.5.4.3 initDaqTask()

```
uv_status initDaqTask (
            void * args )
```

initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks

This is a fairly standard function. Here are the things that it does in order:

Step 1: Get Daq settings.

Step 2: Create and configure DAQ task.

Step 3: Read which parameters we want to read.

Step 4: Generate Subtask Metadata

Step 5: Assign params to subtasks

Definition at line 252 of file daq.c.

References uv_task_info::active_states, configureDaqSubTasks(), curr_daq_settings, current_vehicle_settings, daq_loop_args::daq_child_priority, uv_vehicle_settings::daq_param_list, uv_vehicle_settings::daq_settings, daqMasterTask(), datapoints, 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().

7.6 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 42 of file driving_loop.c.

References uv_task_info::active_states, adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, APPS1_ADC_↩ VAL, APPS2_ADC_VAL, associateDaqParamWithVar(), BPS1_ADC_VAL, BPS2_ADC_VAL, current_vehicle_↩ settings, uv_task_info::deletion_states, driving_args, uv_vehicle_settings::driving_loop_settings, PROGRAMMING, uv_task_info::stack_size, StartDrivingLoop(), uv_task_info::suspension_states, uv_task_info::task_args, uv_task↩ _info::task_function, uv_task_info::task_name, uv_task_info::task_period, uv_task_info::task_priority, UV_DRIVI↩ NG, UV_ERROR, UV_ERROR_STATE, UV_INIT, UV_LAUNCH_CONTROL, UV_OK, UV_READY, UV_SUSPE↩ NDED, 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 rachan: ensures the function runs exactly 100ms, regardless of execution time.

```
 */
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = xTaskGetTickCount();
/**
```

Performs safety checks on APPS (Throttle) and BPS (Brake) sensors.

This function ensures that:

- Throttle position sensors (APPS1 & APPS2) are within 10% of each other.
- Brake pressure sensors (BPS1 & BPS2) are within 5% of each other.
- Sensors are within their expected min/max ranges.
- Brake and throttle are not pressed at the same time.

If a **fatal error** is detected (e.g., sensor out of range), the function:

- **Stops the motor**.
- **Kills the task execution** (`killSelf()`).

If a **non-fatal error** occurs (e.g., sensor mismatch exceeding the limit):

- **Stops the motor**.
- **Suspends the task temporarily** (`suspendSelf()`).

If safety conditions return to normal, the function:

- **Restarts the motor**.

Parameters

dl_params	Pointer to the driving loop parameters.
apps1_value	Raw sensor reading from APPS1.
apps2_value	Raw sensor reading from APPS2.
bps1_value	Raw sensor reading from BPS1.
bps2_value	Raw sensor reading from BPS2.
params	Pointer to the current task information.

Return values

true	All safety checks passed.
false	One or more safety checks failed.

Definition at line 194 of file driving_loop.c.

References adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, uv_task_info::cmd_data, is_accelerating, killSelf(), performSafetyChecks(), Plausible, suspendSelf(), T_PREV, T_REQ, 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)32768)
- #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 128
- #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 128
```

Definition at line 97 of file FreeRTOSConfig.h.

7.10.1.26 configTOTAL_HEAP_SIZE

```
#define configTOTAL_HEAP_SIZE ((size_t)32768)
```

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"
```

Data Structures

- struct uv_imd_settings

Typedefs

- typedef struct uv_imd_settings uv_imd_settings

Enumerations

- enum imd_status_bits {
  Isolation_status_bit0 = 0b00000001, Isolation_status_bit1 = 0b00000010, Low_Battery_Voltage = 0b00000100, High_Battery_Voltage = 0b00001000,
  Exc_off = 0b00010000, High_Uncertainty = 0b00100000, Touch_energy_fault = 0b01000000, Hardware_Error = 0b10000000 }
- enum imd_status_requests {
  isolation_state = 0xE0, isolation_resistances = 0xE1, isolation_capacitances = 0xE2, voltages_Vp_and_Vn = 0xE3,
  battery_voltage = 0xE4, Error_flags = 0xE5, safety_touch_energy = 0xE6, safety_touch_current = 0xE7,
  Max_battery_working_voltage = 0xF0, Temperature = 0x80 }
- enum imd_error_flags {
  Err_temp = 0x0080, Err_clock = 0x0100, Err_Watchdog = 0x0200, Err_Vpwr = 0x0400,
  Err_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 Typedef Documentation

7.12.1.1 uv_imd_settings

```
typedef struct uv_imd_settings uv_imd_settings
```

7.12.2 Enumeration Type Documentation

7.12.2.1 imd_error_flags

```
enum imd_error_flags
```

Enumerator

Err_temp	
Err_clock	
Err_Watchdog	
Err_Vpwr	
Err_Vexi	
Err_VxR	
Err_CH	
Err_Vx1	
Err_Vx2	

Definition at line 75 of file imd.h.

7.12.2.2 imd_high_resolution_measurements

```
enum imd_high_resolution_measurements
```

Enumerator

Vn_hi_res	
Vp_hi_res	
Vexc_hi_res	
Vb_hi_res	
Vpwr_hi_res	

Definition at line 105 of file imd.h.

7.12.2.3 imd_manufacturer_requests

```
enum imd_manufacturer_requests
```

Enumerator

Part_name_0	
Part_name_1	
Part_name_2	
Part_name_3	
Version_0	
Version_1	
Version_2	
Serial_number↩ _0	
Serial_number↩ _1	
Serial_number↩ _2	
Serial_number↩ _3	
Uptime_counter	

Definition at line 89 of file imd.h.

7.12.2.4 imd_status_bits

```
enum imd_status_bits
```

Enumerator

Isolation_status_bit0	
Isolation_status_bit1	
Low_Battery_Voltage	
High_Battery_Voltage	
Exc_off	
High_Uncertainty	
Touch_energy_fault	
Hardware_Error	

Definition at line 16 of file imd.h.

7.12.2.5 imd_status_requests

```
enum imd_status_requests
```

Enumerator

isolation_state	
isolation_resistances	
isolation_capacitances	
voltages_Vp_and_Vn	
battery_voltage	
Error_flags	
safety_touch_energy	
safety_touch_current	
Max_battery_working_voltage	
Temperature	

Definition at line 39 of file imd.h.

7.12.3 Function Documentation

7.12.3.1 IMD_Check_Battery_Voltage()

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

Definition at line 356 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.3.2 IMD_Check_Error_Flags()

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

Definition at line 262 of file imd.c.

References Err_CH, Err_clock, Err_temp, Err_Vexi, Err_Vpwr, Err_Vx1, Err_Vx2, Err_VxR, and Err_Watchdog.

Referenced by IMD_Parse_Message().

7.12.3.3 IMD_Check_Isolation_Capacitances()

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

Definition at line 342 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.3.4 IMD_Check_Isolation_Resistances()

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

Definition at line 317 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.12.3.5 IMD_Check_Isolation_State()

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

Definition at line 301 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.12.3.6 IMD_Check_Max_Battery_Working_Voltage()

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

Definition at line 393 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.3.7 IMD_Check_Part_Name()

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

Definition at line 406 of file imd.c.

References IMD_Expected_Part_Name, IMD_Part_Name_0_Set, IMD_Part_Name_1_Set, IMD_Part_Name_2_↩ Set, IMD_Part_Name_3_Set, IMD_Part_Name_Set, IMD_Read_Part_Name, Part_name_0, Part_name_1, Part↩ _name_2, and Part_name_3.

Referenced by IMD_Parse_Message().

7.12.3.8 IMD_Check_Safety_Touch_Current()

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

Definition at line 381 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.3.9 IMD_Check_Safety_Touch_Energy()

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

Definition at line 374 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.3.10 IMD_Check_Serial_Number()

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

Definition at line 488 of file imd.c.

References IMD_Expected_Serial_Number, IMD_Read_Serial_Number, IMD_Serial_Number_0_Set, IMD↩ _Serial_Number_1_Set, IMD_Serial_Number_2_Set, IMD_Serial_Number_3_Set, IMD_Serial_Number_Set, Serial_number_0, Serial_number_1, Serial_number_2, and Serial_number_3.

Referenced by IMD_Parse_Message().

7.12.3.11 IMD_Check_Status_Bits()

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

Definition at line 218 of file imd.c.

References Error_flags, Hardware_Error, High_Battery_Voltage, High_Uncertainty, IMD_error_flags_requested, IMD_High_Uncertainty, IMD_Request_Status(), Isolation_status_bit0, Isolation_status_bit1, and Low_Battery_Voltage.

Referenced by IMD_Parse_Message().

7.12.3.12 IMD_Check_Temperature()

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

Definition at line 363 of file imd.c.

References IMD_Temperature.

Referenced by IMD_Parse_Message().

7.12.3.13 IMD_Check_Uptime()

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

Definition at line 529 of file imd.c.

7.12.3.14 IMD_Check_Version()

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

Definition at line 448 of file imd.c.

References IMD_Expected_Version, IMD_Read_Version, IMD_Version_0_Set, IMD_Version_1_Set, IMD_Version_2_Set, IMD_Version_Set, Version_0, Version_1, and Version_2.

Referenced by IMD_Parse_Message().

7.12.3.15 IMD_Check_Voltages_Vp_and_Vn()

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

Definition at line 349 of file imd.c.

Referenced by IMD_Parse_Message().

7.12.3.16 IMD_Parse_Message()

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

Definition at line 73 of file imd.c.

References battery_voltage, Error_flags, Error_Handler(), IMD_Check_Battery_Voltage(), IMD_Check_Error_Flags(), IMD_Check_Isolation_Capacitances(), IMD_Check_Isolation_Resistances(), IMD_Check_Isolation_State(), IMD_Check_Max_Battery_Working_Voltage(), IMD_Check_Part_Name(), IMD_Check_Safety_Touch_Current(), IMD_Check_Safety_Touch_Energy(), IMD_Check_Serial_Number(), IMD_Check_Status_Bits(), IMD_Check_Temperature(), IMD_Check_Version(), IMD_Check_Voltages_Vp_and_Vn(), isolation_capacitances, isolation_resistances, isolation_state, Max_battery_working_voltage, Part_name_0, Part_name_1, Part_name_2, Part_name_3, safety_touch_current, safety_touch_energy, Serial_number_0, Serial_number_1, Serial_number_2, Serial_number_3, Temperature, Uptime_counter, Vb_hi_res, Version_0, Version_1, Version_2, Vexc_hi_res, Vn_hi_res, voltages_Vp_and_Vn, Vp_hi_res, and Vpwr_hi_res.

7.12.3.17 IMD_Request_Status()

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

Definition at line 185 of file imd.c.

References Error_Handler(), hcan2, IMD_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

Referenced by IMD_Check_Status_Bits(), and IMD_Startup().

7.12.3.18 IMD_Startup()

```
void IMD_Startup ( )
```

Definition at line 533 of file imd.c.

References IMD_Request_Status(), isolation_state, Max_battery_working_voltage, Part_name_0, Part_name_1, Part_name_2, Part_name_3, Serial_number_0, Serial_number_1, Serial_number_2, Serial_number_3, Version_0, Version_1, and Version_2.

7.12.3.19 initIMD()

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

Definition at line 559 of file imd.c.

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

Referenced by uvInit().

7.13 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 384 of file main.c.

Referenced by HAL_ADC_MspInit(), HAL_CAN_RxFifo0MsgPendingCallback(), IMD_Parse_Message(), IMD_Request_Status(), initADCTask(), 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"
#include "can.h"
```

Macros

- #define SERIAL_NUMBER_REGISTER 0x62
- #define FIRMWARE_VERSION_REGISTER 0x1B
- #define N_set_cmd N_set

Typedefs

- typedef struct motor_controller_settings motor_controller_settings
- typedef struct uv_CAN_msg uv_CAN_msg

Enumerations

- enum motor_controller_speed_parameters { N_actual = 0x30, N_set = 0x31, N_cmd = 0x32, N_error = 0x33 }
- enum motor_controller_status { LOGIMAP_ERRORS = 0x82, LOGIMAP_IO = 0x83, POS_ACTUAL = 0x86 }
- enum motor_controller_current_parameters { CURRENT_ACTUAL = 0x69 }
- enum motor_controller_motor_constants {
  nominal_motor_frequency = 0x05, nominal_motor_voltage = 0x06, power_factor = 0x0E, motor_max_current = 0x4D,
  motor_continuous_current = 0x4E, motor_pole_number = 0x4F, motor_kt_constant = 0x87, motor_ke_constant = 0x87,
  rated_motor_speed = 0x59, motor_temperature_switch_off_point = 0xA3, stator_leakage_inductance = 0xB1, nominal_magnitizing_current = 0xB2,
  motor_magnetising_inductance = 0xB3, rotor_resistance = 0xB4, minimum_magnetising_current = 0xB5, time_constant_rotor = 0xB6,
  leakage_inductance_ph_ph = 0xBB, stator_resistance_ph_ph = 0xBC, time_constant_stator = 0xBD }
- enum motor_controller_temperatures {
  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)

  Initializes the motor controller.
- uint16_t MotorControllerSpinTest (float T_filtered)

  Sends a direct torque command to the motor controller.
- void MC_Request_Data (uint8_t RegID)

  Sends a CAN request to retrieve a specific register from the motor controller.
- void ProcessMotorControllerResponse (uv_CAN_msg *msg)

  Processes a motor controller response received via CAN.
- void Parse_Bamocar_Response (uv_CAN_msg *msg)

  Parses a 32-bit value from a CAN message in little-endian format.

Variables

- motor_controller_settings mc_default_settings

7.14.1 Macro Definition Documentation

7.14.1.1 FIRMWARE_VERSION_REGISTER

```
#define FIRMWARE_VERSION_REGISTER 0x1B
```

Definition at line 165 of file motor_controller.h.

7.14.1.2 N_set_cmd

```
#define N_set_cmd N_set
```

Definition at line 169 of file motor_controller.h.

7.14.1.3 SERIAL_NUMBER_REGISTER

```
#define SERIAL_NUMBER_REGISTER 0x62
```

Definition at line 164 of file motor_controller.h.

7.14.2 Typedef Documentation

7.14.2.1 motor_controller_settings

```
typedef struct motor_controller_settings motor_controller_settings
```

Definition at line 16 of file motor_controller.h.

7.14.2.2 uv_CAN_msg

```
typedef struct uv_CAN_msg uv_CAN_msg
```

Definition at line 17 of file motor_controller.h.

7.14.3 Enumeration Type Documentation

7.14.3.1 motor_controller_current_parameters

```
enum motor_controller_current_parameters
```

Enumerator

CURRENT_ACTUAL	

Definition at line 37 of file motor_controller.h.

7.14.3.2 motor_controller_io

```
enum motor_controller_io
```

Enumerator

todo6969	

Definition at line 124 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 148 of file motor_controller.h.

7.14.3.4 motor_controller_measurements

```
enum motor_controller_measurements
```

Enumerator

DC_bus_voltage	

Definition at line 77 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 42 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 129 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 142 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 22 of file motor_controller.h.

7.14.3.9 motor_controller_startup

```
enum motor_controller_startup
```

Enumerator

clear_errors	
firmware_version	

Definition at line 155 of file motor_controller.h.

7.14.3.10 motor_controller_status

```
enum motor_controller_status
```

Enumerator

LOGIMAP_ERRORS	
LOGIMAP_IO	
POS_ACTUAL	

Definition at line 29 of file motor_controller.h.

7.14.3.11 motor_controller_status_information_errors_warnings

```
enum motor_controller_status_information_errors_warnings
```

Enumerator

motor_controller_errors_warnings	
eprom_read_error	
hardware_fault	
rotate_field_enable_not_present_run	
CAN_timeout_error	
feedback_signal_error	
mains_voltage_min_limit	
motor_temp_max_limit	
IGBT_temp_max_limit	
mains_voltage_max_limit	
critical_AC_current	
race_away_detected	
ecode_timeout_error	
watchdog_reset	
AC_current_offset_fault	
internal_hardware_voltage_problem	
bleed_resistor_overload	
parameter_conflict_detected	
special_CPU_fault	
rotate_field_enable_not_present_norun	
auxiliary_voltage_min_limit	
feedback_signal_problem	
warning_5	
motor_temperature_warning	
IGBT_temperature_warning	
Vout_saturation_max_limit	
warning_9	
speed_actual_resolution_limit	
check_ecode_ID	
tripzone_glitch_detected	
ADC_sequencer_problem	
ADC_measurement_problem	
bleeder_resistor_warning	

Definition at line 82 of file motor_controller.h.

7.14.3.12 motor_controller_temperatures

```
enum motor_controller_temperatures
```

Enumerator

igbt_temperature	
motor_temperature	
air_temperature	
current_derate_temperature	
temp_sensor_pt1	
temp_sensor_pt2	
temp_sensor_pt3	
temp_sensor_pt4	

Definition at line 65 of file motor_controller.h.

7.14.4 Function Documentation

7.14.4.1 MC_Request_Data()

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

Sends a CAN request to retrieve a specific register from the motor controller.

The request message is formatted as: [0x3D, RegID, 0], which should trigger an immediate reply.

Definition at line 63 of file motor_controller.c.

References motor_controller_settings::can_id_tx, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, mc_default_settings, uv_CAN_msg::msg_id, UV_OK, and uvSendCanMSG().

Referenced by MC_Startup().

7.14.4.2 MC_Startup()

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

Initializes the motor controller.

This routine performs the following steps:

1. Requests the serial number and firmware version.

2. Sends a nominal torque command (spin test).
3. Requests error/warning data.
4. Suspends itself after successful initialization.

Definition at line 278 of file motor_controller.c.

References uv_init_task_response::device, FIRMWARE_VERSION_REGISTER, uv_init_task_args::init_info↩_queue, insertCANMessageHandler(), mc_default_settings, MC_Request_Data(), MC_Set_Param(), MOTO↩R_CONTROLLER, motor_controller_errors_warnings, MotorControllerSpinTest(), Parse_Bamocar_Response(), motor_controller_settings::proportional_gain, SERIAL_NUMBER_REGISTER, uv_init_task_response::status, and UV_OK.

Referenced by uvInit().

7.14.4.3 MotorControllerSpinTest()

```
uint16_t MotorControllerSpinTest (
            float T_filtered )
```

Sends a direct torque command to the motor controller.

This function accepts a float for the desired torque (T_filtered), clamps it between 0 and 100, converts it to a 16-bit integer, and then sends it over CAN using uvSendCanMSG.

Definition at line 30 of file motor_controller.c.

References motor_controller_settings::can_id_tx, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, mc_default_settings, uv_CAN_msg::msg_id, N_set, UV_OK, and uvSendCanMSG().

Referenced by MC_Startup(), and sendTorqueToMotorController().

7.14.4.4 Parse_Bamocar_Response()

```
void Parse_Bamocar_Response (
            uv_CAN_msg * msg )
```

Parses a 32-bit value from a CAN message in little-endian format.

This example assumes that the data bytes are stored as: data[0] = LSB, data[3] = MSB.

Definition at line 104 of file motor_controller.c.

References uv_CAN_msg::data, deserializeSmallE16, uv_CAN_msg::dlc, mc_default_settings, and motor_↩controller_settings::proportional_gain.

Referenced by MC_Startup().

7.14.4.5 ProcessMotorControllerResponse()

```
void ProcessMotorControllerResponse (
            uv_CAN_msg * msg )
```

Processes a motor controller response received via CAN.

This function examines the first byte as the register ID and then processes the rest of the message using little-endian parsing. For error/warning responses (for example, when reg_id equals motor_controller_errors_warnings), it calls the error handler.

Definition at line 211 of file motor_controller.c.

References CURRENT_ACTUAL, uv_CAN_msg::data, uv_CAN_msg::dlc, LOGIMAP_ERRORS, LOGIMAP_IO, motor_controller_errors_warnings, MotorControllerErrorHandler_16bitLE(), N_actual, and POS_ACTUAL.

7.14.5 Variable Documentation

7.14.5.1 mc_default_settings

```
motor_controller_settings mc_default_settings
```

Definition at line 14 of file motor_controller.c.

7.15 Core/Inc/odometer.h File Reference

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

Data Structures

- struct uv_persistant_data_frame

Typedefs

- typedef struct uv_persistant_data_frame uv_persistant_data_frame

Functions

- uv_status initOdometer (void *args)
- void odometerTask (void *args)

  , gotta know what the distance travelled is fam

7.15.1 Typedef Documentation

7.15.1.1 uv_persistant_data_frame

```
typedef struct uv_persistant_data_frame uv_persistant_data_frame
```

7.15.2 Function Documentation

7.15.2.1 initOdometer()

```
uv_status initOdometer (
            void * args )
```

Definition at line 11 of file odometer.c.

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

Referenced by uvInitStateEngine().

7.15.2.2 odometerTask()

```
void odometerTask (
            void * args )
```

, gotta know what the distance travelled is fam

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
 */
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = xTaskGetTickCount();
    /**
```

Definition at line 46 of file odometer.c.

References uv_task_info::cmd_data, killSelf(), suspendSelf(), uv_task_info::task_period, UV_KILL_CMD, and UV_SUSPEND_CMD.

Referenced by initOdometer().

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"
#include "uvfr_utils.h"
```

Data Structures

- struct uv19_pdu_settings

Typedefs

- typedef struct uv19_pdu_settings uv19_pdu_settings

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 Typedef Documentation

7.17.1.1 uv19_pdu_settings

```
typedef struct uv19_pdu_settings uv19_pdu_settings
```

7.17.2 Enumeration Type Documentation

7.17.2.1 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 37 of file pdu.h.

7.17.2.2 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 26 of file pdu.h.

7.17.3 Function Documentation

7.17.3.1 initPDU()

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

Definition at line 189 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().

7.17.3.2 PDU_disable_brake_light()

```
void PDU_disable_brake_light ( )
```

Definition at line 54 of file pdu.c.

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

7.17.3.3 PDU_disable_coolant_pump()

```
void PDU_disable_coolant_pump ( )
```

Definition at line 176 of file pdu.c.

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

7.17.3.4 PDU_disable_cooling_fans()

```
void PDU_disable_cooling_fans ( )
```

Definition at line 142 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.

7.17.3.5 PDU_disable_motor_controller()

```
void PDU_disable_motor_controller ( )
```

Definition at line 80 of file pdu.c.

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

7.17.3.6 PDU_disable_shutdown_circuit()

```
void PDU_disable_shutdown_circuit ( )
```

Definition at line 106 of file pdu.c.

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

7.17.3.7 PDU_enable_brake_light()

```
void PDU_enable_brake_light ( )
```

Definition at line 40 of file pdu.c.

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

7.17.3.8 PDU_enable_coolant_pump()

```
void PDU_enable_coolant_pump ( )
```

Definition at line 164 of file pdu.c.

References enable_coolant_pump_msg, Error_Handler(), hcan2, PDU_CAN_ID_Tx, TxData, TxHeader, and Tx↩ Mailbox.

7.17.3.9 PDU_enable_cooling_fans()

```
void PDU_enable_cooling_fans ( )
```

Definition at line 121 of file pdu.c.

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

7.17.3.10 PDU_enable_motor_controller()

```
void PDU_enable_motor_controller ( )
```

Definition at line 68 of file pdu.c.

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

7.17.3.11 PDU_enable_shutdown_circuit()

```
void PDU_enable_shutdown_circuit ( )
```

Definition at line 93 of file pdu.c.

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

7.17.3.12 PDU_speaker_chirp()

```
void PDU_speaker_chirp ( )
```

Definition at line 17 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 *))

  Function used to print the contents of the tree.
- rbnode * rbInsert (rbtree *rbt, void *data)

  Function that inserts data into the tree, and creates a new node.
- void * rbDelete (rbtree *rbt, rbnode *node, int keep)

  Deletes a node from the tree.
- int rbCheckOrder (rbtree *rbt, void *min, void *max)

  Function that validates that the order of the nodes in the tree is correct.
- int rbCheckBlackHeight (rbtree *rbt)

  Function that Checks the height of black nodes.

7.18.1 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.

Function that applies some function to a subtree.

Parameters

rbt	Pointer to an the `rbtree` you wish to apply the functions to
node	Pointer to the node that is the root of the tree you wish to apply functions to. This can be a subtree of another tree if needed
func	The function you would like to apply to the nodes. This takes in two parameters. The first is a pointer to the data of the node. The second parameter is a pointer to the shared cookie, which allows information to be preserved between different calls to `func`. This function returns an integer value, with 0 being ok, and the others being various error states.
cookie	This is a pointer to some space in memory that all of the calls to @func share, to preserve information.
order	This is a member of that specifies the order in which the tree will be traversed.

Attention

DANGER!! RECURSION!! Beware of stack memory usage, since most tasks are memory limited.

Enumerator

PREORDER	
INORDER	
POSTORDER	

Definition at line 18 of file rb_tree.h.

7.18.4 Function Documentation

7.18.4.1 rbApplyNode()

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

Definition at line 116 of file rb_tree.c.

References rbnode::data, INORDER, rbnode::left, POSTORDER, PREORDER, RB_NIL, and rbnode::right.

7.18.4.2 rbCheckBlackHeight()

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

Function that Checks the height of black nodes.

Attention

DANGER!! THIS FUNCTION IS RECURSIVE. This is intended to be used in a laptop debugging context. The VCU simply does not have enough memory to deal with recursion of this manner.

Deprecated Leftovers from laptop unit tests.

Definition at line 571 of file rb_tree.c.

References 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 )
```

Function that validates that the order of the nodes in the tree is correct.

Attention

DANGER!! THIS FUNCTION IS RECURSIVE. This is intended to be used in a laptop debugging context. The VCU simply does not have enough memory to deal with recursion of this manner.

Deprecated Leftovers from laptop unit tests.

Definition at line 545 of file rb_tree.c.

References 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.

Parameters

compare_func	A function that compares the data of two nodes. Accepts pointers to the data as parameters
destroy_func	The destructor function for the data, for safe disposal of dynamically allocated data

Definition at line 28 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::compare, rbtree::count, rbnode::data, rbtree::destroy, rbnode::left, rbtree::min, rbtree::nil, rbnode::parent, RB_NIL, rbnode::right, and rbtree::root.

7.18.4.5 rbDelete()

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

Deletes a node from the tree.

Parameters

rbt	Instance of a `rbtree` that we are removing the node from.
node	Pointer to the node that we would like to remove
keep	If `keep` is a truthy value, a pointer to the data of the node will be returned. Otherwise, the node and it's data will be destroyed.

Return values

If	`keep` is "true", this will return a pointer to the data held by the deleted node. Otherwise it will return NULL.

Definition at line 360 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::count, rbnode::data, 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 61 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.

Parameters

rbt	Pointer to the tree in which we would like to find the node.

Return values

Returns	a pointer to the node if the node is present in the tree. Otherwise, it will return NULL to indicate the node could not be found.

Definition at line 71 of file rb_tree.c.

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

7.18.4.8 rbInsert()

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

Function that inserts data into the tree, and creates a new node.

Parameters

rbt	Instance of a `rbtree` that we would like to insert data into
data	Pointer to the data we wish to insert

Return values

This	function returns a pointer to the `rbnode` that was added. The function will return NULL if the system is out of memory, or is otherwise unable to insert the node.

Definition at line 207 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::compare, rbtree::count, rbnode::data, rbtree::destroy, 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 )
```

Function used to print the contents of the tree.

Parameters

rbt	a pointer to the `rbtree` you wish to print.
print_func	A pointer to a print function specific to the data.

Attention

DANGER!! RECURSION!!

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

Definition at line 607 of file rb_tree.c.

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

7.18.4.10 rbSuccessor()

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

Definition at line 92 of file rb_tree.c.

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

Referenced by rbDelete().

7.19 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 ADC_IRQHandler (void)
  This function handles ADC1, ADC2 and ADC3 global interrupts.
- 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 ADC_IRQHandler()

```
void ADC_IRQHandler (
            void  )
```

This function handles ADC1, ADC2 and ADC3 global interrupts.

Definition at line 183 of file stm32f4xx_it.c.

References hadc1, and hadc2.

7.21.2.2 BusFault_Handler()

```
void BusFault_Handler (
            void  )
```

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

Definition at line 119 of file stm32f4xx_it.c.

7.21.2.3 CAN2_RX0_IRQHandler()

```
void CAN2_RX0_IRQHandler (
            void  )
```

This function handles CAN2 RX0 interrupts.

Definition at line 240 of file stm32f4xx_it.c.

References hcan2.

7.21.2.4 CAN2_RX1_IRQHandler()

```
void CAN2_RX1_IRQHandler (
             void  )
```

This function handles CAN2 RX1 interrupt.

Definition at line 254 of file stm32f4xx_it.c.

References hcan2.

7.21.2.5 CAN2_TX_IRQHandler()

```
void CAN2_TX_IRQHandler (
             void  )
```

This function handles CAN2 TX interrupts.

Definition at line 226 of file stm32f4xx_it.c.

References hcan2.

7.21.2.6 DebugMon_Handler()

```
void DebugMon_Handler (
             void  )
```

This function handles Debug monitor.

Definition at line 149 of file stm32f4xx_it.c.

7.21.2.7 DMA2_Stream0_IRQHandler()

```
void DMA2_Stream0_IRQHandler (
             void  )
```

This function handles DMA2 stream0 global interrupt.

Definition at line 212 of file stm32f4xx_it.c.

References hdma_adc1.

7.21.2.8 EXTI0_IRQHandler()

```
void EXTI0_IRQHandler (
             void  )
```

This function handles EXTI line0 interrupt.

Definition at line 169 of file stm32f4xx_it.c.

References Start_Button_Input_Pin.

7.21.2.9 HardFault_Handler()

```
void HardFault_Handler (
             void  )
```

This function handles Hard fault interrupt.

Definition at line 89 of file stm32f4xx_it.c.

7.21.2.10 MemManage_Handler()

```
void MemManage_Handler (
             void  )
```

This function handles Memory management fault.

Definition at line 104 of file stm32f4xx_it.c.

7.21.2.11 NMI_Handler()

```
void NMI_Handler (
             void  )
```

This function handles Non maskable interrupt.

Definition at line 74 of file stm32f4xx_it.c.

7.21.2.12 TIM1_UP_TIM10_IRQHandler()

```
void TIM1_UP_TIM10_IRQHandler (
            void  )
```

This function handles TIM1 update interrupt and TIM10 global interrupt.

Definition at line 198 of file stm32f4xx_it.c.

References htim1.

7.21.2.13 UsageFault_Handler()

```
void UsageFault_Handler (
            void  )
```

This function handles Undefined instruction or illegal state.

Definition at line 134 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, insertCAN↩MessageHandler(), killSelf(), uv_CAN_msg::msg_id, suspendSelf(), uv_task_info::task_period, testfunc(), testfunc2(), TxData, TxHeader, UV_KILL_CMD, UV_SUSPEND_CMD, and uvTaskDelayUntil.

Referenced by initTempMonitor().

7.23 Core/Inc/tim.h File Reference

This file contains all the function prototypes for the tim.c file.

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

Functions

- void MX_TIM3_Init (void)

Variables

- TIM_HandleTypeDef htim3

7.23.1 Detailed Description

This file contains all the function prototypes for the tim.c file.

Attention

Copyright (c) 2024 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

7.23.2 Function Documentation

7.23.2.1 MX_TIM3_Init()

```
void MX_TIM3_Init (
             void  )
```

Definition at line 30 of file tim.c.

References Error_Handler(), and htim3.

Referenced by main().

7.23.3 Variable Documentation

7.23.3.1 htim3

```
TIM_HandleTypeDef htim3
```

Definition at line 27 of file tim.c.

Referenced by HAL_TIM_PeriodElapsedCallback(), and MX_TIM3_Init().

7.24 Core/Inc/uvfr_global_config.h File Reference

Macros

- #define UV19_PDU 1
- #define ECUMASTER_PMU 0
- #define STM32_F407 1
- #define STM32_H7xx 0
- #define FIRMWARE_MAJOR_RELEASE 0
- #define FIRMWARE_MINOR_RELEASE 1
- #define FIRMWARE_PATCH_NUM 0
- #define UV_MALLOC_LIMIT ((size_t)1024)
- #define USE_OS_MEM_MGMT 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 FIRMWARE_MAJOR_RELEASE

```
#define FIRMWARE_MAJOR_RELEASE 0
```

Definition at line 18 of file uvfr_global_config.h.

7.24.1.3 FIRMWARE_MINOR_RELEASE

```
#define FIRMWARE_MINOR_RELEASE 1
```

Definition at line 19 of file uvfr_global_config.h.

7.24.1.4 FIRMWARE_PATCH_NUM

```
#define FIRMWARE_PATCH_NUM 0
```

Definition at line 20 of file uvfr_global_config.h.

7.24.1.5 STM32_F407

```
#define STM32_F407 1
```

Definition at line 15 of file uvfr_global_config.h.

7.24.1.6 STM32_H7xx

```
#define STM32_H7xx 0
```

Definition at line 16 of file uvfr_global_config.h.

7.24.1.7 USE_OS_MEM_MGMT

```
#define USE_OS_MEM_MGMT 0
```

Definition at line 30 of file uvfr_global_config.h.

7.24.1.8 UV19_PDU

```
#define UV19_PDU 1
```

Definition at line 12 of file uvfr_global_config.h.

7.24.1.9 UV_MALLOC_LIMIT

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

Definition at line 26 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
- struct motor_controller_settings

Macros

- #define ENABLE_FLASH_SETTINGS 0
- #define START_OF_USER_FLASH &_s_uvdata
- #define TOP_OF_USER_FLASH &_e_uvdata
- #define TOP_OF_FLASH_SBLOCK (void∗)0x080FFFFF
- #define SBLOCK_CSR_OFFSET 0x0C00
- #define SYS_DATA_OFFSET 0x00
- #define SIZE_OF_MGROUP 0x0100
- #define SBLOCK_CRC_REGION_OFFSET
- #define SIZEOF_USER_FLASH 4096
- #define SIZEOF_SBLOCK 4096
- #define FLASH_SBLOCK_START START_OF_USER_FLASH
- #define isPtrToFlash(p) ((p>=0x08000000)&&(p<TOP_OF_USER_FLASH))
- #define GENERAL_VEH_INFO_MGROUP 0
- #define GENERAL_VEH_INFO_OFFSET 32
- #define OS_SETTINGS_MGROUP 0
- #define OS_SETTINGS_OFFSET 128
- #define OS_SETTINGS_ADDR ((void∗)(START_OF_USER_FLASH + OS_SETTINGS_MGROUP∗256 + OS_SETTINGS_OFFSET))
- #define MOTOR_MGROUP 1
- #define MOTOR_OFFSET 0
- #define MOTOR_ADDR (struct motor_controller_settings∗)(START_OF_USER_FLASH + MOTOR_MGROUP∗256 + MOTOR_OFFSET)
- #define DRIVING_MGROUP 2
- #define DRIVING_OFFSET 0
- #define DRIVING_ADDR ((driving_loop_args∗)(START_OF_USER_FLASH + DRIVING_MGROUP + DRIVING_OFFSET))
- #define BMS_MGROUP 4
- #define BMS_OFFSET 0
- #define BMS_ADDR NULL
- #define IMD_MGROUP 4
- #define IMD_OFFSET 128
- #define IMD_ADDR ((void∗)(START_OF_USER_FLASH + IMD_MGROUP∗256 + IMD_OFFSET))
- #define PDU_MGROUP 5
- #define PDU_OFFSET 0
- #define PDU_ADDR ((void∗)(START_OF_USER_FLASH + PDU_MGROUP∗256 + PDU_OFFSET))
- #define DAQ_HEAD_MGROUP 5
- #define DAQ_HEAD_OFFSET 128
- #define DAQ_HEAD_ADDR ((daq_loop_args∗)(START_OF_USER_FLASH + DAQ_HEAD_MGROUP∗256 + DAQ_HEAD_OFFSET))
- #define DAQ_PARAMS1_MGROUP 6
- #define DAQ_PARAMS1_OFFSET 0
- #define DAQ_PARAMS1_ADDR ((void∗)(START_OF_USER_FLASH + DAQ_PARAMS1_MGROUP∗256 + DAQ_PARAMS1_OFFSET))
- #define DAQ_PARAMS2_MGROUP 7
- #define DAQ_PARAMS2_OFFSET 0
- #define DAQ_PARAMS2_ADDR
- #define DAQ_PARAMS3_MGROUP 8
- #define DAQ_PARAMS3_OFFSET 0
- #define DAQ_PARAMS3_ADDR
- #define PERSISTENT_DATA_MGROUP 13
- #define CRC_MGROUP1 14
- #define CRC_MGROUP2 15
- #define SETTING_BRANCH_SIZE (256∗10)
- #define CRC_POLY 0x04C11DB7
- #define MAGIC_NUMBER 0xEFBEADDE

Typedefs

- typedef struct veh_gen_info veh_gen_info
- typedef enum uv_status_t uv_status
- typedef struct uv_vehicle_settings uv_vehicle_settings
- typedef struct motor_controller_settings motor_controller_settings

Enumerations

- enum laptop_CMD {
  HANDSHAKE = 0x01, ENTER_PROGRAMMING_MODE = 0x02, REQUEST_VCU_STATUS = 0x03,
  CLEAR_FAULTS = 0x04,
  REQUEST_VCU_FIRMWARE_VERSION = 0x05, GENERIC_ACK = 0x10, CANNOT_PERFORM_REQUEST
  = 0x11, SET_SPECIFIC_PARAM = 0x20,
  END_OF_SPECIFIC_PARAMS = 0x3F, REQUEST_ALL_SETTINGS = 0x40, REQUEST_ALL_JOURNAL_ENTRIES
  = 0x41, REQUEST_JOURNAL_ENTRIES_BY_TIME = 0x42,
  REQUEST_SPECIFIC_SETTING = 0x43, SAVE_AND_APPLY_NEW_SETTINGS = 0x80, DISCARD_NEW_SETTINGS
  = 0x82, DISCARD_NEW_SETTINGS_AND_EXIT = 0x83,
  FORCE_RESTORE_FACTORY_DEFAULT = 0x84 }

Functions

- uv_status uvConfigSettingTask (void *args)

  Function to setup the parameters of the setting setter task.
- void nukeSettings (uv_vehicle_settings **settings_to_delete)
- uv_status uvValidateSettingsFromFlash ()
- enum uv_status_t uvSettingsInit ()

  this function does one thing, and one thing only, it checks if we have custom settings, then it attempts to get them. If it fails, then we revert to factory defaults.
- uv_status setupDefaultSettings ()

  Function that allocates the neccessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.
- uv_status uvSaveSettingsToFlash ()
- uv_status uvComputeMemRegionChecksum (void *sblock, uint32_t *csums, char mregion)
- uv_status uvComputeSettingsCheckSums (void *sblock)
- uv_status uvValidateChecksums (void *sblock)

Variables

- uv_vehicle_settings * current_vehicle_settings

7.25.1 Macro Definition Documentation

7.25.1.1 BMS_ADDR

```
#define BMS_ADDR NULL
```

Definition at line 60 of file uvfr_settings.h.

7.25.1.2 BMS_MGROUP

```
#define BMS_MGROUP 4
```

Definition at line 58 of file uvfr_settings.h.

7.25.1.3 BMS_OFFSET

```
#define BMS_OFFSET 0
```

Definition at line 59 of file uvfr_settings.h.

7.25.1.4 CRC_MGROUP1

```
#define CRC_MGROUP1 14
```

Definition at line 87 of file uvfr_settings.h.

7.25.1.5 CRC_MGROUP2

```
#define CRC_MGROUP2 15
```

Definition at line 88 of file uvfr_settings.h.

7.25.1.6 CRC_POLY

```
#define CRC_POLY 0x04C11DB7
```

Definition at line 92 of file uvfr_settings.h.

7.25.1.7 DAQ_HEAD_ADDR

```
#define DAQ_HEAD_ADDR ((daq_loop_args*)(START_OF_USER_FLASH + DAQ_HEAD_MGROUP*256 + DAQ_HEAD_OFFSET))
```

Definition at line 72 of file uvfr_settings.h.

7.25.1.8 DAQ_HEAD_MGROUP

```
#define DAQ_HEAD_MGROUP 5
```

Definition at line 70 of file uvfr_settings.h.

7.25.1.9 DAQ_HEAD_OFFSET

```
#define DAQ_HEAD_OFFSET 128
```

Definition at line 71 of file uvfr_settings.h.

7.25.1.10 DAQ_PARAMS1_ADDR

```
#define DAQ_PARAMS1_ADDR ((void*)(START_OF_USER_FLASH + DAQ_PARAMS1_MGROUP*256 + DAQ_PARAMS1_OFFSET))
```

Definition at line 76 of file uvfr_settings.h.

7.25.1.11 DAQ_PARAMS1_MGROUP

```
#define DAQ_PARAMS1_MGROUP 6
```

Definition at line 74 of file uvfr_settings.h.

7.25.1.12 DAQ_PARAMS1_OFFSET

```
#define DAQ_PARAMS1_OFFSET 0
```

Definition at line 75 of file uvfr_settings.h.

7.25.1.13 DAQ_PARAMS2_ADDR

```
#define DAQ_PARAMS2_ADDR
```

Definition at line 80 of file uvfr_settings.h.

7.25.1.14 DAQ_PARAMS2_MGROUP

```
#define DAQ_PARAMS2_MGROUP 7
```

Definition at line 78 of file uvfr_settings.h.

7.25.1.15 DAQ_PARAMS2_OFFSET

```
#define DAQ_PARAMS2_OFFSET 0
```

Definition at line 79 of file uvfr_settings.h.

7.25.1.16 DAQ_PARAMS3_ADDR

```
#define DAQ_PARAMS3_ADDR
```

Definition at line 84 of file uvfr_settings.h.

7.25.1.17 DAQ_PARAMS3_MGROUP

```
#define DAQ_PARAMS3_MGROUP 8
```

Definition at line 82 of file uvfr_settings.h.

7.25.1.18 DAQ_PARAMS3_OFFSET

```
#define DAQ_PARAMS3_OFFSET 0
```

Definition at line 83 of file uvfr_settings.h.

7.25.1.19 DRIVING_ADDR

```
#define DRIVING_ADDR ((driving_loop_args*)(START_OF_USER_FLASH + DRIVING_MGROUP + DRIVING_OFFSET))
```

Definition at line 56 of file uvfr_settings.h.

7.25.1.20 DRIVING_MGROUP

```
#define DRIVING_MGROUP 2
```

Definition at line 54 of file uvfr_settings.h.

7.25.1.21 DRIVING_OFFSET

```
#define DRIVING_OFFSET 0
```

Definition at line 55 of file uvfr_settings.h.

7.25.1.22 ENABLE_FLASH_SETTINGS

```
#define ENABLE_FLASH_SETTINGS 0
```

Definition at line 21 of file uvfr_settings.h.

7.25.1.23 FLASH_SBLOCK_START

```
#define FLASH_SBLOCK_START START_OF_USER_FLASH
```

Definition at line 37 of file uvfr_settings.h.

7.25.1.24 GENERAL_VEH_INFO_MGROUP

```
#define GENERAL_VEH_INFO_MGROUP 0
```

Definition at line 43 of file uvfr_settings.h.

7.25.1.25 GENERAL_VEH_INFO_OFFSET

```
#define GENERAL_VEH_INFO_OFFSET 32
```

Definition at line 44 of file uvfr_settings.h.

7.25.1.26 IMD_ADDR

```
#define IMD_ADDR ((void*)(START_OF_USER_FLASH + IMD_MGROUP*256 + IMD_OFFSET))
```

Definition at line 64 of file uvfr_settings.h.

7.25.1.27 IMD_MGROUP

```
#define IMD_MGROUP 4
```

Definition at line 62 of file uvfr_settings.h.

7.25.1.28 IMD_OFFSET

```
#define IMD_OFFSET 128
```

Definition at line 63 of file uvfr_settings.h.

7.25.1.29 isPtrToFlash

```
#define isPtrToFlash(
            p ) ((p>=0x08000000)&&(p<TOP_OF_USER_FLASH))
```

Definition at line 40 of file uvfr_settings.h.

7.25.1.30 MAGIC_NUMBER

```
#define MAGIC_NUMBER 0xEFBEADDE
```

Definition at line 96 of file uvfr_settings.h.

7.25.1.31 MOTOR_ADDR

```
#define MOTOR_ADDR (struct motor_controller_settings*)(START_OF_USER_FLASH + MOTOR_MGROUP*256 +
MOTOR_OFFSET)
```

Definition at line 52 of file uvfr_settings.h.

7.25.1.32 MOTOR_MGROUP

```
#define MOTOR_MGROUP 1
```

Definition at line 50 of file uvfr_settings.h.

7.25.1.33 MOTOR_OFFSET

```
#define MOTOR_OFFSET 0
```

Definition at line 51 of file uvfr_settings.h.

7.25.1.34 OS_SETTINGS_ADDR

```
#define OS_SETTINGS_ADDR ((void*)(START_OF_USER_FLASH + OS_SETTINGS_MGROUP*256 + OS_SETTINGS_OFFSET))
```

Definition at line 48 of file uvfr_settings.h.

7.25.1.35 OS_SETTINGS_MGROUP

```
#define OS_SETTINGS_MGROUP 0
```

Definition at line 46 of file uvfr_settings.h.

7.25.1.36 OS_SETTINGS_OFFSET

```
#define OS_SETTINGS_OFFSET 128
```

Definition at line 47 of file uvfr_settings.h.

7.25.1.37 PDU_ADDR

```
#define PDU_ADDR ((void*)(START_OF_USER_FLASH + PDU_MGROUP*256 + PDU_OFFSET))
```

Definition at line 68 of file uvfr_settings.h.

7.25.1.38 PDU_MGROUP

```
#define PDU_MGROUP 5
```

Definition at line 66 of file uvfr_settings.h.

7.25.1.39 PDU_OFFSET

```
#define PDU_OFFSET 0
```

Definition at line 67 of file uvfr_settings.h.

7.25.1.40 PERSISTENT_DATA_MGROUP

```
#define PERSISTENT_DATA_MGROUP 13
```

Definition at line 86 of file uvfr_settings.h.

7.25.1.41 SBLOCK_CRC_REGION_OFFSET

```
#define SBLOCK_CRC_REGION_OFFSET
```

Definition at line 31 of file uvfr_settings.h.

7.25.1.42 SBLOCK_CSR_OFFSET

```
#define SBLOCK_CSR_OFFSET 0x0C00
```

Definition at line 28 of file uvfr_settings.h.

7.25.1.43 SETTING_BRANCH_SIZE

```
#define SETTING_BRANCH_SIZE (256*10)
```

Definition at line 90 of file uvfr_settings.h.

7.25.1.44 SIZE_OF_MGROUP

```
#define SIZE_OF_MGROUP 0x0100
```

Definition at line 30 of file uvfr_settings.h.

7.25.1.45 SIZEOF_SBLOCK

```
#define SIZEOF_SBLOCK 4096
```

Definition at line 35 of file uvfr_settings.h.

7.25.1.46 SIZEOF_USER_FLASH

```
#define SIZEOF_USER_FLASH 4096
```

Definition at line 34 of file uvfr_settings.h.

7.25.1.47 START_OF_USER_FLASH

```
#define START_OF_USER_FLASH &_s_uvdata
```

Definition at line 25 of file uvfr_settings.h.

7.25.1.48 SYS_DATA_OFFSET

```
#define SYS_DATA_OFFSET 0x00
```

Definition at line 29 of file uvfr_settings.h.

7.25.1.49 TOP_OF_FLASH_SBLOCK

```
#define TOP_OF_FLASH_SBLOCK (void*)0x080FFFFF
```

Definition at line 27 of file uvfr_settings.h.

7.25.1.50 TOP_OF_USER_FLASH

```
#define TOP_OF_USER_FLASH &_e_uvdata
```

Definition at line 26 of file uvfr_settings.h.

7.25.2 Typedef Documentation

7.25.2.1 motor_controller_settings

```
typedef struct motor_controller_settings motor_controller_settings
```

7.25.2.2 uv_status

```
typedef enum uv_status_t uv_status
```

Definition at line 112 of file uvfr_settings.h.

7.25.2.3 uv_vehicle_settings

```
typedef struct uv_vehicle_settings uv_vehicle_settings
```

7.25.2.4 veh_gen_info

```
typedef struct veh_gen_info veh_gen_info
```

7.25.3 Enumeration Type Documentation

7.25.3.1 laptop_CMD

```
enum laptop_CMD
```

Enumerator

HANDSHAKE	
ENTER_PROGRAMMING_MODE	
REQUEST_VCU_STATUS	
CLEAR_FAULTS	
REQUEST_VCU_FIRMWARE_VERSION	
GENERIC_ACK	
CANNOT_PERFORM_REQUEST	
SET_SPECIFIC_PARAM	
END_OF_SPECIFIC_PARAMS	
REQUEST_ALL_SETTINGS	
REQUEST_ALL_JOURNAL_ENTRIES	
REQUEST_JOURNAL_ENTRIES_BY_TIME	
REQUEST_SPECIFIC_SETTING	
SAVE_AND_APPLY_NEW_SETTINGS	
DISCARD_NEW_SETTINGS	
DISCARD_NEW_SETTINGS_AND_EXIT	
FORCE_RESTORE_FACTORY_DEFAULT	

Definition at line 114 of file uvfr_settings.h.

7.25.4 Function Documentation

7.25.4.1 nukeSettings()

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

Definition at line 260 of file uvfr_settings.c.

7.25.4.2 setupDefaultSettings()

```
uv_status setupDefaultSettings ( )
```

Function that allocates the neccessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.

Definition at line 232 of file uvfr_settings.c.

References uv_vehicle_settings::bms_settings, current_vehicle_settings, uv_vehicle_settings::daq_param_list, uv_vehicle_settings::daq_settings, default_bms_settings, default_daq_settings, default_datapoints, default_dl_↩ settings, default_imd_settings, default_os_settings, default_pdu_settings, default_vehicle, uv_vehicle_settings↩ ::driving_loop_settings, uv_vehicle_settings::flags, uv_vehicle_settings::imd_settings, mc_default_settings, uv_↩ vehicle_settings::mc_settings, uv_vehicle_settings::os_settings, uv_vehicle_settings::pdu_settings, UV_ERROR, UV_OK, and uv_vehicle_settings::veh_info.

Referenced by uvSettingsInit().

7.25.4.3 uvComputeMemRegionChecksum()

```
uv_status uvComputeMemRegionChecksum (
            void * sblock,
            uint32_t * csums,
            char mregion )
```

Definition at line 463 of file uvfr_settings.c.

References UV_OK, and uvComputeChunkChecksum().

Referenced by uvComputeSettingsChecksums().

7.25.4.4 uvComputeSettingsCheckSums()

```
uv_status uvComputeSettingsCheckSums (
            void * sblock )
```

7.25.4.5 uvConfigSettingTask()

```
uv_status uvConfigSettingTask (
            void * args )
```

Function to setup the parameters of the setting setter task.

Definition at line 305 of file uvfr_settings.c.

References ABOVE_NORMAL, uv_task_info::active_states, uv_task_info::deletion_states, PROGRAMMING, uv_task_info::stack_size, uv_task_info::suspension_states, uv_task_info::task_args, uv_task_info::task_function, uv_task_info::task_name, uv_task_info::task_period, uv_task_info::task_priority, UV_ERROR, UV_OK, uvCreateTask(), and uvSettingsProgrammerTask().

Referenced by uvInitStateEngine().

7.25.4.6 uvSaveSettingsToFlash()

```
uv_status uvSaveSettingsToFlash ( )
```

7.25.4.7 uvSettingsInit()

```
enum uv_status_t uvSettingsInit ( )
```

this function does one thing, and one thing only, it checks if we have custom settings, then it attempts to get them. If it fails, then we revert to factory defaults.

Definition at line 334 of file uvfr_settings.c.

References __uvInitPanic(), current_vehicle_settings, handleIncomingLaptopMsg(), insertCANMessageHandler(), SETTING_BRANCH_SIZE, setupDefaultSettings(), START_OF_USER_FLASH, UV_ERROR, UV_OK, uvLoad↩SettingsFromFlash(), uvResetFlashToDefault(), uvSendCanMSG(), uvValidateFlashSettings(), and vcu_ack_msg.

Referenced by uvInit().

7.25.4.8 uvValidateChecksums()

```
uv_status uvValidateChecksums (
            void * sblock )
```

Definition at line 495 of file uvfr_settings.c.

References UV_ERROR, UV_OK, and uvComputeSettingsChecksums().

7.25.4.9 uvValidateSettingsFromFlash()

```
uv_status uvValidateSettingsFromFlash ( )
```

7.25.5 Variable Documentation

7.25.5.1 current_vehicle_settings

```
uv_vehicle_settings* current_vehicle_settings
```

Definition at line 20 of file uvfr_settings.c.

Referenced by configureDaqSubTasks(), initDaqTask(), initDrivingLoop(), setupDefaultSettings(), uvInit(), uvLoad↩SettingsFromFlash(), uvSettingsInit(), and uvStartStateMachine().

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.
- enum os_flag { UV_OS_LOG_MEM = 0x01, UV_OS_LOG_TASK_END_TIME = 0x02, UV_OS_ATTEMPT_RESTART_NC_TAS
  = 0x04, UV_OS_ENABLE_NONCRIT_TASK_THROTTLE = 0x08 }

Functions

- void uvTaskPeriodEnd (uv_task_info *t)

  Function called at the end of the task period.
- struct uv_task_info * uvCreateTask ()

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

  Create a new service task, because fuck you, thats why.
- struct uv_task_info * 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&0x00FF0000)>>16; x[i+3]=(d&0xFF000000)>>24
- #define serializeBigE16(x, d, i) x[i+1]=d&0x00FF; x[i]=(d&0xFF00)>>8
- #define serializeBigE32(x, d, i) x[i+3]=d&0x000000FF; x[i+2]=(d&0x0000FF00)>>8; x[i+1]=(d&0x00FF0000)>>16; x[i]=(d&0xFF000000)>>24
- #define setBits(x, msk, data) x=(x&(~msk)|data)

  macro to set bits of an int without touching the ones we dont want to edit
- #define isPowerOfTwo(x) (x&&(!(x&(x-1))))

  Returns a truthy value if "x" is a power of two.
- #define safePtrRead(x) (*((x)?x:uvPanic("nullptr_deref",0)))

  lil treat to help us avoid the dreaded null pointer dereference
- #define safePtrWrite(p, x) (*((p)?p:&x))
- #define false 0
- #define true !false
- #define MAX_INIT_TIME 2500
- #define INIT_CHECK_PERIOD 100
- #define UV_CAN1
- #define UV_CAN2
- #define USE_OLED_DEBUG 1
- #define UV_CAN_EXTENDED_ID 0x01
- #define UV_CAN_CHANNEL_MASK 0b00000110
- #define UV_CAN_DYNAMIC_MEM 0b00001000

Typedefs

- typedef uint8_t bool
- typedef uint8_t uv_task_id
- typedef enum uv_task_cmd_e uv_task_cmd
- typedef uint8_t uv_ext_device_id
- typedef uint32_t uv_timespan_ms
- typedef enum uv_status_t uv_status

  This is meant to be a return type from functions that indicates what is actually going on.
- typedef enum access_control_t access_control_type
- typedef enum uv_msg_type_t uv_msg_type

  Enum dictating the meaning of a generic message.
- typedef union access_control_info access_control_info
- typedef struct uv_CAN_msg uv_CAN_msg

  Representative of a CAN message.
- typedef struct uv_init_struct uv_init_struct
- typedef struct uv_task_msg_t uv_task_msg

  Struct containing a message between two tasks.
- typedef struct p_status p_status
- typedef struct uv_init_task_args uv_init_task_args

  Struct designed to act like the `uv_task_info` struct, but for the initialisation tasks. As a result it takes fewer arguments.
- typedef struct uv_internal_params uv_internal_params

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

  Struct representing the response of one of the initialization tasks.

Enumerations

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

  This is meant to be a return type from functions that indicates what is actually going on.
- enum data_type {
  UV_UINT8 = 0, UV_INT8 = 1, UV_UINT16 = 2, UV_INT16 = 3,
  UV_UINT32 = 4, UV_INT32 = 5, UV_FLOAT = 6, UV_DOUBLE = 7,
  UV_INT64 = 8, UV_UINT64 = 9, UV_STRING = 10 }

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

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

  Enum dictating the meaning of a generic message.

Functions

- void uvInit (void *arguments)

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

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

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

Variables

- uv_internal_params global_context
- const uint8_t data_size [ ]

7.27.1 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	

Enumerator

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	
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 296 of file uvfr_utils.c.

Referenced by uvInit(), uvInitStateEngine(), uvSettingsInit(), 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
initADCTask(); //START THE ADCs
```

Wait for all the spawned in tasks to do their thing. This should not take that long, but we wanna be sure that everything is chill If we are say, missing a BMS, then it will not allow you to proceed past the initialisation step This is handled by a message buffer, that takes inputs from all of the tasks

We allocate space for a response from the initialization.

Clean up, clean up, everybody clean up, clean up, clean up, everybody do your share! The following code cleans up all the threads that were running, and free up used memory

Definition at line 46 of file uvfr_utils.c.

References __uvInitPanic(), uv_task_info::active_states, BMS_Init(), uv_vehicle_settings::bms_settings, CAN_RX_DAEMON_NAME, CAN_TX_DAEMON_NAME, CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), 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, initADCTask(), initIMD(), initPDU(), MAX_INIT_TIME, uv_vehicle_settings::mc_settings, MC_Startup(), uv_init_task_args::meta_task_handle, uv_init_task_response::nchar, uv_init_task_args::specific_args, uv_init_task_response::status, uv_task_info::task_function, uv_task_info::task_name, UV_OK, UV_READY, uvCreateServiceTask(), uvInitStateEngine(), uvSettingsInit(), uvStartStateMachine(), and uvStartTask().

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 426 of file uvfr_utils.c.

References UV_ERROR, UV_OK, and UV_WARNING.

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

7.27.6 Variable Documentation

7.27.6.1 data_size

```
const uint8_t data_size[]
```

Definition at line 17 of file uvfr_utils.c.

Referenced by insertParamToRegister(), and uvSettingsProgrammerTask().

7.27.6.2 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"
#include "uvfr_utils.h"
#include "cmsis_os.h"
```

Macros

- #define DEBUG_TOGGLE_LED() HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15)

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)
- void StartADCTask (void *argument)
- void processADCBuffer ()
- void initADCTask (void)

Variables

- volatile uint32_t adc_buf1 [4]
- uint16_t adc1_APPS1
- uint16_t adc1_APPS2
- uint16_t adc1_BPS1
- uint16_t adc1_BPS2
- 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 Macro Definition Documentation

7.29.2.1 DEBUG_TOGGLE_LED

```
#define DEBUG_TOGGLE_LED( ) HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15)
```

Definition at line 305 of file adc.c.

7.29.3 Function Documentation

7.29.3.1 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

Uncomment the line below to disable the "ADC_IRQn" interrupt Be aware, disabling shared interrupt may affect other IPs

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

Uncomment the line below to disable the "ADC_IRQn" interrupt Be aware, disabling shared interrupt may affect other IPs

Definition at line 238 of file adc.c.

7.29.3.2 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 161 of file adc.c.

References Error_Handler(), and hdma_adc1.

7.29.3.3 initADCTask()

```
void initADCTask (
            void  )
```

Definition at line 358 of file adc.c.

References Error_Handler(), and StartADCTask().

Referenced by uvInit().

7.29.3.4 MX_ADC1_Init()

```
void MX_ADC1_Init (
            void  )
```

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

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

Definition at line 40 of file adc.c.

References Error_Handler(), and hadc1.

Referenced by main().

7.29.3.5 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 114 of file adc.c.

References Error_Handler(), and hadc2.

Referenced by main().

7.29.3.6 processADCBuffer()

```
void processADCBuffer ( )
```

Definition at line 320 of file adc.c.

References adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, and adc_buf1.

Referenced by HAL_ADC_ConvCpltCallback().

7.29.3.7 StartADCTask()

```
void StartADCTask (
             void * argument )
```

Definition at line 307 of file adc.c.

References adc_buf1, DEBUG_TOGGLE_LED, and hadc1.

Referenced by initADCTask().

7.29.4 Variable Documentation

7.29.4.1 adc1_APPS1

```
uint16_t adc1_APPS1
```

Definition at line 64 of file main.c.

Referenced by processADCBuffer().

7.29.4.2 adc1_APPS2

```
uint16_t adc1_APPS2
```

Definition at line 65 of file main.c.

Referenced by processADCBuffer().

7.29.4.3 adc1_BPS1

```
uint16_t adc1_BPS1
```

Definition at line 66 of file main.c.

Referenced by processADCBuffer().

7.29.4.4 adc1_BPS2

```
uint16_t adc1_BPS2
```

Definition at line 67 of file main.c.

Referenced by processADCBuffer().

7.29.4.5 adc_buf1

```
volatile uint32_t adc_buf1[4]
```

Definition at line 62 of file main.c.

Referenced by processADCBuffer(), and StartADCTask().

7.29.4.6 hadc1

```
ADC_HandleTypeDef hadc1
```

Definition at line 35 of file adc.c.

Referenced by ADC_IRQHandler(), HAL_ADC_LevelOutOfWindowCallback(), MX_ADC1_Init(), and StartADC↩ Task().

7.29.4.7 hadc2

```
ADC_HandleTypeDef hadc2
```

Definition at line 36 of file adc.c.

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

7.29.4.8 hdma_adc1

```
DMA_HandleTypeDef hdma_adc1
```

Definition at line 37 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"
#include "uvfr_utils.h"
```

Functions

- void handleBMSVoltageMsg (uv_CAN_msg *msg)
- void BMS_Init (void *args)

Variables

- bms_settings_t default_bms_settings

7.30.1 Function Documentation

7.30.1.1 BMS_Init()

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

Definition at line 22 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().

7.30.1.2 handleBMSVoltageMsg()

```
void handleBMSVoltageMsg (
            uv_CAN_msg * msg )
```

Definition at line 17 of file bms.c.

7.30.2 Variable Documentation

7.30.2.1 default_bms_settings

```
bms_settings_t default_bms_settings
```

Initial value:

```
= {
    .BMS_CAN_timeout = 100,
    .max_temp = 60
}
```

Definition at line 12 of file bms.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

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 632 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 uvInit().

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 563 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 uvInit().

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_IR↩QHandler(), CANbusTxSvcDaemon(), HAL_CAN_RxFifo0MsgPendingCallback(), 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(), 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.

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(), 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"
```

Data Structures

- struct daq_param_list_node
- struct daq_child_task

Macros

- #define _SRC_UVFR_DAQ

Typedefs

- typedef struct daq_param_list_node daq_param_list_node
- typedef struct daq_child_task daq_child_task

Functions

- void daqSubTask (void *args)
- uv_status associateDaqParamWithVar (uint16_t paramID, void *var)
- static void insertParamToParamList (daq_param_list_node *node, daq_param_list_node **list)
- uv_status insertParamToRegister (daq_param_list_node *node, daq_datapoint *datapoint)
- uv_status configureDaqSubTasks ()

  This pre-allocates parameters to one of the daq subtasks.
- uv_status startDaqSubTasks ()

  Function that starts up all the subtasks.
- uv_status stopDaqSubTasks ()

  Function that shuts down the subtasks.
- uv_status initDaqTask (void *args)

  initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks
- void daqMasterTask (void *args)

  Controls the Daq.
- static void sendDaqMsg (daq_param_list_node *param)
- static void sendAllParamsFromList (daq_param_list_node *list)

Variables

- daq_loop_args * curr_daq_settings = NULL
- daq_loop_args default_daq_settings
- daq_datapoint default_datapoints [ ]
- static void * param_ptrs [MAX_LOGGABLE_PARAMS]
- static daq_child_task * daq_tlist = NULL
- static daq_datapoint * datapoints = NULL
- static daq_param_list_node * param_bank = NULL
- uv_CAN_msg tmp_daq_msg

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 Typedef Documentation

7.33.2.1 daq_child_task

```
typedef struct daq_child_task daq_child_task
```

7.33.2.2 daq_param_list_node

```
typedef struct daq_param_list_node daq_param_list_node
```

7.33.3 Function Documentation

7.33.3.1 associateDaqParamWithVar()

```
uv_status associateDaqParamWithVar (
            uint16_t paramID,
            void * var )
```

Definition at line 99 of file daq.c.

References MAX_LOGGABLE_PARAMS, param_ptrs, UV_ERROR, UV_OK, and uvIsPTRValid().

Referenced by initDrivingLoop().

7.33.3.2 configureDaqSubTasks()

```
uv_status configureDaqSubTasks ( )
```

This pre-allocates parameters to one of the daq subtasks.

Definition at line 184 of file daq.c.

References curr_daq_settings, current_vehicle_settings, uv_vehicle_settings::daq_param_list, insertParamTo↩Register(), param_bank, daq_loop_args::total_params_logged, UV_ERROR, and UV_OK.

Referenced by initDaqTask().

7.33.3.3 daqMasterTask()

```
void daqMasterTask (
            void * args )
```

Controls the Daq.

These here lines set the delay. This task executes exactly at the period specified, regardless of how long the task execution actually takes

```
 */
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
//TickType_t last_time = xTaskGetTickCount();       /**
```

Definition at line 291 of file daq.c.

References uv_task_info::cmd_data, killSelf(), startDaqSubTasks(), stopDaqSubTasks(), suspendSelf(), uv_task_info::task_period, UV_KILL_CMD, UV_OK, UV_SUSPEND_CMD, and uvTaskDelay.

Referenced by initDaqTask().

7.33.3.4 daqSubTask()

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

Definition at line 381 of file daq.c.

References daq_child_task::param_list, daq_child_task::period, and sendAllParamsFromList().

Referenced by startDaqSubTasks().

7.33.3.5 initDaqTask()

```
uv_status initDaqTask (
            void * args )
```

initializes the master DAQ task, all that fun stuff. This task probably manages a while plethora of smaller tasks

This is a fairly standard function. Here are the things that it does in order:

Step 1: Get Daq settings.

Step 2: Create and configure DAQ task.

Step 3: Read which parameters we want to read.

Step 4: Generate Subtask Metadata

Step 5: Assign params to subtasks

Definition at line 252 of file daq.c.

References uv_task_info::active_states, configureDaqSubTasks(), curr_daq_settings, current_vehicle_settings, daq_loop_args::daq_child_priority, uv_vehicle_settings::daq_param_list, uv_vehicle_settings::daq_settings, daqMasterTask(), datapoints, 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().

7.33.3.6 insertParamToParamList()

```
static void insertParamToParamList (
            daq_param_list_node * node,
            daq_param_list_node ** list )  [inline], [static]
```

Definition at line 116 of file daq.c.

References daq_param_list_node::next.

Referenced by insertParamToRegister().

7.33.3.7 insertParamToRegister()

```
uv_status insertParamToRegister (
            daq_param_list_node * node,
            daq_datapoint * datapoint )
```

Definition at line 136 of file daq.c.

References daq_param_list_node::can_id, daq_datapoint::can_id, daq_tlist, data_size, insertParamToParam↩List(), daq_param_list_node::next, daq_child_task::next_task, daq_param_list_node::param, daq_datapoint↩::param, daq_child_task::param_list, daq_child_task::period, daq_datapoint::period, daq_param_list_node::size, daq_datapoint::type, and UV_OK.

Referenced by configureDaqSubTasks().

7.33.3.8 sendAllParamsFromList()

```
static void sendAllParamsFromList (
            daq_param_list_node * list )  [inline], [static]
```

Definition at line 366 of file daq.c.

References daq_param_list_node::next, and sendDaqMsg().

Referenced by daqSubTask().

7.33.3.9 sendDaqMsg()

```
static void sendDaqMsg (
            daq_param_list_node * param )  [inline], [static]
```

Definition at line 347 of file daq.c.

References daq_param_list_node::can_id, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::msg_id, daq_↩param_list_node::param, param_ptrs, daq_param_list_node::size, tmp_daq_msg, and uvSendCanMSG().

Referenced by sendAllParamsFromList().

7.33.3.10 startDaqSubTasks()

```
uv_status startDaqSubTasks ( )
```

Function that starts up all the subtasks.

Definition at line 211 of file daq.c.

References curr_daq_settings, daq_loop_args::daq_child_priority, daq_tlist, daqSubTask(), daq_child_task::meta↩
_task_handle, daq_child_task::next_task, and UV_OK.

Referenced by daqMasterTask().

7.33.3.11 stopDaqSubTasks()

```
uv_status stopDaqSubTasks ( )
```

Function that shuts down the subtasks.

Definition at line 228 of file daq.c.

References daq_tlist, daq_child_task::meta_task_handle, and UV_ERROR.

Referenced by daqMasterTask().

7.33.4 Variable Documentation

7.33.4.1 curr_daq_settings

```
daq_loop_args* curr_daq_settings = NULL
```

Definition at line 28 of file daq.c.

Referenced by configureDaqSubTasks(), initDaqTask(), and startDaqSubTasks().

7.33.4.2 daq_tlist

```
daq_child_task* daq_tlist = NULL  [static]
```

Definition at line 88 of file daq.c.

Referenced by insertParamToRegister(), startDaqSubTasks(), and stopDaqSubTasks().

7.33.4.3 datapoints

```
daq_datapoint* datapoints = NULL  [static]
```

Definition at line 90 of file daq.c.

Referenced by initDaqTask().

7.33.4.4 default_daq_settings

```
daq_loop_args default_daq_settings
```

Initial value:

```
= {
    .total_params_logged = 4,
    .throttle_daq_to_preserve_performance = 1,
    .minimum_daq_period = 10,
    .can_channel = 1,
    .daq_child_priority = 1
}
```

Definition at line 30 of file daq.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.33.4.5 default_datapoints

```
daq_datapoint default_datapoints[]
```

Definition at line 38 of file daq.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.33.4.6 param_bank

```
daq_param_list_node* param_bank = NULL  [static]
```

Definition at line 92 of file daq.c.

Referenced by configureDaqSubTasks().

7.33.4.7 param_ptrs

```
void* param_ptrs[MAX_LOGGABLE_PARAMS]  [static]
```

Definition at line 86 of file daq.c.

Referenced by associateDaqParamWithVar(), and sendDaqMsg().

7.33.4.8 tmp_daq_msg

```
uv_CAN_msg tmp_daq_msg
```

Definition at line 342 of file daq.c.

Referenced by sendDaqMsg().

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 <stdlib.h>
#include <stdio.h>
#include "driving_loop.h"
```

Macros

- #define ACCELERATION 0
- #define AUTOCROSS 1
- #define ENDURANCE 2

Functions

- float calculateThrottlePercentage (uint16_t apps1, uint16_t apps2)
- bool performSafetyChecks (driving_loop_args *dl_params, uint16_t apps1_value, uint16_t apps2_value, uint16_t bps1_value, uint16_t bps2_value, enum DL_internal_state *dl_status)

  Converts APPS sensor readings into a throttle percentage.
- enum uv_status_t initDrivingLoop (void *argument)
- static float mapThrottleToTorque (float throttle_percent)
- float getKValue (int raceMode)
- static float applyTorqueFilter (float T_req, float T_prev, bool is_accelearting)

  Applies filtering to smooth torque transitions.
- static void sendTorqueToMotorController (float T_filtered)

  Sends the filtered torque value to the motor controller.
- 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
- driving_loop_args default_dl_settings
- driving_loop_args * driving_args = NULL
- bool is_accelerating = false
- float T_PREV = 0
- float T_REQ = 0

7.36.1 Detailed Description

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

7.36.2 Macro Definition Documentation

7.36.2.1 ACCELERATION

```
#define ACCELERATION 0
```

Definition at line 100 of file driving_loop.c.

7.36.2.2 AUTOCROSS

```
#define AUTOCROSS 1
```

Definition at line 101 of file driving_loop.c.

7.36.2.3 ENDURANCE

```
#define ENDURANCE 2
```

Definition at line 102 of file driving_loop.c.

7.36.3 Function Documentation

7.36.3.1 applyTorqueFilter()

```
static float applyTorqueFilter (
            float T_req,
            float T_prev,
            bool is_accelearting )  [inline], [static]
```

Applies filtering to smooth torque transitions.

Rachan

Parameters

T_req	Requested torque before filtering.
T_prev	Previous filtered torque value.

Returns

Smoothed torque value.

Definition at line 149 of file driving_loop.c.

References getKValue(), and is_accelerating.

7.36.3.2 calculateThrottlePercentage()

```
float calculateThrottlePercentage (
            uint16_t apps1,
            uint16_t apps2 )
```

Definition at line 117 of file driving_loop.c.

References driving_args, driving_loop_args::max_apps1_value, driving_loop_args::max_apps2_value, and driving_loop_args::min_apps1_value.

7.36.3.3 getKValue()

```
float getKValue (
            int raceMode )  [inline]
```

Definition at line 105 of file driving_loop.c.

References ACCELERATION, AUTOCROSS, and ENDURANCE.

Referenced by applyTorqueFilter().

7.36.3.4 initDrivingLoop()

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

Definition at line 42 of file driving_loop.c.

References uv_task_info::active_states, adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, APPS1_ADC_↩ VAL, APPS2_ADC_VAL, associateDaqParamWithVar(), BPS1_ADC_VAL, BPS2_ADC_VAL, current_vehicle_↩ settings, uv_task_info::deletion_states, driving_args, uv_vehicle_settings::driving_loop_settings, PROGRAMMING, uv_task_info::stack_size, StartDrivingLoop(), uv_task_info::suspension_states, uv_task_info::task_args, uv_task↩ _info::task_function, uv_task_info::task_name, uv_task_info::task_period, uv_task_info::task_priority, UV_DRIVI↩ NG, UV_ERROR, UV_ERROR_STATE, UV_INIT, UV_LAUNCH_CONTROL, UV_OK, UV_READY, UV_SUSPE↩ NDED, and uvCreateTask().

Referenced by uvInitStateEngine().

7.36.3.5 mapThrottleToTorque()

```
static float mapThrottleToTorque (
            float throttle_percent )  [inline], [static]
```

Definition at line 87 of file driving_loop.c.

References driving_loop_args::absolute_max_motor_torque, and driving_args.

7.36.3.6 performSafetyChecks()

```
bool performSafetyChecks (
            driving_loop_args * dl_params,
            uint16_t apps1_value,
            uint16_t apps2_value,
            uint16_t bps1_value,
            uint16_t bps2_value,
            enum DL_internal_state * dl_status )
```

Converts APPS sensor readings into a throttle percentage.

rachan

Parameters

apps1	Raw voltage value from APPS1 sensor
apps2	Raw voltage value from APPS2 sensor

TO DO: Set RPM_MAX & RPM_MIN

Returns

Throttle percentage (0% to 100%). AMMAR

Maps Throttle Percentage to Torque Request. Call the calculateThrottlePercentage function to get the value

Parameters

throttle_percent	Throttle percentage (0% - 100%).

Returns

Requested torque in Nm.

Definition at line 452 of file driving_loop.c.

Referenced by StartDrivingLoop().

7.36.3.7 sendTorqueToMotorController()

```
static void sendTorqueToMotorController (
            float T_filtered )  [static]
```

Sends the filtered torque value to the motor controller.

Rachan

Parameters

T_filtered	Final torque value after filtering.

Definition at line 182 of file driving_loop.c.

References MotorControllerSpinTest().

7.36.3.8 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 rachan: ensures the function runs exactly 100ms, regardless of execution time.
```
 */
TickType_t tick_period = pdMS_TO_TICKS(params->task_period); //Convert ms of period to the RTOS ticks
TickType_t last_time = xTaskGetTickCount();
/**
```

Performs safety checks on APPS (Throttle) and BPS (Brake) sensors.

This function ensures that:

- Throttle position sensors (APPS1 & APPS2) are within 10% of each other.

- Brake pressure sensors (BPS1 & BPS2) are within 5% of each other.
- Sensors are within their expected min/max ranges.
- Brake and throttle are not pressed at the same time.

If a **fatal error** is detected (e.g., sensor out of range), the function:

- **Stops the motor**.
- **Kills the task execution** (`killSelf()`).

If a **non-fatal error** occurs (e.g., sensor mismatch exceeding the limit):

- **Stops the motor**.
- **Suspends the task temporarily** (`suspendSelf()`).

If safety conditions return to normal, the function:

- **Restarts the motor**.

Parameters

dl_params	Pointer to the driving loop parameters.
apps1_value	Raw sensor reading from APPS1.
apps2_value	Raw sensor reading from APPS2.
bps1_value	Raw sensor reading from BPS1.
bps2_value	Raw sensor reading from BPS2.
params	Pointer to the current task information.
dl_status	Pointer to the driving loop internal state.

Return values

true	All safety checks passed.
false	One or more safety checks failed.

Definition at line 194 of file driving_loop.c.

References adc1_APPS1, adc1_APPS2, adc1_BPS1, adc1_BPS2, uv_task_info::cmd_data, is_accelerating, killSelf(), performSafetyChecks(), Plausible, suspendSelf(), T_PREV, T_REQ, uv_task_info::task_args, uv_task_info::task_period, UV_KILL_CMD, and UV_SUSPEND_CMD.

Referenced by initDrivingLoop().

7.36.4 Variable Documentation

7.36.4.1 adc1_APPS1

```
uint16_t adc1_APPS1
```

Definition at line 64 of file main.c.

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

7.36.4.2 adc1_APPS2

```
uint16_t adc1_APPS2
```

Definition at line 65 of file main.c.

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

7.36.4.3 adc1_BPS1

```
uint16_t adc1_BPS1
```

Definition at line 66 of file main.c.

Referenced by initDrivingLoop(), and StartDrivingLoop().

7.36.4.4 adc1_BPS2

```
uint16_t adc1_BPS2
```

Definition at line 67 of file main.c.

Referenced by initDrivingLoop(), and StartDrivingLoop().

7.36.4.5 default_dl_settings

```
driving_loop_args default_dl_settings
```

Definition at line 27 of file driving_loop.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.36.4.6 driving_args

```
driving_loop_args* driving_args = NULL
```

Definition at line 29 of file driving_loop.c.

Referenced by calculateThrottlePercentage(), initDrivingLoop(), and mapThrottleToTorque().

7.36.4.7 is_accelerating

```
bool is_accelerating = false
```

Definition at line 34 of file driving_loop.c.

Referenced by applyTorqueFilter(), and StartDrivingLoop().

7.36.4.8 T_PREV

```
float T_PREV = 0
```

Definition at line 35 of file driving_loop.c.

Referenced by StartDrivingLoop().

7.36.4.9 T_REQ

```
float T_REQ = 0
```

Definition at line 36 of file driving_loop.c.

Referenced by StartDrivingLoop().

7.37 Core/Src/freertos.c File Reference

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

Functions

- void StartDefaultTask (void const *argument)

  Function implementing the defaultTask thread.
- void MX_FREERTOS_Init (void)

  FreeRTOS initialization.
- void vApplicationGetIdleTaskMemory (StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize)
- void vApplicationGetTimerTaskMemory (StaticTask_t **ppxTimerTaskTCBBuffer, StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize)
- void vApplicationTickHook (void)
- void vApplicationStackOverflowHook (TaskHandle_t xTask, signed char *pcTaskName)
- void vApplicationMallocFailedHook (void)
- void vApplicationIdleHook (void)

Variables

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

7.37.1 Function Documentation

7.37.1.1 MX_FREERTOS_Init()

```
void MX_FREERTOS_Init (
            void  )
```

FreeRTOS initialization.

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

```
osThreadId defaultTaskHandle
```

Definition at line 53 of file freertos.c.

Referenced by MX_FREERTOS_Init().

7.37.2.2 init_settings

```
uv_init_struct init_settings
```

File Name : freertos.c Description : Code for freertos applications

Attention

Copyright (c) 2024 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition at line 48 of file freertos.c.

Referenced by MX_FREERTOS_Init().

7.37.2.3 init_task_handle

```
TaskHandle_t init_task_handle
```

Definition at line 51 of file freertos.c.

Referenced by MX_FREERTOS_Init(), and uvInit().

7.37.2.4 xIdleStack

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

Definition at line 130 of file freertos.c.

Referenced by vApplicationGetIdleTaskMemory().

7.37.2.5 xIdleTaskTCBBuffer

```
StaticTask_t xIdleTaskTCBBuffer  [static]
```

Definition at line 129 of file freertos.c.

Referenced by vApplicationGetIdleTaskMemory().

7.37.2.6 xTimerStack

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

Definition at line 144 of file freertos.c.

Referenced by vApplicationGetTimerTaskMemory().

7.37.2.7 xTimerTaskTCBBuffer

```
StaticTask_t xTimerTaskTCBBuffer  [static]
```

Definition at line 143 of file freertos.c.

Referenced by vApplicationGetTimerTaskMemory().

7.38 Core/Src/gpio.c File Reference

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

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

Functions

- void MX_GPIO_Init (void)

7.38.1 Detailed Description

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

Attention

Copyright (c) 2023 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

7.38.2 Function Documentation

7.38.2.1 MX_GPIO_Init()

```
void MX_GPIO_Init (
             void  )
```

Configure pins as Analog Input Output EVENT_OUT EXTI

Definition at line 42 of file gpio.c.

References Blue_LED_Pin, Orange_LED_Pin, Red_LED_Pin, Start_Button_Input_GPIO_Port, and Start_Button↩
_Input_Pin.

Referenced by main().

7.39 Core/Src/imd.c File Reference

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

Functions

- void IMD_Parse_Message (int DLC, uint8_t Data[ ])
- void IMD_Request_Status (uint8_t Status)
- void IMD_Check_Status_Bits (uint8_t Data)
- void IMD_Check_Error_Flags (uint8_t Data[ ])
- void IMD_Check_Isolation_State (uint8_t Data[ ])
- void IMD_Check_Isolation_Resistances (uint8_t Data[ ])
- void IMD_Check_Isolation_Capacitances (uint8_t Data[ ])
- void IMD_Check_Voltages_Vp_and_Vn (uint8_t Data[ ])
- void IMD_Check_Battery_Voltage (uint8_t Data[ ])
- void IMD_Check_Temperature (uint8_t Data[ ])
- void IMD_Check_Safety_Touch_Energy (uint8_t Data[ ])
- void IMD_Check_Safety_Touch_Current (uint8_t Data[ ])
- void IMD_Check_Max_Battery_Working_Voltage (uint8_t Data[ ])
- void IMD_Check_Part_Name (uint8_t Data[ ])
- void IMD_Check_Version (uint8_t Data[ ])
- void IMD_Check_Serial_Number (uint8_t Data[ ])
- void IMD_Check_Uptime (uint8_t Data[ ])
- void IMD_Startup ()
- void initIMD (void ∗args)

Variables

- uint8_t IMD_status_bits = 0
- uint8_t IMD_High_Uncertainty = 0
- uint32_t IMD_Read_Part_Name [4]
- const uint32_t IMD_Expected_Part_Name [4]
- uint8_t IMD_Part_Name_0_Set = 0
- uint8_t IMD_Part_Name_1_Set = 0
- uint8_t IMD_Part_Name_2_Set = 0
- uint8_t IMD_Part_Name_3_Set = 0
- uint8_t IMD_Part_Name_Set = 0
- uint32_t IMD_Read_Version [3]
- const uint32_t IMD_Expected_Version [3]
- uint8_t IMD_Version_0_Set = 0
- uint8_t IMD_Version_1_Set = 0
- uint8_t IMD_Version_2_Set = 0
- uint8_t IMD_Version_Set = 0
- uint32_t IMD_Read_Serial_Number [4]
- const uint32_t IMD_Expected_Serial_Number [4]
- uint8_t IMD_Serial_Number_0_Set = 0
- uint8_t IMD_Serial_Number_1_Set = 0
- uint8_t IMD_Serial_Number_2_Set = 0
- uint8_t IMD_Serial_Number_3_Set = 0
- uint8_t IMD_Serial_Number_Set = 0
- int32_t IMD_Temperature
- uint8_t IMD_error_flags_requested = 0
- uv_imd_settings default_imd_settings

7.39.1 Function Documentation

7.39.1.1 IMD_Check_Battery_Voltage()

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

Definition at line 356 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.2 IMD_Check_Error_Flags()

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

Definition at line 262 of file imd.c.

References Err_CH, Err_clock, Err_temp, Err_Vexi, Err_Vpwr, Err_Vx1, Err_Vx2, Err_VxR, and Err_Watchdog.

Referenced by IMD_Parse_Message().

7.39.1.3 IMD_Check_Isolation_Capacitances()

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

Definition at line 342 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.4 IMD_Check_Isolation_Resistances()

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

Definition at line 317 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.39.1.5 IMD_Check_Isolation_State()

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

Definition at line 301 of file imd.c.

References IMD_High_Uncertainty.

Referenced by IMD_Parse_Message().

7.39.1.6 IMD_Check_Max_Battery_Working_Voltage()

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

Definition at line 393 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.7 IMD_Check_Part_Name()

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

Definition at line 406 of file imd.c.

References IMD_Expected_Part_Name, IMD_Part_Name_0_Set, IMD_Part_Name_1_Set, IMD_Part_Name_2_↩ Set, IMD_Part_Name_3_Set, IMD_Part_Name_Set, IMD_Read_Part_Name, Part_name_0, Part_name_1, Part↩ _name_2, and Part_name_3.

Referenced by IMD_Parse_Message().

7.39.1.8 IMD_Check_Safety_Touch_Current()

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

Definition at line 381 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.9 IMD_Check_Safety_Touch_Energy()

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

Definition at line 374 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.10 IMD_Check_Serial_Number()

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

Definition at line 488 of file imd.c.

References IMD_Expected_Serial_Number, IMD_Read_Serial_Number, IMD_Serial_Number_0_Set, IMD↩ _Serial_Number_1_Set, IMD_Serial_Number_2_Set, IMD_Serial_Number_3_Set, IMD_Serial_Number_Set, Serial_number_0, Serial_number_1, Serial_number_2, and Serial_number_3.

Referenced by IMD_Parse_Message().

7.39.1.11 IMD_Check_Status_Bits()

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

Definition at line 218 of file imd.c.

References Error_flags, Hardware_Error, High_Battery_Voltage, High_Uncertainty, IMD_error_flags_requested, IMD_High_Uncertainty, IMD_Request_Status(), Isolation_status_bit0, Isolation_status_bit1, and Low_Battery_Voltage.

Referenced by IMD_Parse_Message().

7.39.1.12 IMD_Check_Temperature()

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

Definition at line 363 of file imd.c.

References IMD_Temperature.

Referenced by IMD_Parse_Message().

7.39.1.13 IMD_Check_Uptime()

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

Definition at line 529 of file imd.c.

7.39.1.14 IMD_Check_Version()

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

Definition at line 448 of file imd.c.

References IMD_Expected_Version, IMD_Read_Version, IMD_Version_0_Set, IMD_Version_1_Set, IMD_Version_2_Set, IMD_Version_Set, Version_0, Version_1, and Version_2.

Referenced by IMD_Parse_Message().

7.39.1.15 IMD_Check_Voltages_Vp_and_Vn()

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

Definition at line 349 of file imd.c.

Referenced by IMD_Parse_Message().

7.39.1.16 IMD_Parse_Message()

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

Definition at line 73 of file imd.c.

References battery_voltage, Error_flags, Error_Handler(), IMD_Check_Battery_Voltage(), IMD_Check_Error_Flags(), IMD_Check_Isolation_Capacitances(), IMD_Check_Isolation_Resistances(), IMD_Check_Isolation_State(), IMD_Check_Max_Battery_Working_Voltage(), IMD_Check_Part_Name(), IMD_Check_Safety_Touch_Current(), IMD_Check_Safety_Touch_Energy(), IMD_Check_Serial_Number(), IMD_Check_Status_Bits(), IMD_Check_Temperature(), IMD_Check_Version(), IMD_Check_Voltages_Vp_and_Vn(), isolation_capacitances, isolation_resistances, isolation_state, Max_battery_working_voltage, Part_name_0, Part_name_1, Part_name_2, Part_name_3, safety_touch_current, safety_touch_energy, Serial_number_0, Serial_number_1, Serial_number_2, Serial_number_3, Temperature, Uptime_counter, Vb_hi_res, Version_0, Version_1, Version_2, Vexc_hi_res, Vn_hi_res, voltages_Vp_and_Vn, Vp_hi_res, and Vpwr_hi_res.

7.39.1.17 IMD_Request_Status()

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

Definition at line 185 of file imd.c.

References Error_Handler(), hcan2, IMD_CAN_ID_Tx, TxData, TxHeader, and TxMailbox.

Referenced by IMD_Check_Status_Bits(), and IMD_Startup().

7.39.1.18 IMD_Startup()

```
void IMD_Startup ( )
```

Definition at line 533 of file imd.c.

References IMD_Request_Status(), isolation_state, Max_battery_working_voltage, Part_name_0, Part_name_1, Part_name_2, Part_name_3, Serial_number_0, Serial_number_1, Serial_number_2, Serial_number_3, Version_0, Version_1, and Version_2.

7.39.1.19 initIMD()

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

Definition at line 559 of file imd.c.

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

Referenced by uvInit().

7.39.2 Variable Documentation

7.39.2.1 default_imd_settings

```
uv_imd_settings default_imd_settings
```

Initial value:

```
= {
    .min_isolation_resistances = 10000,
    .expected_isolation_capacitances = 100,
    .max_imd_temperature = 60
}
```

Definition at line 65 of file imd.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.39.2.2 IMD_error_flags_requested

```
uint8_t IMD_error_flags_requested = 0
```

Definition at line 62 of file imd.c.

Referenced by IMD_Check_Status_Bits().

7.39.2.3 IMD_Expected_Part_Name

```
const uint32_t IMD_Expected_Part_Name[4]
```

Definition at line 26 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.4 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().

7.39.2.5 IMD_Expected_Version

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

Definition at line 36 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.6 IMD_High_Uncertainty

```
uint8_t IMD_High_Uncertainty = 0
```

Definition at line 20 of file imd.c.

Referenced by IMD_Check_Isolation_Resistances(), IMD_Check_Isolation_State(), and IMD_Check_Status_Bits().

7.39.2.7 IMD_Part_Name_0_Set

```
uint8_t IMD_Part_Name_0_Set = 0
```

Definition at line 28 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.8 IMD_Part_Name_1_Set

```
uint8_t IMD_Part_Name_1_Set = 0
```

Definition at line 29 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.9 IMD_Part_Name_2_Set

```
uint8_t IMD_Part_Name_2_Set = 0
```

Definition at line 30 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.10 IMD_Part_Name_3_Set

```
uint8_t IMD_Part_Name_3_Set = 0
```

Definition at line 31 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.11 IMD_Part_Name_Set

```
uint8_t IMD_Part_Name_Set = 0
```

Definition at line 32 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.12 IMD_Read_Part_Name

```
uint32_t IMD_Read_Part_Name[4]
```

Definition at line 25 of file imd.c.

Referenced by IMD_Check_Part_Name().

7.39.2.13 IMD_Read_Serial_Number

```
uint32_t IMD_Read_Serial_Number[4]
```

Definition at line 45 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.14 IMD_Read_Version

```
uint32_t IMD_Read_Version[3]
```

Definition at line 35 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.15 IMD_Serial_Number_0_Set

```
uint8_t IMD_Serial_Number_0_Set = 0
```

Definition at line 50 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.16 IMD_Serial_Number_1_Set

```
uint8_t IMD_Serial_Number_1_Set = 0
```

Definition at line 51 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.17 IMD_Serial_Number_2_Set

```
uint8_t IMD_Serial_Number_2_Set = 0
```

Definition at line 52 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.18 IMD_Serial_Number_3_Set

```
uint8_t IMD_Serial_Number_3_Set = 0
```

Definition at line 53 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.19 IMD_Serial_Number_Set

```
uint8_t IMD_Serial_Number_Set = 0
```

Definition at line 54 of file imd.c.

Referenced by IMD_Check_Serial_Number().

7.39.2.20 IMD_status_bits

```
uint8_t IMD_status_bits = 0
```

Definition at line 19 of file imd.c.

7.39.2.21 IMD_Temperature

```
int32_t IMD_Temperature
```

Definition at line 57 of file imd.c.

Referenced by IMD_Check_Temperature().

7.39.2.22 IMD_Version_0_Set

```
uint8_t IMD_Version_0_Set = 0
```

Definition at line 38 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.23 IMD_Version_1_Set

```
uint8_t IMD_Version_1_Set = 0
```

Definition at line 39 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.24 IMD_Version_2_Set

```
uint8_t IMD_Version_2_Set = 0
```

Definition at line 40 of file imd.c.

Referenced by IMD_Check_Version().

7.39.2.25 IMD_Version_Set

```
uint8_t IMD_Version_Set = 0
```

Definition at line 41 of file imd.c.

Referenced by IMD_Check_Version().

7.40 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 [4]
- 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 384 of file main.c.

Referenced by HAL_ADC_MspInit(), HAL_CAN_RxFifo0MsgPendingCallback(), IMD_Parse_Message(), IMD_Request_Status(), initADCTask(), 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 processADCBuffer().

7.40.3.3 HAL_ADC_LevelOutOfWindowCallback()

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

Definition at line 336 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 327 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 360 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_LevelOutOfWindowCallback(), initDrivingLoop(), main(), processADCBuffer(), and StartDrivingLoop().

7.40.4.2 adc1_APPS2

```
uint16_t adc1_APPS2
```

Definition at line 65 of file main.c.

Referenced by HAL_ADC_LevelOutOfWindowCallback(), initDrivingLoop(), main(), processADCBuffer(), and StartDrivingLoop().

7.40.4.3 adc1_BPS1

```
uint16_t adc1_BPS1
```

Definition at line 66 of file main.c.

Referenced by initDrivingLoop(), processADCBuffer(), and StartDrivingLoop().

7.40.4.4 adc1_BPS2

```
uint16_t adc1_BPS2
```

Definition at line 67 of file main.c.

Referenced by initDrivingLoop(), processADCBuffer(), and StartDrivingLoop().

7.40.4.5 adc2_CoolantFlow

```
uint16_t adc2_CoolantFlow
```

Definition at line 71 of file main.c.

7.40.4.6 adc2_CoolantTemp

```
uint16_t adc2_CoolantTemp
```

Definition at line 70 of file main.c.

7.40.4.7 adc_buf1

```
volatile uint32_t adc_buf1[4]
```

Definition at line 62 of file main.c.

Referenced by processADCBuffer(), and StartADCTask().

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_TIM_PeriodElapsedCallback().

7.41 Core/Src/motor_controller.c File Reference

```
#include "motor_controller.h"
#include "can.h"
#include "cmsis_os.h"
#include "uvfr_utils.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

Functions

- uint16_t MotorControllerSpinTest (float T_filtered)

  Sends a direct torque command to the motor controller.
- void MC_Request_Data (uint8_t RegID)

  Sends a CAN request to retrieve a specific register from the motor controller.
- uv_status MC_Set_Param (uint8_t RegID, uint16_t d)
- void Parse_Bamocar_Response (uv_CAN_msg *msg)

  Parses a 32-bit value from a CAN message in little-endian format.
- static void MotorControllerErrorHandler_16bitLE (uint8_t *data, uint8_t length)

  Helper function to process a 16-bit error/warning field (little-endian).
- void ProcessMotorControllerResponse (uv_CAN_msg *msg)

  Processes a motor controller response received via CAN.
- void MC_Startup (void *args)

  Initializes the motor controller.

Variables

- motor_controller_settings mc_default_settings

7.41.1 Function Documentation

7.41.1.1 MC_Request_Data()

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

Sends a CAN request to retrieve a specific register from the motor controller.

The request message is formatted as: [0x3D, RegID, 0], which should trigger an immediate reply.

Definition at line 63 of file motor_controller.c.

References motor_controller_settings::can_id_tx, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, mc_default_settings, uv_CAN_msg::msg_id, UV_OK, and uvSendCanMSG().

Referenced by MC_Startup().

7.41.1.2 MC_Set_Param()

```
uv_status MC_Set_Param (
            uint8_t RegID,
            uint16_t d )
```

Definition at line 80 of file motor_controller.c.

References motor_controller_settings::can_id_tx, uv_CAN_msg::data, uv_CAN_msg::dlc, mc_default_settings, uv_CAN_msg::msg_id, UV_ERROR, UV_OK, and uvSendCanMSG().

Referenced by MC_Startup().

7.41.1.3 MC_Startup()

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

Initializes the motor controller.

This routine performs the following steps:

1. Requests the serial number and firmware version.
2. Sends a nominal torque command (spin test).
3. Requests error/warning data.
4. Suspends itself after successful initialization.

Definition at line 278 of file motor_controller.c.

References uv_init_task_response::device, FIRMWARE_VERSION_REGISTER, uv_init_task_args::init_info↩_queue, insertCANMessageHandler(), mc_default_settings, MC_Request_Data(), MC_Set_Param(), MOTO↩R_CONTROLLER, motor_controller_errors_warnings, MotorControllerSpinTest(), Parse_Bamocar_Response(), motor_controller_settings::proportional_gain, SERIAL_NUMBER_REGISTER, uv_init_task_response::status, and UV_OK.

Referenced by uvInit().

7.41.1.4 MotorControllerErrorHandler_16bitLE()

```
static void MotorControllerErrorHandler_16bitLE (
            uint8_t * data,
            uint8_t length )  [static]
```

Helper function to process a 16-bit error/warning field (little-endian).

It expects the error data in 2 bytes where: data[0] = LSB, data[1] = MSB. It checks the error flags and calls uvPanic for critical errors.

Definition at line 146 of file motor_controller.c.

References AC_current_offset_fault, bleed_resistor_overload, CAN_timeout_error, critical_AC_current, ecode↩ _timeout_error, eprom_read_error, feedback_signal_error, hardware_fault, IGBT_temp_max_limit, internal_↩ hardware_voltage_problem, mains_voltage_max_limit, mains_voltage_min_limit, motor_temp_max_limit, race_↩ away_detected, rotate_field_enable_not_present_run, and watchdog_reset.

Referenced by ProcessMotorControllerResponse().

7.41.1.5 MotorControllerSpinTest()

```
uint16_t MotorControllerSpinTest (
            float T_filtered )
```

Sends a direct torque command to the motor controller.

This function accepts a float for the desired torque (T_filtered), clamps it between 0 and 100, converts it to a 16-bit integer, and then sends it over CAN using uvSendCanMSG.

Definition at line 30 of file motor_controller.c.

References motor_controller_settings::can_id_tx, uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, mc_default_settings, uv_CAN_msg::msg_id, N_set, UV_OK, and uvSendCanMSG().

Referenced by MC_Startup(), and sendTorqueToMotorController().

7.41.1.6 Parse_Bamocar_Response()

```
void Parse_Bamocar_Response (
            uv_CAN_msg * msg )
```

Parses a 32-bit value from a CAN message in little-endian format.

This example assumes that the data bytes are stored as: data[0] = LSB, data[3] = MSB.

Definition at line 104 of file motor_controller.c.

References uv_CAN_msg::data, deserializeSmallE16, uv_CAN_msg::dlc, mc_default_settings, and motor_↩ controller_settings::proportional_gain.

Referenced by MC_Startup().

7.41.1.7 ProcessMotorControllerResponse()

```
void ProcessMotorControllerResponse (
            uv_CAN_msg * msg )
```

Processes a motor controller response received via CAN.

This function examines the first byte as the register ID and then processes the rest of the message using little-endian parsing. For error/warning responses (for example, when reg_id equals motor_controller_errors_warnings), it calls the error handler.

Definition at line 211 of file motor_controller.c.

References CURRENT_ACTUAL, uv_CAN_msg::data, uv_CAN_msg::dlc, LOGIMAP_ERRORS, LOGIMAP_IO, motor_controller_errors_warnings, MotorControllerErrorHandler_16bitLE(), N_actual, and POS_ACTUAL.

7.41.2 Variable Documentation

7.41.2.1 mc_default_settings

```
motor_controller_settings mc_default_settings
```

Initial value:

```
= {
    .can_id_tx              = 0x201,
    .can_id_rx              = 0x181,
    .mc_CAN_timeout         = 2,
    .proportional_gain      = 10,
    .integral_time_constant = 400,
    .integral_memory_max    = 60
}
```

Definition at line 14 of file motor_controller.c.

Referenced by MC_Request_Data(), MC_Set_Param(), MC_Startup(), MotorControllerSpinTest(), Parse_↩ Bamocar_Response(), setupDefaultSettings(), and uvResetFlashToDefault().

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 "uvfr_utils.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)

Variables

- uv19_pdu_settings default_pdu_settings

7.44.1 Function Documentation

7.44.1.1 initPDU()

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

Definition at line 189 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 54 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 176 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 142 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 80 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 106 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 40 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 164 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 121 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 68 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 93 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 17 of file pdu.c.

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

7.44.2 Variable Documentation

7.44.2.1 default_pdu_settings

```
uv19_pdu_settings default_pdu_settings
```

Initial value:

```
= {
    .PDU_rx_addr = 0x310,
    .PDU_tx_addr = 0x311,
    .sdc_channel = 0x0C
}
```

Definition at line 10 of file pdu.c.

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

  Function that inserts data into the tree, and creates a new node.
- void * rbDelete (rbtree *rbt, rbnode *node, int keep)

  Deletes a node from the tree.
- int rbCheckOrder (rbtree *rbt, void *min, void *max)

  Function that validates that the order of the nodes in the tree is correct.
- int rbCheckBlackHeight (rbtree *rbt)

  Function that Checks the height of black nodes.
- void rbPrint (rbtree *rbt, void(*print_func)(void *))

  Function used to print the contents of the tree.

7.45.1 Function Documentation

7.45.1.1 checkBlackHeight()

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

Definition at line 582 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 553 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 454 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 649 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 293 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 617 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 rbApplyNode()

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

Definition at line 116 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 )
```

Function that Checks the height of black nodes.

Attention

DANGER!! THIS FUNCTION IS RECURSIVE. This is intended to be used in a laptop debugging context. The VCU simply does not have enough memory to deal with recursion of this manner.

Deprecated Leftovers from laptop unit tests.

Definition at line 571 of file rb_tree.c.

References 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 )
```

Function that validates that the order of the nodes in the tree is correct.

Attention

DANGER!! THIS FUNCTION IS RECURSIVE. This is intended to be used in a laptop debugging context. The VCU simply does not have enough memory to deal with recursion of this manner.

Deprecated Leftovers from laptop unit tests.

Definition at line 545 of file rb_tree.c.

References 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.

Parameters

compare_func	A function that compares the data of two nodes. Accepts pointers to the data as parameters
destroy_func	The destructor function for the data, for safe disposal of dynamically allocated data

Definition at line 28 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::compare, rbtree::count, rbnode::data, rbtree::destroy, rbnode::left, rbtree::min, rbtree::nil, rbnode::parent, RB_NIL, rbnode::right, and rbtree::root.

7.45.1.11 rbDelete()

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

Deletes a node from the tree.

Parameters

rbt	Instance of a `rbtree` that we are removing the node from.
node	Pointer to the node that we would like to remove
keep	If `keep` is a truthy value, a pointer to the data of the node will be returned. Otherwise, the node and it's data will be destroyed.

Return values

If	`keep` is "true", this will return a pointer to the data held by the deleted node. Otherwise it will return NULL.

Definition at line 360 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::count, rbnode::data, 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 61 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.

Parameters

rbt	Pointer to the tree in which we would like to find the node.

Return values

Returns	a pointer to the node if the node is present in the tree. Otherwise, it will return NULL to indicate the node could not be found.

Definition at line 71 of file rb_tree.c.

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

7.45.1.14 rbInsert()

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

Function that inserts data into the tree, and creates a new node.

Parameters

rbt	Instance of a `rbtree` that we would like to insert data into
data	Pointer to the data we wish to insert

Return values

This	function returns a pointer to the `rbnode` that was added. The function will return NULL if the system is out of memory, or is otherwise unable to insert the node.

Definition at line 207 of file rb_tree.c.

References BLACK, rbnode::color, rbtree::compare, rbtree::count, rbnode::data, rbtree::destroy, 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 )
```

Function used to print the contents of the tree.

Parameters

rbt	a pointer to the `rbtree` you wish to print.
print_func	A pointer to a print function specific to the data.

Attention

DANGER!! RECURSION!!

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

Definition at line 607 of file rb_tree.c.

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

7.45.1.16 rbSuccessor()

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

Definition at line 92 of file rb_tree.c.

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

Referenced by rbDelete().

7.45.1.17 rotateLeft()

```
void rotateLeft (
            rbtree * rbt,
            rbnode * x )  [static]
```

Definition at line 153 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 179 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 ADC_IRQHandler (void)

  This function handles ADC1, ADC2 and ADC3 global interrupts.
- 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
- ADC_HandleTypeDef hadc1
- ADC_HandleTypeDef hadc2
- 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 ADC_IRQHandler()

```
void ADC_IRQHandler (
             void  )
```

This function handles ADC1, ADC2 and ADC3 global interrupts.

Definition at line 183 of file stm32f4xx_it.c.

References hadc1, and hadc2.

7.49.2.2 BusFault_Handler()

```
void BusFault_Handler (
             void  )
```

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

Definition at line 119 of file stm32f4xx_it.c.

7.49.2.3 CAN2_RX0_IRQHandler()

```
void CAN2_RX0_IRQHandler (
             void  )
```

This function handles CAN2 RX0 interrupts.

Definition at line 240 of file stm32f4xx_it.c.

References hcan2.

7.49.2.4 CAN2_RX1_IRQHandler()

```
void CAN2_RX1_IRQHandler (
             void  )
```

This function handles CAN2 RX1 interrupt.

Definition at line 254 of file stm32f4xx_it.c.

References hcan2.

7.49.2.5 CAN2_TX_IRQHandler()

```
void CAN2_TX_IRQHandler (
             void  )
```

This function handles CAN2 TX interrupts.

Definition at line 226 of file stm32f4xx_it.c.

References hcan2.

7.49.2.6 DebugMon_Handler()

```
void DebugMon_Handler (
             void  )
```

This function handles Debug monitor.

Definition at line 149 of file stm32f4xx_it.c.

7.49.2.7 DMA2_Stream0_IRQHandler()

```
void DMA2_Stream0_IRQHandler (
             void  )
```

This function handles DMA2 stream0 global interrupt.

Definition at line 212 of file stm32f4xx_it.c.

References hdma_adc1.

7.49.2.8 EXTI0_IRQHandler()

```
void EXTI0_IRQHandler (
             void  )
```

This function handles EXTI line0 interrupt.

Definition at line 169 of file stm32f4xx_it.c.

References Start_Button_Input_Pin.

7.49.2.9 HardFault_Handler()

```
void HardFault_Handler (
             void  )
```

This function handles Hard fault interrupt.

Definition at line 89 of file stm32f4xx_it.c.

7.49.2.10 MemManage_Handler()

```
void MemManage_Handler (
             void  )
```

This function handles Memory management fault.

Definition at line 104 of file stm32f4xx_it.c.

7.49.2.11 NMI_Handler()

```
void NMI_Handler (
             void  )
```

This function handles Non maskable interrupt.

Definition at line 74 of file stm32f4xx_it.c.

7.49.2.12 TIM1_UP_TIM10_IRQHandler()

```
void TIM1_UP_TIM10_IRQHandler (
             void  )
```

This function handles TIM1 update interrupt and TIM10 global interrupt.

Definition at line 198 of file stm32f4xx_it.c.

References htim1.

7.49.2.13 UsageFault_Handler()

```
void UsageFault_Handler (
             void  )
```

This function handles Undefined instruction or illegal state.

Definition at line 134 of file stm32f4xx_it.c.

7.49.3 Variable Documentation

7.49.3.1 hadc1

```
ADC_HandleTypeDef hadc1
```

Definition at line 35 of file adc.c.

Referenced by ADC_IRQHandler(), MX_ADC1_Init(), and StartADCTask().

7.49.3.2 hadc2

```
ADC_HandleTypeDef hadc2
```

Definition at line 36 of file adc.c.

Referenced by ADC_IRQHandler(), and MX_ADC2_Init().

7.49.3.3 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().

7.49.3.4 hdma_adc1

```
DMA_HandleTypeDef hdma_adc1
```

Definition at line 37 of file adc.c.

Referenced by DMA2_Stream0_IRQHandler(), and HAL_ADC_MspInit().

7.49.3.5 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, 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"
```

Data Structures

- struct tx_all_settings_args
- struct tx_journal_args
- union helper_task_args
- struct setting_helper_task

Macros

- #define SRC_UVFR_SETTINGS_C_
- #define VCU_TO_LAPTOP_ID 0x420
- #define LAPTOP_TO_VCU_ID 0x520
- #define FLASH_OK 0
- #define INVALID_SBLOCK 1
- #define FLASH_NOT_UNLOCKED 2
- #define DID_NOT_FINISH_PROGRAMMING 3
- #define DATA_MISMATCH 4
- #define PRE_CHECKSUM 5
- #define POST_CHECKSUM 6

Typedefs

- typedef struct tx_all_settings_args tx_all_settings_args
- typedef struct tx_journal_args tx_journal_args
- typedef union helper_task_args helper_task_args
- typedef struct setting_helper_task setting_helper_task

Functions

- uv_status uvValidateFlashSettings ()

  Function that looks into the flash memory of the VCU, and determines if a valid set of settings exists.
- uv_status uvResetFlashToDefault ()
- void uvSettingsProgrammerTask (void *args) PRIVILEGED_FUNCTION

  This is a task that executes the functionality necessary for vehicle diagnostics.
- static void settingCopy (uint8_t *from, uint8_t *to, uint16_t length) PRIVILEGED_FUNCTION

  Internal function that is used to copy an arbitrary amount of data from point A to point B.
- uv_status uvTransmitVehicleStatus ()

  Transmits the status of the vehicle, including errorcodes and similar.
- uv_status uvAwaitLaptopAckMsg ()
- void handleIncomingLaptopMsg (uv_CAN_msg *msg) PRIVILEGED_FUNCTION

  Callback function for event where CAN message is received from laptop.
- uv_status 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)
- uv_status uvLoadSettingsFromFlash ()

  This loads us some new settings from the flash memory.
- uv_status uvConfigSettingTask (void *args)

  Function to setup the parameters of the setting setter task.
- uv_status uvSettingsInit ()

this function does one thing, and one thing only, it checks if we have custom settings, then it attempts to get them. If it fails, then we revert to factory defaults.

- uv_status uvUpdateTmpSettings (uint8_t *tmp_settings, uint8_t memgroup, uint8_t m_offset, uint8_t size, uint8_t *data)

  Updates a setting in a temporary SBlock by memory group and offset.
- uint32_t uvComputeChunkChecksum (uint32_t *chunk)
- uv_status uvComputeMemRegionChecksum (void *sblock, uint32_t *csums, char mregion)
- uv_status uvComputeSettingsChecksums (void *sblock, uint32_t *csums)
- uv_status uvValidateChecksums (void *sblock)
- void uvSBlockCorruptionHandler ()

  Function to call when the Sblock Flash has become corrupted somehow.
- uv_status uvOverwriteCsr (void *sblock, uint32_t *csr)
- uv_status uvSaveSettingsToFlash (void *sblock, uint32_t *ecode) PRIVILEGED_FUNCTION

  Function to save a finished settings block to flash memory.
- uv_status uvSetSettingResponseReminder (uint32_t reminder)

  This reminds the VCU to shoot off it's own message later when it reboots.
- uv_status uvForceDefaultReversionUponDeviceReset ()
- void * uvCreateTmpSettingsCopy ()

  Function that creates a temporary copy of the existing settings.
- uv_status uvSendSpecificParam (uint8_t *origin, uint8_t mgroup, uint8_t m_offset, uint8_t size)

  Function that locates a specific setting parameter and sends the value over canbus.
- uv_status uvResetFlashToDefault (void *new_sblock)
- uv_status uvSendSettingGroup (uint8_t *origin, uint8_t memgroup, uint8_t m_offset, uint16_t size)

  Helper function for sending all settings.
- void sendAllSettingsWorker (void *args)

  Sub-task that sends all settings from the VCU to the.
- void sendJournalWorker (void *args)

  Sub-task that reads out the journal.

Variables

- PRIVILEGED_DATA uint8_t _s_uvdata
- PRIVILEGED_DATA uint8_t _e_uvdata
- uv_vehicle_settings * current_vehicle_settings = NULL
- struct uv_os_settings default_os_settings
- struct motor_controller_settings mc_default_settings
- struct driving_loop_args default_dl_settings
- struct daq_loop_args default_daq_settings
- struct uv_imd_settings default_imd_settings
- bms_settings_t default_bms_settings
- uv19_pdu_settings default_pdu_settings
- daq_datapoint default_datapoints [ ]
- veh_gen_info default_vehicle
- static bool is_laptop_connected = false
- TickType_t last_contact_with_laptop = 0
- static QueueHandle_t settings_queue = NULL
- static TaskHandle_t ptask_handle = NULL
- static TaskHandle_t child_task_handle = NULL
- static bool child_task_active = false
- const uv_CAN_msg vcu_ack_msg
- const uv_CAN_msg vcu_ack_failed_msg

7.55.1 Macro Definition Documentation

7.55.1.1 DATA_MISMATCH

```
#define DATA_MISMATCH 4
```

Definition at line 567 of file uvfr_settings.c.

7.55.1.2 DID_NOT_FINISH_PROGRAMMING

```
#define DID_NOT_FINISH_PROGRAMMING 3
```

Definition at line 566 of file uvfr_settings.c.

7.55.1.3 FLASH_NOT_UNLOCKED

```
#define FLASH_NOT_UNLOCKED 2
```

Definition at line 565 of file uvfr_settings.c.

7.55.1.4 FLASH_OK

```
#define FLASH_OK 0
```

These defines are for the return error code of the next function

Definition at line 563 of file uvfr_settings.c.

7.55.1.5 INVALID_SBLOCK

```
#define INVALID_SBLOCK 1
```

Definition at line 564 of file uvfr_settings.c.

7.55.1.6 LAPTOP_TO_VCU_ID

```
#define LAPTOP_TO_VCU_ID 0x520
```

Definition at line 14 of file uvfr_settings.c.

7.55.1.7 POST_CHECKSUM

```
#define POST_CHECKSUM 6
```

Definition at line 569 of file uvfr_settings.c.

7.55.1.8 PRE_CHECKSUM

```
#define PRE_CHECKSUM 5
```

Definition at line 568 of file uvfr_settings.c.

7.55.1.9 SRC_UVFR_SETTINGS_C_

```
#define SRC_UVFR_SETTINGS_C_
```

Definition at line 7 of file uvfr_settings.c.

7.55.1.10 VCU_TO_LAPTOP_ID

```
#define VCU_TO_LAPTOP_ID 0x420
```

Definition at line 13 of file uvfr_settings.c.

7.55.2 Typedef Documentation

7.55.2.1 helper_task_args

```
typedef union helper_task_args helper_task_args
```

7.55.2.2 setting_helper_task

```
typedef struct setting_helper_task setting_helper_task
```

7.55.2.3 tx_all_settings_args

```
typedef struct tx_all_settings_args tx_all_settings_args
```

These are arguments passed to the "Transmit All Settings Over CANbus" Subroutine.

7.55.2.4 tx_journal_args

```
typedef struct tx_journal_args tx_journal_args
```

7.55.3 Function Documentation

7.55.3.1 handleIncomingLaptopMsg()

```
void handleIncomingLaptopMsg (
            uv_CAN_msg * msg )
```

Callback function for event where CAN message is received from laptop.

Definition at line 152 of file uvfr_settings.c.

References changeVehicleState(), CLEAR_FAULTS, uv_CAN_msg::data, DISCARD_NEW_SETTINGS, DISCARD_NEW_SETTINGS_AND_EXIT, uv_CAN_msg::dlc, END_OF_SPECIFIC_PARAMS, ENTER_PROGRAMMING_MODE, FIRMWARE_MAJOR_RELEASE, FIRMWARE_MINOR_RELEASE, FIRMWARE_PATCH_NUM, FORCE_RESTORE_FACTORY_DEFAULT, GENERIC_ACK, HANDSHAKE, is_laptop_connected, last_contact_with_laptop, uv_CAN_msg::msg_id, PROGRAMMING, REQUEST_ALL_JOURNAL_ENTRIES, REQUEST_ALL_SETTINGS, REQUEST_JOURNAL_ENTRIES_BY_TIME, REQUEST_SPECIFIC_SETTING, REQUEST_VCU_STATUS, SAVE_AND_APPLY_NEW_SETTINGS, serializeSmallE32, SET_SPECIFIC_PARAM, settings_queue, UV_OK, UV_READY, uvSendCanMSG(), uvTransmitVehicleStatus(), vcu_ack_failed_msg, and vehicle_state.

Referenced by uvSettingsInit().

7.55.3.2 nukeSettings()

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

Definition at line 260 of file uvfr_settings.c.

7.55.3.3 sendAllSettingsWorker()

```
void sendAllSettingsWorker (
            void * args )
```

Sub-task that sends all settings from the VCU to the.

Definition at line 1163 of file uvfr_settings.c.

References setting_helper_task::args, BMS_MGROUP, BMS_OFFSET, child_task_active, child_task_handle, DAQ_HEAD_MGROUP, DAQ_HEAD_OFFSET, DAQ_PARAMS1_MGROUP, DAQ_PARAMS1_OFFSET, DRIVING_MGROUP, DRIVING_OFFSET, FLASH_SBLOCK_START, GENERAL_VEH_INFO_MGROUP, GENERAL_VEH_INFO_OFFSET, IMD_MGROUP, IMD_OFFSET, setting_helper_task::meta_task_handle, MOTOR_MGROUP, MOTOR_OFFSET, OS_SETTINGS_MGROUP, OS_SETTINGS_OFFSET, PDU_MGROUP, PDU_OFFSET, tx_all_settings_args::sblock_origin, helper_task_args::setting_tx_args, setting_helper_task::status, UV_OK, and uvSendSettingGroup().

7.55.3.4 sendJournalWorker()

```
void sendJournalWorker (
            void * args )
```

Sub-task that reads out the journal.

Attention

EVENT LOGGING IS NOT IMPLEMENTED YET, DO NOT TOUCH THIS

Definition at line 1226 of file uvfr_settings.c.

7.55.3.5 settingCopy()

```
static void settingCopy (
            uint8_t * from,
            uint8_t * to,
            uint16_t length )  [inline], [static]
```

Internal function that is used to copy an arbitrary amount of data from point A to point B.

Definition at line 128 of file uvfr_settings.c.

Referenced by uvResetFlashToDefault().

7.55.3.6 setupDefaultSettings()

```
uv_status setupDefaultSettings ( )
```

Function that allocates the neccessary space for all the vehicle settings, and handles sets all of the settings structs to defaults.

Definition at line 232 of file uvfr_settings.c.

References uv_vehicle_settings::bms_settings, current_vehicle_settings, uv_vehicle_settings::daq_param_list, uv_vehicle_settings::daq_settings, default_bms_settings, default_daq_settings, default_datapoints, default_dl_↩ settings, default_imd_settings, default_os_settings, default_pdu_settings, default_vehicle, uv_vehicle_settings↩ ::driving_loop_settings, uv_vehicle_settings::flags, uv_vehicle_settings::imd_settings, mc_default_settings, uv_↩ vehicle_settings::mc_settings, uv_vehicle_settings::os_settings, uv_vehicle_settings::pdu_settings, UV_ERROR, UV_OK, and uv_vehicle_settings::veh_info.

Referenced by uvSettingsInit().

7.55.3.7 uvAwaitLaptopAckMsg()

```
uv_status uvAwaitLaptopAckMsg ( )
```

Definition at line 145 of file uvfr_settings.c.

References UV_ABORTED.

7.55.3.8 uvComputeChunkChecksum()

```
uint32_t uvComputeChunkChecksum (
            uint32_t * chunk )
```

Definition at line 448 of file uvfr_settings.c.

References CRC_POLY, and UV_OK.

Referenced by uvComputeMemRegionChecksum().

7.55.3.9 uvComputeMemRegionChecksum()

```
uv_status uvComputeMemRegionChecksum (
            void * sblock,
            uint32_t * csums,
            char mregion )
```

Definition at line 463 of file uvfr_settings.c.

References UV_OK, and uvComputeChunkChecksum().

Referenced by uvComputeSettingsChecksums().

7.55.3.10 uvComputeSettingsChecksums()

```
uv_status uvComputeSettingsChecksums (
            void * sblock,
            uint32_t * csums )
```

Definition at line 472 of file uvfr_settings.c.

References UV_OK, and uvComputeMemRegionChecksum().

Referenced by uvValidateChecksums().

7.55.3.11 uvConfigSettingTask()

```
uv_status uvConfigSettingTask (
            void * args )
```

Function to setup the parameters of the setting setter task.

Definition at line 305 of file uvfr_settings.c.

References ABOVE_NORMAL, uv_task_info::active_states, uv_task_info::deletion_states, PROGRAMMING, uv_task_info::stack_size, uv_task_info::suspension_states, uv_task_info::task_args, uv_task_info::task_function, uv_task_info::task_name, uv_task_info::task_period, uv_task_info::task_priority, UV_ERROR, UV_OK, uvCreateTask(), and uvSettingsProgrammerTask().

Referenced by uvInitStateEngine().

7.55.3.12 uvCreateTmpSettingsCopy()

```
void* uvCreateTmpSettingsCopy ( )
```

Function that creates a temporary copy of the existing settings.

Definition at line 756 of file uvfr_settings.c.

References SETTING_BRANCH_SIZE, and START_OF_USER_FLASH.

Referenced by uvSettingsProgrammerTask().

7.55.3.13 uvForceDefaultReversionUponDeviceReset()

```
uv_status uvForceDefaultReversionUponDeviceReset ( )
```

Deprecated

Definition at line 735 of file uvfr_settings.c.

References START_OF_USER_FLASH, UV_ERROR, and UV_OK.

7.55.3.14 uvLoadSettingsFromFlash()

```
uv_status uvLoadSettingsFromFlash ( )
```

This loads us some new settings from the flash memory.

Definition at line 274 of file uvfr_settings.c.

References BMS_ADDR, uv_vehicle_settings::bms_settings, current_vehicle_settings, DAQ_HEAD_ADDR, uv_vehicle_settings::daq_param_list, DAQ_PARAMS1_ADDR, uv_vehicle_settings::daq_settings, uv_vehicle_settings::driving_loop_settings, DRIVING_MGROUP, DRIVING_OFFSET, GENERAL_VEH_INFO_MGROUP, GENERAL_VEH_INFO_OFFSET, IMD_ADDR, uv_vehicle_settings::imd_settings, uv_vehicle_settings::mc_settings, MOTOR_MGROUP, MOTOR_OFFSET, uv_vehicle_settings::os_settings, OS_SETTINGS_MGROUP, OS_SETTINGS_OFFSET, PDU_MGROUP, PDU_OFFSET, uv_vehicle_settings::pdu_settings, START_OF_USER_FLASH, UV_ERROR, UV_OK, and uv_vehicle_settings::veh_info.

Referenced by uvSettingsInit().

7.55.3.15 uvOverwriteCsr()

```
uv_status uvOverwriteCsr (
            void * sblock,
            uint32_t * csr )
```

Definition at line 555 of file uvfr_settings.c.

References UV_OK.

7.55.3.16 uvResetFlashToDefault() [1/2]

```
uv_status uvResetFlashToDefault ( )
```

Referenced by uvSettingsInit(), and uvSettingsProgrammerTask().

7.55.3.17 uvResetFlashToDefault() [2/2]

```
uv_status uvResetFlashToDefault (
            void * new_sblock )
```

uvSendCanMSG(&vcu_ack_msg);

Definition at line 1057 of file uvfr_settings.c.

References BMS_MGROUP, BMS_OFFSET, DAQ_HEAD_MGROUP, DAQ_HEAD_OFFSET, DAQ_PARAMS1_MGROUP, DAQ_PARAMS1_OFFSET, DATA_MISMATCH, default_bms_settings, default_daq_settings, default_datapoints, default_dl_settings, default_imd_settings, default_os_settings, default_pdu_settings, default_vehicle, DID_NOT_FINISH_PROGRAMMING, DRIVING_MGROUP, DRIVING_OFFSET, FLASH_NOT_UNLOCKED, GENERAL_VEH_INFO_OFFSET, IMD_MGROUP, IMD_OFFSET, INVALID_SBLOCK, MAGIC_NUMBER, mc_default_settings, MOTOR_MGROUP, MOTOR_OFFSET, OS_SETTINGS_OFFSET, PDU_MGROUP, PDU_OFFSET, POST_CHECKSUM, PRE_CHECKSUM, SETTING_BRANCH_SIZE, settingCopy(), daq_loop_args::total_params_logged, UV_ERROR, UV_OK, and uvSaveSettingsToFlash().

7.55.3.18 uvSaveSettingsToFlash()

```
uv_status uvSaveSettingsToFlash (
            void * sblock,
            uint32_t * ecode )
```

Function to save a finished settings block to flash memory.

Definition at line 574 of file uvfr_settings.c.

References DATA_MISMATCH, DID_NOT_FINISH_PROGRAMMING, FIRMWARE_MAJOR_RELEASE, FIRMWARE_MINOR_RELEASE, FIRMWARE_PATCH_NUM, FLASH_NOT_UNLOCKED, FLASH_SBLOCK_START, INVALID_SBLOCK, MAGIC_NUMBER, START_OF_USER_FLASH, TOP_OF_FLASH_SBLOCK, UV_ERROR, and UV_OK.

Referenced by uvResetFlashToDefault(), and uvSettingsProgrammerTask().

7.55.3.19 uvSBlockCorruptionHandler()

```
void uvSBlockCorruptionHandler ( )
```

Function to call when the Sblock Flash has become corrupted somehow.

Definition at line 511 of file uvfr_settings.c.

References FLASH_SBLOCK_START.

7.55.3.20 uvSendSettingGroup()

```
uv_status uvSendSettingGroup (
            uint8_t * origin,
            uint8_t memgroup,
            uint8_t m_offset,
            uint16_t size )
```

Helper function for sending all settings.

Definition at line 1143 of file uvfr_settings.c.

References UV_ERROR, UV_OK, and uvSendSpecificParam().

Referenced by sendAllSettingsWorker().

7.55.3.21 uvSendSpecificParam()

```
uv_status uvSendSpecificParam (
            uint8_t * origin,
            uint8_t mgroup,
            uint8_t m_offset,
            uint8_t size )
```

Function that locates a specific setting parameter and sends the value over canbus.

Definition at line 778 of file uvfr_settings.c.

References uv_CAN_msg::data, uv_CAN_msg::dlc, uv_CAN_msg::flags, uv_CAN_msg::msg_id, START_OF_U↩SER_FLASH, UV_ERROR, UV_OK, and uvSendCanMSG().

Referenced by uvSendSettingGroup(), and uvSettingsProgrammerTask().

7.55.3.22 uvSetSettingResponseReminder()

```
uv_status uvSetSettingResponseReminder (
            uint32_t reminder )
```

This reminds the VCU to shoot off it's own message later when it reboots.

@deperecated

Definition at line 716 of file uvfr_settings.c.

References START_OF_USER_FLASH, UV_ERROR, and UV_OK.

7.55.3.23 uvSettingsInit()

```
uv_status 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 334 of file uvfr_settings.c.

References __uvInitPanic(), current_vehicle_settings, handleIncomingLaptopMsg(), insertCANMessageHandler(), SETTING_BRANCH_SIZE, setupDefaultSettings(), START_OF_USER_FLASH, UV_ERROR, UV_OK, uvLoad↩SettingsFromFlash(), uvResetFlashToDefault(), uvSendCanMSG(), uvValidateFlashSettings(), and vcu_ack_msg.

Referenced by uvInit().

7.55.3.24 uvSettingsProgrammerTask()

```
void uvSettingsProgrammerTask (
            void * args )
```

This is a task that executes the functionality necessary for vehicle diagnostics.

This task is only active once the car has been placed in programming mode.

Definition at line 864 of file uvfr_settings.c.

References changeVehicleState(), uv_task_info::cmd_data, uv_CAN_msg::data, DATA_MISMATCH, data_size, DID_NOT_FINISH_PROGRAMMING, DISCARD_NEW_SETTINGS, DISCARD_NEW_SETTINGS_AND_EXIT, END_OF_SPECIFIC_PARAMS, FLASH_NOT_UNLOCKED, FLASH_SBLOCK_START, FORCE_RESTORE_F↩ ACTORY_DEFAULT, INVALID_SBLOCK, killSelf(), POST_CHECKSUM, PRE_CHECKSUM, ptask_handle, R↩ EQUEST_ALL_JOURNAL_ENTRIES, REQUEST_ALL_SETTINGS, REQUEST_JOURNAL_ENTRIES_BY_TIME, REQUEST_SPECIFIC_SETTING, SAVE_AND_APPLY_NEW_SETTINGS, SET_SPECIFIC_PARAM, settings_↩ queue, uv_task_info::task_handle, UV_KILL_CMD, UV_OK, UV_READY, uvCreateTmpSettingsCopy(), uvReset↩ FlashToDefault(), uvSaveSettingsToFlash(), uvSendCanMSG(), uvSendSpecificParam(), uvUpdateTmpSettings(), vcu_ack_failed_msg, and vcu_ack_msg.

Referenced by uvConfigSettingTask().

7.55.3.25 uvTransmitVehicleStatus()

```
uv_status uvTransmitVehicleStatus ( )
```

Transmits the status of the vehicle, including errorcodes and similar.

Definition at line 141 of file uvfr_settings.c.

References UV_OK.

Referenced by handleIncomingLaptopMsg().

7.55.3.26 uvUpdateTmpSettings()

```
uv_status uvUpdateTmpSettings (
            uint8_t * tmp_settings,
            uint8_t memgroup,
            uint8_t m_offset,
            uint8_t size,
            uint8_t * data )
```

Updates a setting in a temporary SBlock by memory group and offset.

This indicates that you are attempting to set a setting that bridges the 128 byte "row" system the STM32F40xx uses to organise the flash memory. Attempting to write a parameter like this will result in a busfault.

Definition at line 411 of file uvfr_settings.c.

References UV_ERROR, and UV_OK.

Referenced by uvSettingsProgrammerTask().

7.55.3.27 uvValidateChecksums()

```
uv_status uvValidateChecksums (
            void * sblock )
```

Definition at line 495 of file uvfr_settings.c.

References UV_ERROR, UV_OK, and uvComputeSettingsChecksums().

7.55.3.28 uvValidateFlashSettings()

```
uv_status uvValidateFlashSettings ( )
```

Function that looks into the flash memory of the VCU, and determines if a valid set of settings exists.

This checks various parameters to ensure that the settings are useable. The first thing it looks at is the very first word of the SBLOCK. If the first word is 0x42069420, then this is a real sblock.

The second thing it checks is the firmware version, which is stored immediately after the first word, and 16 bytes of padding. These 3 16 bit ints need to be correct, in order for it to proceed. This prevents settings made for incompatible firmware versions from being used.

Definition at line 810 of file uvfr_settings.c.

References MAGIC_NUMBER, START_OF_USER_FLASH, UV_ERROR, and UV_OK.

Referenced by uvSettingsInit().

7.55.4 Variable Documentation

7.55.4.1 _e_uvdata

```
PRIVILEGED_DATA uint8_t _e_uvdata
```

7.55.4.2 _s_uvdata

```
PRIVILEGED_DATA uint8_t _s_uvdata
```

7.55.4.3 child_task_active

```
bool child_task_active = false  [static]
```

Definition at line 93 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

7.55.4.4 child_task_handle

```
TaskHandle_t child_task_handle = NULL  [static]
```

Definition at line 92 of file uvfr_settings.c.

Referenced by sendAllSettingsWorker().

7.55.4.5 current_vehicle_settings

```
uv_vehicle_settings* current_vehicle_settings = NULL
```

Definition at line 20 of file uvfr_settings.c.

Referenced by configureDaqSubTasks(), initDaqTask(), initDrivingLoop(), setupDefaultSettings(), uvInit(), uvLoad↩SettingsFromFlash(), uvSettingsInit(), and uvStartStateMachine().

7.55.4.6 default_bms_settings

```
bms_settings_t default_bms_settings
```

Definition at line 12 of file bms.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.7 default_daq_settings

```
struct daq_loop_args default_daq_settings
```

Definition at line 30 of file daq.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.8 default_datapoints

```
daq_datapoint default_datapoints[]
```

Definition at line 38 of file daq.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.9 default_dl_settings

```
struct driving_loop_args default_dl_settings
```

Definition at line 27 of file driving_loop.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.10 default_imd_settings

```
struct uv_imd_settings default_imd_settings
```

Definition at line 65 of file imd.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.11 default_pdu_settings

```
uv19_pdu_settings default_pdu_settings
```

Definition at line 28 of file uvfr_settings.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.12 default_vehicle

```
veh_gen_info default_vehicle
```

Initial value:

```
= {
    .wheel_size = 0xEEEEEEEE,
    .drive_ratio = 0xDDDDDDDD,
    .test1 = 0xCCCC,
    .test2 = 0xBBBB,
    .test3 = 0xAAAA,
    .test4 = 0x99,
    .test5 = 0x88,
    .test6 = 0x77777777
}
```

Definition at line 72 of file uvfr_settings.c.

Referenced by setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.13 is_laptop_connected

```
bool is_laptop_connected = false  [static]
```

Definition at line 84 of file uvfr_settings.c.

Referenced by handleIncomingLaptopMsg().

7.55.4.14 last_contact_with_laptop

```
TickType_t last_contact_with_laptop = 0
```

Definition at line 85 of file uvfr_settings.c.

Referenced by handleIncomingLaptopMsg().

7.55.4.15 mc_default_settings

```
struct motor_controller_settings mc_default_settings
```

Definition at line 14 of file motor_controller.c.

Referenced by MC_Request_Data(), MC_Set_Param(), MC_Startup(), MotorControllerSpinTest(), Parse_↩ Bamocar_Response(), setupDefaultSettings(), and uvResetFlashToDefault().

7.55.4.16 ptask_handle

```
TaskHandle_t ptask_handle = NULL  [static]
```

Definition at line 90 of file uvfr_settings.c.

Referenced by uvSettingsProgrammerTask().

7.55.4.17 settings_queue

```
QueueHandle_t settings_queue = NULL  [static]
```

Definition at line 88 of file uvfr_settings.c.

Referenced by handleIncomingLaptopMsg(), and uvSettingsProgrammerTask().

7.55.4.18 vcu_ack_failed_msg

```
const uv_CAN_msg vcu_ack_failed_msg
```

Initial value:

```
= {
    .flags = 0x00,
    .dlc = 1,
    .data = {0x11,0,0,0,0,0,0,0},
    .msg_id = VCU_TO_LAPTOP_ID
}
```

Pre-created message that indicates the VCU has failed to perform some requested action

Definition at line 111 of file uvfr_settings.c.

Referenced by handleIncomingLaptopMsg(), and uvSettingsProgrammerTask().

7.55.4.19 vcu_ack_msg

```
const uv_CAN_msg vcu_ack_msg
```

Initial value:

```
= {
    .flags = 0x00,
    .dlc = 1,
    .data = {0x10,0,0,0,0,0,0,0},
    .msg_id = VCU_TO_LAPTOP_ID
}
```

Pre-create message that is transmitted by the VCU upon successful completion of a settings change or diagnostic operation requested by a laptop

Definition at line 100 of file uvfr_settings.c.

Referenced by uvSettingsInit(), and uvSettingsProgrammerTask().

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 uvThrottleNonCritTasks ()
- void uvLateTaskHandler (uv_task_info *t, TickType_t tdiff, uint8_t task_tardiness)
- uv_task_info * uvCreateServiceTask ()

  Create a new service task, because fuck you, thats why.
- uv_status uvStartSVCTask (uv_task_info *t)

  Function to start a service task specifically.
- uv_status uvSuspendSVCTask (uv_task_info *t)

  Function that suspends a service task.
- uv_status uvDeleteSVCTask (uv_task_info *t)

  For when you need to delete a service task... for some reason...
- uv_status uvRestartSVCTask (uv_task_info *t)

  Function that takes a service part that may be messed up and tries to reboot it to recover.
- uv_task_info * uvGetTaskFromName (char *tsk_name)
- uv_task_info * uvGetTaskFromRTOSHandle (TaskHandle_t t_handle)

  Returns the pointer to the task info structure.
- void uvTaskPeriodEnd (uv_task_info *t)

  Function called at the end of the task period.

Variables

- static uv_task_id _next_task_id = 0
- static uv_task_info * _task_register = NULL
- static TickType_t * last_task_start_times = NULL
- static TickType_t * last_task_end_times = NULL
- uint8_t * task_tardiness
- static uv_task_id _next_svc_task_id = 0
- TaskHandle_t * scd_handle_ptr
- static volatile bool SCD_active = false
- static volatile bool throttle_daq = false
- static volatile bool nc_throttling = 1
- static QueueHandle_t state_change_queue = NULL
- rbtree * task_name_lut = NULL
- uint32_t error_bitfield [4]
- enum uv_vehicle_state_t vehicle_state = UV_BOOT
- enum uv_vehicle_state_t previous_state = UV_BOOT
- uv_task_info * task_manager = NULL
- uv_task_info * svc_task_manager = NULL
- rbtree * task_name_tree
- uv_os_settings * os_settings = NULL
- uv_os_settings default_os_settings

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 (uint8_t reset_type)

  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
- const uint8_t data_size [ ]

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 353 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 404 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 296 of file uvfr_utils.c.

Referenced by uvInit(), uvInitStateEngine(), uvSettingsInit(), 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 317 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 378 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 284 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
initADCTask(); //START THE ADCs
```

Wait for all the spawned in tasks to do their thing. This should not take that long, but we wanna be sure that everything is chill If we are say, missing a BMS, then it will not allow you to proceed past the initialisation step This is handled by a message buffer, that takes inputs from all of the tasks

We allocate space for a response from the initialization.

Clean up, clean up, everybody clean up, clean up, clean up, everybody do your share! The following code cleans up all the threads that were running, and free up used memory

Definition at line 46 of file uvfr_utils.c.

References __uvInitPanic(), uv_task_info::active_states, BMS_Init(), uv_vehicle_settings::bms_settings, CAN_R↩X_DAEMON_NAME, CAN_TX_DAEMON_NAME, CANbusRxSvcDaemon(), CANbusTxSvcDaemon(), change↩VehicleState(), 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, initADCTask(), initIMD(), initPDU(), MAX_INIT_TIME, uv_vehicle_settings::mc_settings, MC_Startup(), uv_↩init_task_args::meta_task_handle, uv_init_task_response::nchar, uv_init_task_args::specific_args, uv_init_task_↩response::status, uv_task_info::task_function, uv_task_info::task_name, UV_OK, UV_READY, uvCreateService↩Task(), uvInitStateEngine(), uvSettingsInit(), uvStartStateMachine(), and uvStartTask().

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 426 of file uvfr_utils.c.

References UV_ERROR, UV_OK, and UV_WARNING.

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

7.57.2.9 uvSysResetDaemon()

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

Definition at line 255 of file uvfr_utils.c.

7.57.2.10 uvUtilsReset()

```
enum uv_status_t uvUtilsReset (
            uint8_t reset_type )
```

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 272 of file uvfr_utils.c.

References UV_OK.

7.57.3 Variable Documentation

7.57.3.1 data_size

```
const uint8_t data_size[]
```

Initial value:

```
= {1,1,
        2,2,
        4,4,
        4,8,
        8,8,
        0}
```

Definition at line 17 of file uvfr_utils.c.

Referenced by insertParamToRegister(), and uvSettingsProgrammerTask().

7.57.3.2 init_task_handle

```
TaskHandle_t init_task_handle
```

Definition at line 51 of file freertos.c.

Referenced by MX_FREERTOS_Init(), and uvInit().

7.57.3.3 reset_handle

```
TaskHandle_t reset_handle = NULL
```

Definition at line 15 of file uvfr_utils.c.

7.57.3.4 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, 43
_BV_16
    Utility Macros, 43
_BV_32
    Utility Macros, 44
_BV_8
    Utility Macros, 44
_LONGEST_SC_TIME
    uvfr_state_engine.h, 241
_NUM_ERRORS_
    errorLUT.h, 145
_NUM_LOGGABLE_PARAMS
    daq.h, 134
_SC_DAEMON_PERIOD
    uvfr_state_engine.h, 242
_SRC_UVFR_DAQ
    daq.c, 275
_UV_DEFAULT_TASK_INSTANCES
    uvfr_state_engine.h, 242
_UV_DEFAULT_TASK_PERIOD
    uvfr_state_engine.h, 242
_UV_DEFAULT_TASK_STACK_SIZE
    uvfr_state_engine.h, 242
_UV_MIN_TASK_PERIOD
    uvfr_state_engine.h, 242
__attribute__
    syscalls.c, 339
__io_getchar
    syscalls.c, 340
__io_putchar
    syscalls.c, 340
__sbrk_heap_end
    sysmem.c, 345
__uvCANtxCritSection
    can.c, 268
__uvFreeCritSection
    uvfr_utils.c, 370
__uvFreeOS
    uvfr_utils.c, 370
__uvInitPanic
    uvfr_utils.c, 370
    uvfr_utils.h, 253
__uvMallocCritSection
    uvfr_utils.c, 371
__uvMallocOS
    uvfr_utils.c, 371
__uvPanic
    State Engine Internals, 31
_close
    syscalls.c, 340
_e_uvdata
    uvfr_settings.c, 362
_execve
    syscalls.c, 340
_exit
    syscalls.c, 340
_fork
    syscalls.c, 341
_fstat
    syscalls.c, 341
_getpid
    syscalls.c, 341
_isatty
    syscalls.c, 341
_kill
    syscalls.c, 341
_link
    syscalls.c, 342
_lseek
    syscalls.c, 342
_next_svc_task_id
    State Engine, 12
_next_task_id
    State Engine, 12
_open
    syscalls.c, 342
_s_uvdata
    uvfr_settings.c, 362
_sbrk
    sysmem.c, 344
_stat
    syscalls.c, 342
_stateChangeDaemon
    State Engine Internals, 31
_task_register
    State Engine, 12
_times
    syscalls.c, 342
_unlink
    syscalls.c, 343
_uvCloseSDC_canBased
    uvfr_vehicle_commands.h, 259
_uvHonkHorn_canBased
    uvfr_vehicle_commands.h, 259
_uvOpenSDC_canBased
    uvfr_vehicle_commands.h, 259
_uvSilenceHorn_canBased

uvfr_vehicle_commands.h, 259
_uvStartCoolantPump_canBased
uvfr_vehicle_commands.h, 259
_uvStopCoolantPump_canBased
uvfr_vehicle_commands.h, 259
_uvValidateSpecificTask
State Engine Internals, 33
_wait
syscalls.c, 343

a
s_curve_torque_map_args, 87
ABOVE_NORMAL
State Engine API, 25
absolute_max_acc_pwr
driving_loop_args, 70
absolute_max_accum_current
driving_loop_args, 70
absolute_max_motor_rpm
driving_loop_args, 70
absolute_max_motor_torque
driving_loop_args, 70
AC_current_offset_fault
motor_controller.h, 175
accel
uvfr_utils.h, 252
accelerate_ramp
motor_controller.h, 173
ACCELERATION
driving_loop.c, 284
ACCELERATOR_PEDAL_RATIO
daq.h, 135
access_control_info, 61
bin_semaphore, 61
mutex, 61
semaphore, 61
uvfr_utils.h, 248
access_control_t
uvfr_utils.h, 251
access_control_type
uvfr_utils.h, 248
ACCUM_POWER
daq.h, 135
ACCUM_POWER_LIMIT
daq.h, 135
accum_regen_soc_threshold
driving_loop_args, 70
ACCUM_SOC
daq.h, 135
ACCUM_SOH
daq.h, 135
activation_time
p_status, 82
active_states
uv_task_info, 108
adc.c
adc1_APPS1, 263
adc1_APPS2, 263
adc1_BPS1, 264
adc1_BPS2, 264
adc_buf1, 264
DEBUG_TOGGLE_LED, 261
hadc1, 264
hadc2, 264
HAL_ADC_MspDeInit, 261
HAL_ADC_MspInit, 261
hdma_adc1, 265
initADCTask, 262
MX_ADC1_Init, 262
MX_ADC2_Init, 262
processADCBuffer, 263
StartADCTask, 263
adc.h
ADC1_BUF_LEN, 122
ADC1_CHNL_CNT, 122
ADC1_MAX_VOLT, 122
ADC1_MIN_VOLT, 122
ADC1_SAMPLES, 122
ADC2_BUF_LEN, 123
ADC2_CHNL_CNT, 123
ADC2_MAX_VOLT, 123
ADC2_MIN_VOLT, 123
ADC2_SAMPLES, 123
hadc1, 125
hadc2, 125
initADCTask, 124
MX_ADC1_Init, 124
MX_ADC2_Init, 124
processADCBuffer, 124
adc1_APPS1
adc.c, 263
driving_loop.c, 288
main.c, 311
adc1_APPS2
adc.c, 263
driving_loop.c, 289
main.c, 311
adc1_BPS1
adc.c, 264
driving_loop.c, 289
main.c, 312
adc1_BPS2
adc.c, 264
driving_loop.c, 289
main.c, 312
ADC1_BUF_LEN
adc.h, 122
ADC1_CHNL_CNT
adc.h, 122
ADC1_MAX_VOLT
adc.h, 122
ADC1_MIN_VOLT
adc.h, 122
ADC1_SAMPLES
adc.h, 122
ADC2_BUF_LEN
adc.h, 123

ADC2_CHNL_CNT
adc.h, 123
adc2_CoolantFlow
main.c, 312
adc2_CoolantTemp
main.c, 312
ADC2_MAX_VOLT
adc.h, 123
ADC2_MIN_VOLT
adc.h, 123
ADC2_SAMPLES
adc.h, 123
adc_buf1
adc.c, 264
main.c, 312
adc_buf2
main.c, 313
ADC_IRQHandler
stm32f4xx_it.c, 334
stm32f4xx_it.h, 217
ADC_measurement_problem
motor_controller.h, 175
ADC_sequencer_problem
motor_controller.h, 175
addTaskToTaskRegister
State Engine Internals, 33
AHBPrescTable
STM32F4xx_System_Private_Variables, 57
air_temperature
motor_controller.h, 176
amogus
oled.h, 180
APBPrescTable
STM32F4xx_System_Private_Variables, 57
applyTorqueFilter
driving_loop.c, 284
APPS1_ADC_VAL
daq.h, 135
APPS2_ADC_VAL
daq.h, 135
apps_bottom
driving_loop_args, 71
apps_implausibility_recovery_threshold
driving_loop_args, 71
apps_plausibility_check_threshold
driving_loop_args, 71
apps_top
driving_loop_args, 71
args
setting_helper_task, 88
assert_param
stm32f4xx_hal_conf.h, 198
associateDaqParamWithVar
daq.c, 276
daq.h, 136
AUTOCROSS
driving_loop.c, 284
auxiliary_voltage_min_limit
motor_controller.h, 175
AVG_CELL_TEMP
daq.h, 135

b
s_curve_torque_map_args, 87
battery_voltage
imd.h, 161
BELOW_NORMAL
State Engine API, 25
bin_semaphore
access_control_info, 61
BLACK
rb_tree.h, 186
bleed_resistor_overload
motor_controller.h, 175
bleeder_resistor_warning
motor_controller.h, 175
Blue_LED_GPIO_Port
main.h, 167
Blue_LED_Pin
main.h, 167
BMS
uvfr_utils.h, 252
bms.c
BMS_Init, 265
default_bms_settings, 266
handleBMSVoltageMsg, 266
bms.h
BMS_Init, 126
bms_settings_t, 126
DEFAULT_BMS_CAN_TIMEOUT, 126
BMS_ADDR
uvfr_settings.h, 226
BMS_CAN_timeout
bms_settings_t, 62
BMS_CURRENT
daq.h, 135
BMS_ERRORS
daq.h, 135
BMS_Init
bms.c, 265
bms.h, 126
BMS_MGROUP
uvfr_settings.h, 226
BMS_OFFSET
uvfr_settings.h, 227
bms_settings
uv_vehicle_settings, 115
bms_settings_t, 62
bms.h, 126
BMS_CAN_timeout, 62
max_temp, 62
BMS_VOLTAGE
daq.h, 135
bool
uvfr_utils.h, 249
BPS1_ADC_VAL
daq.h, 135

BPS2_ADC_VAL
daq.h, 135
bps_implausibility_recovery_threshold
driving_loop_args, 71
bps_plausibility_check_threshold
driving_loop_args, 72
brake_light_channel
uv19_pdu_settings, 93
BRAKE_PRESSURE_PA
daq.h, 135
BusFault_Handler
stm32f4xx_it.c, 334
stm32f4xx_it.h, 217
buzzer_channel
uv19_pdu_settings, 93

c
s_curve_torque_map_args, 87
calculateThrottlePercentage
driving_loop.c, 285
callback_table_mutex
can.c, 271
callFunctionFromCANid
can.c, 268
can.c
__uvCANtxCritSection, 268
callback_table_mutex, 271
callFunctionFromCANid, 268
CAN_Callback, 268
CAN_callback_table, 272
CANbusRxSvcDaemon, 269
CANbusTxSvcDaemon, 269
generateHash, 269
HAL_CAN_ERROR_INVALID_CALLBACK, 268
HAL_CAN_MspDeInit, 270
HAL_CAN_MspInit, 270
HAL_CAN_RxFifo0MsgPendingCallback, 270
HAL_CAN_RxFifo1MsgPendingCallback, 270
handleCANbusError, 271
hcan2, 272
MX_CAN2_Init, 271
nuke_hash_table, 271
Rx_msg_queue, 272
table_size, 268
Tx_msg_queue, 272
can.h
CAN_RX_DAEMON_NAME, 128
CAN_TX_DAEMON_NAME, 128
CANbusRxSvcDaemon, 128
CANbusTxSvcDaemon, 129
HAL_CAN_RxFifo0MsgPendingCallback, 129
HAL_CAN_RxFifo1MsgPendingCallback, 129
hcan2, 130
MX_CAN2_Init, 130
nuke_hash_table, 130
uv_CAN_msg, 128
uv_status, 128
CAN2_RX0_IRQHandler
stm32f4xx_it.c, 334
stm32f4xx_it.h, 217
CAN2_RX1_IRQHandler
stm32f4xx_it.c, 335
stm32f4xx_it.h, 217
CAN2_TX_IRQHandler
stm32f4xx_it.c, 335
stm32f4xx_it.h, 218
CAN_Callback, 63
can.c, 268
CAN_id, 63
function, 63
next, 63
CAN_callback_table
can.c, 272
can_channel
daq_loop_args, 67
CAN_id
CAN_Callback, 63
can_id
daq_datapoint, 65
daq_param_list_node, 68
can_id_rx
motor_controller_settings, 80
can_id_tx
motor_controller_settings, 80
CAN_IDs
constants.h, 131
CAN_RX_DAEMON_NAME
can.h, 128
CAN_timeout_error
motor_controller.h, 175
CAN_TX_DAEMON_NAME
can.h, 128
CANbusRxSvcDaemon
can.c, 269
can.h, 128
CANbusTxSvcDaemon
can.c, 269
can.h, 129
CANNOT_PERFORM_REQUEST
uvfr_settings.h, 236
changeVehicleState
State Engine API, 27
check_ecode_ID
motor_controller.h, 175
checkBlackHeight
rb_tree.c, 322
checkOrder
rb_tree.c, 322
child_task_active
uvfr_settings.c, 362
child_task_handle
uvfr_settings.c, 363
clear_errors
motor_controller.h, 174
CLEAR_FAULTS
uvfr_settings.h, 236
cmd_data

uv_task_info, 108
CMSIS, 51
color
rbnode, 83
compare
rbtree, 85
compareTaskByName
State Engine, 11
configASSERT
FreeRTOSConfig.h, 146
configCHECK_FOR_STACK_OVERFLOW
FreeRTOSConfig.h, 147
configCPU_CLOCK_HZ
FreeRTOSConfig.h, 147
configENABLE_BACKWARD_COMPATIBILITY
FreeRTOSConfig.h, 147
configENABLE_FPU
FreeRTOSConfig.h, 147
configENABLE_MPU
FreeRTOSConfig.h, 147
configKERNEL_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 148
configLIBRARY_LOWEST_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 148
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 148
configMAX_CO_ROUTINE_PRIORITIES
FreeRTOSConfig.h, 148
configMAX_PRIORITIES
FreeRTOSConfig.h, 148
configMAX_SYSCALL_INTERRUPT_PRIORITY
FreeRTOSConfig.h, 148
configMAX_TASK_NAME_LEN
FreeRTOSConfig.h, 149
configMESSAGE_BUFFER_LENGTH_TYPE
FreeRTOSConfig.h, 149
configMINIMAL_STACK_SIZE
FreeRTOSConfig.h, 149
configPRIO_BITS
FreeRTOSConfig.h, 149
configQUEUE_REGISTRY_SIZE
FreeRTOSConfig.h, 149
configRECORD_STACK_HIGH_ADDRESS
FreeRTOSConfig.h, 149
configSUPPORT_DYNAMIC_ALLOCATION
FreeRTOSConfig.h, 150
configSUPPORT_STATIC_ALLOCATION
FreeRTOSConfig.h, 150
configTICK_RATE_HZ
FreeRTOSConfig.h, 150
configTIMER_QUEUE_LENGTH
FreeRTOSConfig.h, 150
configTIMER_TASK_PRIORITY
FreeRTOSConfig.h, 150
configTIMER_TASK_STACK_DEPTH
FreeRTOSConfig.h, 150
configTOTAL_HEAP_SIZE
FreeRTOSConfig.h, 151
configureDaqSubTasks
daq.c, 276
configUSE_16_BIT_TICKS
FreeRTOSConfig.h, 151
configUSE_APPLICATION_TASK_TAG
FreeRTOSConfig.h, 151
configUSE_CO_ROUTINES
FreeRTOSConfig.h, 151
configUSE_COUNTING_SEMAPHORES
FreeRTOSConfig.h, 151
configUSE_IDLE_HOOK
FreeRTOSConfig.h, 151
configUSE_MALLOC_FAILED_HOOK
FreeRTOSConfig.h, 152
configUSE_MUTEXES
FreeRTOSConfig.h, 152
configUSE_PORT_OPTIMISED_TASK_SELECTION
FreeRTOSConfig.h, 152
configUSE_PREEMPTION
FreeRTOSConfig.h, 152
configUSE_TICK_HOOK
FreeRTOSConfig.h, 152
configUSE_TIMERS
FreeRTOSConfig.h, 153
constants.c
RxData, 273
RxHeader, 273
TxData, 273
TxHeader, 274
TxMailbox, 274
constants.h
CAN_IDs, 131
IMD_CAN_ID_Rx, 131
IMD_CAN_ID_Tx, 131
MC_CAN_ID_Rx, 131
MC_CAN_ID_Tx, 131
PDU_CAN_ID_Tx, 131
RxData, 131
RxHeader, 131
TxData, 132
TxHeader, 132
TxMailbox, 132
control_map_fn
drivingMode, 76
coolant_pump_channel
uv19_pdu_settings, 93
Core/Inc/adc.h, 121
Core/Inc/bms.h, 125
Core/Inc/can.h, 127
Core/Inc/constants.h, 131
Core/Inc/daq.h, 133
Core/Inc/dash.h, 137
Core/Inc/dma.h, 139
Core/Inc/driving_loop.h, 139
Core/Inc/errorLUT.h, 145
Core/Inc/FreeRTOSConfig.h, 145
Core/Inc/gpio.h, 157
Core/Inc/imd.h, 158

Core/Inc/main.h, 166
Core/Inc/motor_controller.h, 169
Core/Inc/odometer.h, 178
Core/Inc/oled.h, 180
Core/Inc/pdu.h, 181
Core/Inc/rb_tree.h, 185
Core/Inc/spi.h, 194
Core/Inc/stm32f4xx_hal_conf.h, 196
Core/Inc/stm32f4xx_it.h, 216
Core/Inc/temp_monitoring.h, 220
Core/Inc/tim.h, 221
Core/Inc/uvfr_global_config.h, 222
Core/Inc/uvfr_settings.h, 224
Core/Inc/uvfr_state_engine.h, 238
Core/Inc/uvfr_utils.h, 244
Core/Inc/uvfr_vehicle_commands.h, 257
Core/Src/adc.c, 260
Core/Src/bms.c, 265
Core/Src/can.c, 266
Core/Src/constants.c, 273
Core/Src/daq.c, 274
Core/Src/dash.c, 281
Core/Src/dma.c, 282
Core/Src/driving_loop.c, 283
Core/Src/freertos.c, 290
Core/Src/gpio.c, 295
Core/Src/imd.c, 296
Core/Src/main.c, 307
Core/Src/motor_controller.c, 313
Core/Src/odometer.c, 316
Core/Src/oled.c, 317
Core/Src/pdu.c, 318
Core/Src/rb_tree.c, 321
Core/Src/spi.c, 328
Core/Src/stm32f4xx_hal_msp.c, 329
Core/Src/stm32f4xx_hal_timebase_tim.c, 330
Core/Src/stm32f4xx_it.c, 333
Core/Src/syscalls.c, 338
Core/Src/sysmem.c, 344
Core/Src/system_stm32f4xx.c, 345
Core/Src/temp_monitoring.c, 346
Core/Src/tim.c, 348
Core/Src/uvfr_settings.c, 349
Core/Src/uvfr_state_engine.c, 366
Core/Src/uvfr_utils.c, 369
Core/Src/uvfr_vehicle_commands.c, 375
count
    rbtree, 85
CRC_MGROUP1
    uvfr_settings.h, 227
CRC_MGROUP2
    uvfr_settings.h, 227
CRC_POLY
    uvfr_settings.h, 227
critical_AC_current
    motor_controller.h, 175
curr_daq_settings
    daq.c, 279
CURRENT_ACTUAL
    motor_controller.h, 172
current_derate_temperature
    motor_controller.h, 176
CURRENT_DRIVING_MODE
    daq.h, 135
current_feed_forward
    motor_controller.h, 173
current_vehicle_settings
    uvfr_settings.c, 363
    uvfr_settings.h, 238

daq.c
    _SRC_UVFR_DAQ, 275
    associateDaqParamWithVar, 276
    configureDaqSubTasks, 276
    curr_daq_settings, 279
    daq_child_task, 276
    daq_param_list_node, 276
    daq_tlist, 279
    daqMasterTask, 276
    daqSubTask, 277
    datapoints, 279
    default_daq_settings, 280
    default_datapoints, 280
    initDaqTask, 277
    insertParamToParamList, 277
    insertParamToRegister, 278
    param_bank, 280
    param_ptrs, 280
    sendAllParamsFromList, 278
    sendDaqMsg, 278
    startDaqSubTasks, 278
    stopDaqSubTasks, 279
    tmp_daq_msg, 281
daq.h
    _NUM_LOGGABLE_PARAMS, 134
    ACCELERATOR_PEDAL_RATIO, 135
    ACCUM_POWER, 135
    ACCUM_POWER_LIMIT, 135
    ACCUM_SOC, 135
    ACCUM_SOH, 135
    APPS1_ADC_VAL, 135
    APPS2_ADC_VAL, 135
    associateDaqParamWithVar, 136
    AVG_CELL_TEMP, 135
    BMS_CURRENT, 135
    BMS_ERRORS, 135
    BMS_VOLTAGE, 135
    BPS1_ADC_VAL, 135
    BPS2_ADC_VAL, 135
    BRAKE_PRESSURE_PA, 135
    CURRENT_DRIVING_MODE, 135
    daq_datapoint, 134
    daq_loop_args, 134
    daqMasterTask, 136
    IMD_ERRORS, 135
    IMD_STATUS, 135
    IMD_VOLTAGE, 135

initDaqTask, 136
loggable_params, 134
MAX_CELL_TEMP, 135
MAX_LOGGABLE_PARAMS, 136
MC_CURRENT, 135
MC_ERRORS, 135
MC_TEMP, 135
MC_VOLTAGE, 135
MIN_CELL_TEMP, 135
MOTOR_CURRENT, 135
MOTOR_RPM, 135
MOTOR_TEMP, 135
POWER_DERATE_FACTOR, 135
SUS_DAMPER_FL, 135
SUS_DAMPER_RL, 135
SUS_DAMPER_RR, 135
SUS_DUMPER_FR, 135
uv_status, 134
VCU_CURRENT_UPTIME, 135
VCU_ERROR_BITFIELD1, 135
VCU_ERROR_BITFIELD2, 135
VCU_ERROR_BITFIELD3, 135
VCU_ERROR_BITFIELD4, 135
VCU_TOTAL_UPTIME, 135
VCU_VEHICLE_STATE, 135
VEH_DISTANCE_RUN, 135
VEH_DISTANCE_TOTAL, 135
VEH_LAPNUM, 135
VEH_SPEED, 135
WSS_F_AVG, 135
WSS_FL, 135
WSS_FR, 135
WSS_R_AVG, 136
WSS_RL, 135
WSS_RR, 135
WSS_SLIP, 136
daq_child_priority
daq_loop_args, 67
daq_child_task, 64
daq.c, 276
meta_task_handle, 64
next_task, 64
param_list, 64
period, 64
daq_datapoint, 65
can_id, 65
daq.h, 134
param, 65
period, 66
type, 66
DAQ_HEAD_ADDR
uvfr_settings.h, 227
DAQ_HEAD_MGROUP
uvfr_settings.h, 227
DAQ_HEAD_OFFSET
uvfr_settings.h, 228
daq_loop_args, 66
can_channel, 67
daq.h, 134
daq_child_priority, 67
minimum_daq_period, 67
throttle_daq_to_preserve_performance, 67
total_params_logged, 67
daq_param_list
uv_vehicle_settings, 115
daq_param_list_node, 68
can_id, 68
daq.c, 276
next, 68
param, 68
size, 69
DAQ_PARAMS1_ADDR
uvfr_settings.h, 228
DAQ_PARAMS1_MGROUP
uvfr_settings.h, 228
DAQ_PARAMS1_OFFSET
uvfr_settings.h, 228
DAQ_PARAMS2_ADDR
uvfr_settings.h, 228
DAQ_PARAMS2_MGROUP
uvfr_settings.h, 228
DAQ_PARAMS2_OFFSET
uvfr_settings.h, 229
DAQ_PARAMS3_ADDR
uvfr_settings.h, 229
DAQ_PARAMS3_MGROUP
uvfr_settings.h, 229
DAQ_PARAMS3_OFFSET
uvfr_settings.h, 229
daq_settings
uv_vehicle_settings, 116
daq_tlist
daq.c, 279
daqMasterTask
daq.c, 276
daq.h, 136
daqSubTask
daq.c, 277
dash.c
Update_Batt_Temp, 281
Update_RPM, 281
Update_State_Of_Charge, 282
dash.h
Dash_Battery_Temperature, 138
dash_can_ids, 137
Dash_Motor_Temperature, 138
Dash_RPM, 138
Dash_State_of_Charge, 138
Update_Batt_Temp, 138
Update_RPM, 138
Update_State_Of_Charge, 138
Dash_Battery_Temperature
dash.h, 138
dash_can_ids
dash.h, 137
Dash_Motor_Temperature

dash.h, 138
Dash_RPM
dash.h, 138
Dash_State_of_Charge
dash.h, 138
data
rbnode, 83
uv_CAN_msg, 95
DATA_CACHE_ENABLE
stm32f4xx_hal_conf.h, 199
DATA_MISMATCH
uvfr_settings.c, 352
data_size
uvfr_utils.c, 375
uvfr_utils.h, 256
data_type
uvfr_utils.h, 251
datapoints
daq.c, 279
DC_bus_voltage
motor_controller.h, 172
DEBUG_CAN_IN_MAIN
main.c, 308
DEBUG_TOGGLE_LED
adc.c, 261
DebugMon_Handler
stm32f4xx_it.c, 335
stm32f4xx_it.h, 218
DEFAULT_BMS_CAN_TIMEOUT
bms.h, 126
default_bms_settings
bms.c, 266
uvfr_settings.c, 363
default_daq_settings
daq.c, 280
uvfr_settings.c, 363
default_datapoints
daq.c, 280
uvfr_settings.c, 363
default_dl_settings
driving_loop.c, 289
uvfr_settings.c, 364
default_imd_settings
imd.c, 302
uvfr_settings.c, 364
default_os_settings
State Engine, 12
default_pdu_settings
pdu.c, 321
uvfr_settings.c, 364
default_vehicle
uvfr_settings.c, 364
defaultTaskHandle
freertos.c, 293
deleteRepair
rb_tree.c, 322
deletion_delay
uv_task_info, 108
deletion_states
uv_task_info, 108
deserializeBigE16
Utility Macros, 44
deserializeBigE32
Utility Macros, 44
deserializeSmallE16
Utility Macros, 44
deserializeSmallE32
Utility Macros, 45
destroy
rbtree, 85
destroyAllNodes
rb_tree.c, 323
device
uv_init_task_response, 100
DID_NOT_FINISH_PROGRAMMING
uvfr_settings.c, 352
disable_brake_light_msg
pdu.h, 183
disable_coolant_pump_msg
pdu.h, 182
disable_left_cooling_fan_msg
pdu.h, 182
disable_motor_controller_msg
pdu.h, 183
disable_right_cooling_fan_msg
pdu.h, 182
disable_shutdown_circuit_msg
pdu.h, 183
disable_speaker_msg
pdu.h, 183
DISCARD_NEW_SETTINGS
uvfr_settings.h, 236
DISCARD_NEW_SETTINGS_AND_EXIT
uvfr_settings.h, 236
dismantling_ramp
motor_controller.h, 173
DL_internal_state
driving_loop.h, 142
dlc
uv_CAN_msg, 95
dm_name
drivingMode, 76
dma.c
MX_DMA_Init, 283
dma.h
MX_DMA_Init, 139
DMA2_Stream0_IRQHandler
stm32f4xx_it.c, 335
stm32f4xx_it.h, 218
dmodes
driving_loop_args, 72
DP83848_PHY_ADDRESS
stm32f4xx_hal_conf.h, 199
drive_ratio
veh_gen_info, 118
DRIVING_ADDR

uvfr_settings.h, 229
driving_args
driving_loop.c, 289
driving_loop.c
ACCELERATION, 284
adc1_APPS1, 288
adc1_APPS2, 289
adc1_BPS1, 289
adc1_BPS2, 289
applyTorqueFilter, 284
AUTOCROSS, 284
calculateThrottlePercentage, 285
default_dl_settings, 289
driving_args, 289
ENDURANCE, 284
getKValue, 285
initDrivingLoop, 285
is_accelerating, 290
mapThrottleToTorque, 285
performSafetyChecks, 286
sendTorqueToMotorController, 286
StartDrivingLoop, 287
T_PREV, 290
T_REQ, 290
driving_loop.h
DL_internal_state, 142
driving_loop_args, 140
drivingMode, 141
drivingModeParams, 141
Erroneous, 142
exp_speed_map, 143
exp_torque_map, 143
exp_torque_map_args, 141
Implausible, 142
initDrivingLoop, 143
linear_speed_map, 143
linear_torque_map, 143
linear_torque_map_args, 141
map_mode, 142
MC_POWER, 141
MC_RPM, 141
MC_Torque, 142
Plausible, 142
s_curve_speed_map, 143
s_curve_torque_map, 143
s_curve_torque_map_args, 142
StartDrivingLoop, 143
driving_loop_args, 69
absolute_max_acc_pwr, 70
absolute_max_accum_current, 70
absolute_max_motor_rpm, 70
absolute_max_motor_torque, 70
accum_regen_soc_threshold, 70
apps_bottom, 71
apps_implausibility_recovery_threshold, 71
apps_plausibility_check_threshold, 71
apps_top, 71
bps_implausibility_recovery_threshold, 71
bps_plausibility_check_threshold, 72
dmodes, 72
driving_loop.h, 140
max_accum_current_5s, 72
max_apps1_value, 72
max_apps2_value, 72
max_apps_offset, 73
max_BPS_value, 73
min_apps1_value, 73
min_apps2_value, 73
min_apps_offset, 73
min_apps_value, 74
min_BPS_value, 74
num_driving_modes, 74
period, 74
regen_rpm_cutoff, 74
driving_loop_settings
uv_vehicle_settings, 116
DRIVING_MGROUP
uvfr_settings.h, 229
DRIVING_OFFSET
uvfr_settings.h, 230
drivingLoopArgs, 75
drivingMode, 75
control_map_fn, 76
dm_name, 76
driving_loop.h, 141
flags, 76
map_fn_params, 76
max_acc_pwr, 76
max_current, 77
max_motor_torque, 77
drivingModeParams, 77
driving_loop.h, 141

e_code
uv_internal_params, 101
ecode_timeout_error
motor_controller.h, 175
econ
uvfr_utils.h, 252
ECUMASTER_PMU
uvfr_global_config.h, 222
enable_brake_light_msg
pdu.h, 183
enable_coolant_pump_msg
pdu.h, 182
ENABLE_FLASH_SETTINGS
uvfr_settings.h, 230
enable_left_cooling_fan_msg
pdu.h, 182
enable_motor_controller_msg
pdu.h, 183
enable_right_cooling_fan_msg
pdu.h, 182
enable_shutdown_circuit_msg
pdu.h, 183
enable_speaker_msg
pdu.h, 183

END_OF_SPECIFIC_PARAMS
uvfr_settings.h, 236
end_time
tx_journal_args, 92
endianSwap
Utility Macros, 45
endianSwap16
Utility Macros, 45
endianSwap32
Utility Macros, 45
endianSwap8
Utility Macros, 45
ENDURANCE
driving_loop.c, 284
ENTER_PROGRAMMING_MODE
uvfr_settings.h, 236
environ
syscalls.c, 343
eprom_read_error
motor_controller.h, 175
Err_CH
imd.h, 159
Err_clock
imd.h, 159
Err_temp
imd.h, 159
Err_Vexi
imd.h, 159
Err_Vpwr
imd.h, 159
Err_Vx1
imd.h, 159
Err_Vx2
imd.h, 159
Err_VxR
imd.h, 159
Err_Watchdog
imd.h, 159
errmsg
uv_init_task_response, 100
Erroneous
driving_loop.h, 142
error_bitfield
State Engine, 13
Error_flags
imd.h, 161
Error_Handler
main.c, 308
main.h, 168
errorLUT.h
_NUM_ERRORS_, 145
ETH_RX_BUF_SIZE
stm32f4xx_hal_conf.h, 199
ETH_RXBUFNB
stm32f4xx_hal_conf.h, 199
ETH_TX_BUF_SIZE
stm32f4xx_hal_conf.h, 199
ETH_TXBUFNB
stm32f4xx_hal_conf.h, 199
Exc_off
imd.h, 160
exp_speed_map
driving_loop.h, 143
exp_torque_map
driving_loop.h, 143
exp_torque_map_args, 77
driving_loop.h, 141
gamma, 78
offset, 78
expected_isolation_capacitances
uv_imd_settings, 97
EXTERNAL_CLOCK_VALUE
stm32f4xx_hal_conf.h, 200
EXTI0_IRQHandler
stm32f4xx_it.c, 336
stm32f4xx_it.h, 218

false
Utility Macros, 46
fan1_channel
uv19_pdu_settings, 93
fan2_channel
uv19_pdu_settings, 93
feedback_signal_error
motor_controller.h, 175
feedback_signal_problem
motor_controller.h, 175
FIRMWARE_MAJOR_RELEASE
uvfr_global_config.h, 223
FIRMWARE_MINOR_RELEASE
uvfr_global_config.h, 223
FIRMWARE_PATCH_NUM
uvfr_global_config.h, 223
firmware_version
motor_controller.h, 174
FIRMWARE_VERSION_REGISTER
motor_controller.h, 170
flags
drivingMode, 76
uv_CAN_msg, 96
uv_vehicle_settings, 116
FLASH_NOT_UNLOCKED
uvfr_settings.c, 352
FLASH_OK
uvfr_settings.c, 352
FLASH_SBLOCK_START
uvfr_settings.h, 230
FORCE_RESTORE_FACTORY_DEFAULT
uvfr_settings.h, 236
freertos.c
defaultTaskHandle, 293
init_settings, 293
init_task_handle, 294
MX_FREERTOS_Init, 291
StartDefaultTask, 292
vApplicationGetIdleTaskMemory, 292
vApplicationGetTimerTaskMemory, 292

vApplicationIdleHook, 292
vApplicationMallocFailedHook, 293
vApplicationStackOverflowHook, 293
vApplicationTickHook, 293
xIdleStack, 294
xIdleTaskTCBBuffer, 294
xTimerStack, 294
xTimerTaskTCBBuffer, 295
FreeRTOSConfig.h
configASSERT, 146
configCHECK_FOR_STACK_OVERFLOW, 147
configCPU_CLOCK_HZ, 147
configENABLE_BACKWARD_COMPATIBILITY, 147
configENABLE_FPU, 147
configENABLE_MPU, 147
configKERNEL_INTERRUPT_PRIORITY, 148
configLIBRARY_LOWEST_INTERRUPT_PRIORITY, 148
configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY, 148
configMAX_CO_ROUTINE_PRIORITIES, 148
configMAX_PRIORITIES, 148
configMAX_SYSCALL_INTERRUPT_PRIORITY, 148
configMAX_TASK_NAME_LEN, 149
configMESSAGE_BUFFER_LENGTH_TYPE, 149
configMINIMAL_STACK_SIZE, 149
configPRIO_BITS, 149
configQUEUE_REGISTRY_SIZE, 149
configRECORD_STACK_HIGH_ADDRESS, 149
configSUPPORT_DYNAMIC_ALLOCATION, 150
configSUPPORT_STATIC_ALLOCATION, 150
configTICK_RATE_HZ, 150
configTIMER_QUEUE_LENGTH, 150
configTIMER_TASK_PRIORITY, 150
configTIMER_TASK_STACK_DEPTH, 150
configTOTAL_HEAP_SIZE, 151
configUSE_16_BIT_TICKS, 151
configUSE_APPLICATION_TASK_TAG, 151
configUSE_CO_ROUTINES, 151
configUSE_COUNTING_SEMAPHORES, 151
configUSE_IDLE_HOOK, 151
configUSE_MALLOC_FAILED_HOOK, 152
configUSE_MUTEXES, 152
configUSE_PORT_OPTIMISED_TASK_SELECTION, 152
configUSE_PREEMPTION, 152
configUSE_TICK_HOOK, 152
configUSE_TIMERS, 153
INCLUDE_eTaskGetState, 153
INCLUDE_pcTaskGetTaskName, 153
INCLUDE_uxTaskGetStackHighWaterMark, 153
INCLUDE_uxTaskGetStackHighWaterMark2, 153
INCLUDE_uxTaskPriorityGet, 153
INCLUDE_vTaskCleanUpResources, 154
INCLUDE_vTaskDelay, 154
INCLUDE_vTaskDelayUntil, 154
INCLUDE_vTaskDelete, 154
INCLUDE_vTaskPrioritySet, 154
INCLUDE_vTaskSuspend, 154
INCLUDE_xEventGroupSetBitFromISR, 155
INCLUDE_xQueueGetMutexHolder, 155
INCLUDE_xSemaphoreGetMutexHolder, 155
INCLUDE_xTaskAbortDelay, 155
INCLUDE_xTaskDelayUntil, 155
INCLUDE_xTaskGetCurrentTaskHandle, 155
INCLUDE_xTaskGetHandle, 156
INCLUDE_xTaskGetSchedulerState, 156
INCLUDE_xTimerPendFunctionCall, 156
vPortSVCHandler, 156
xPortPendSVHandler, 156
xPortSysTickHandler, 156
function
CAN_Callback, 63

gamma
exp_torque_map_args, 78
GENERAL_VEH_INFO_MGROUP
uvfr_settings.h, 230
GENERAL_VEH_INFO_OFFSET
uvfr_settings.h, 230
generateHash
can.c, 269
GENERIC_ACK
uvfr_settings.h, 236
getKValue
driving_loop.c, 285
getSVCTaskID
uvfr_state_engine.h, 243
global_context
uvfr_utils.h, 256
gpio.c
MX_GPIO_Init, 295
gpio.h
MX_GPIO_Init, 157

hadc1
adc.c, 264
adc.h, 125
stm32f4xx_it.c, 337
hadc2
adc.c, 264
adc.h, 125
stm32f4xx_it.c, 337
HAL_ADC_ConvCpltCallback
main.c, 309
HAL_ADC_LevelOutOfWindowCallback
main.c, 309
HAL_ADC_MODULE_ENABLED
stm32f4xx_hal_conf.h, 200
HAL_ADC_MspDeInit
adc.c, 261
HAL_ADC_MspInit
adc.c, 261
HAL_CAN_ERROR_INVALID_CALLBACK
can.c, 268

HAL_CAN_MODULE_ENABLED
stm32f4xx_hal_conf.h, 200
HAL_CAN_MspDeInit
can.c, 270
HAL_CAN_MspInit
can.c, 270
HAL_CAN_RxFifo0MsgPendingCallback
can.c, 270
can.h, 129
HAL_CAN_RxFifo1MsgPendingCallback
can.c, 270
can.h, 129
HAL_CORTEX_MODULE_ENABLED
stm32f4xx_hal_conf.h, 200
HAL_DMA_MODULE_ENABLED
stm32f4xx_hal_conf.h, 200
HAL_EXTI_MODULE_ENABLED
stm32f4xx_hal_conf.h, 200
HAL_FLASH_MODULE_ENABLED
stm32f4xx_hal_conf.h, 201
HAL_GPIO_EXTI_Callback
main.c, 309
HAL_GPIO_MODULE_ENABLED
stm32f4xx_hal_conf.h, 201
HAL_InitTick
stm32f4xx_hal_timebase_tim.c, 331
HAL_MODULE_ENABLED
stm32f4xx_hal_conf.h, 201
HAL_MspInit
stm32f4xx_hal_msp.c, 330
HAL_PWR_MODULE_ENABLED
stm32f4xx_hal_conf.h, 201
HAL_RCC_MODULE_ENABLED
stm32f4xx_hal_conf.h, 201
HAL_ResumeTick
stm32f4xx_hal_timebase_tim.c, 331
HAL_SPI_MODULE_ENABLED
stm32f4xx_hal_conf.h, 201
HAL_SPI_MspDeInit
spi.c, 328
HAL_SPI_MspInit
spi.c, 329
HAL_SuspendTick
stm32f4xx_hal_timebase_tim.c, 332
HAL_TIM_Base_MspDeInit
tim.c, 348
HAL_TIM_Base_MspInit
tim.c, 349
HAL_TIM_MODULE_ENABLED
stm32f4xx_hal_conf.h, 202
HAL_TIM_PeriodElapsedCallback
main.c, 309
handle
uv_binary_semaphore_info, 95
uv_mutex_info, 102
uv_semaphore_info, 107
handleBMSVoltageMsg
bms.c, 266
handleCANbusError
can.c, 271
handleIncomingLaptopMsg
uvfr_settings.c, 354
HANDSHAKE
uvfr_settings.h, 236
HardFault_Handler
stm32f4xx_it.c, 336
stm32f4xx_it.h, 219
Hardware_Error
imd.h, 160
hardware_fault
motor_controller.h, 175
hcan2
can.c, 272
can.h, 130
stm32f4xx_it.c, 338
hdma_adc1
adc.c, 265
stm32f4xx_it.c, 338
helper_task_args, 78
journal_tx_args, 79
setting_tx_args, 79
uvfr_settings.c, 353
High_Battery_Voltage
imd.h, 160
HIGH_PRIORITY
State Engine API, 25
High_Uncertainty
imd.h, 160
HSE_STARTUP_TIMEOUT
stm32f4xx_hal_conf.h, 202
HSE_VALUE
stm32f4xx_hal_conf.h, 202
STM32F4xx_System_Private_Includes, 53
HSI_VALUE
stm32f4xx_hal_conf.h, 202
STM32F4xx_System_Private_Includes, 53
hspi1
spi.c, 329
spi.h, 195
htim1
stm32f4xx_hal_timebase_tim.c, 333
stm32f4xx_it.c, 338
htim3
tim.c, 349
tim.h, 222

IDLE_TASK_PRIORITY
State Engine API, 25
IGBT_temp_max_limit
motor_controller.h, 175
igbt_temperature
motor_controller.h, 176
IGBT_temperature_warning
motor_controller.h, 175
IMD
uvfr_utils.h, 252
imd.c

default_imd_settings, 302
IMD_Check_Battery_Voltage, 297
IMD_Check_Error_Flags, 297
IMD_Check_Isolation_Capacitances, 297
IMD_Check_Isolation_Resistances, 298
IMD_Check_Isolation_State, 298
IMD_Check_Max_Battery_Working_Voltage, 298
IMD_Check_Part_Name, 298
IMD_Check_Safety_Touch_Current, 299
IMD_Check_Safety_Touch_Energy, 299
IMD_Check_Serial_Number, 299
IMD_Check_Status_Bits, 299
IMD_Check_Temperature, 300
IMD_Check_Uptime, 300
IMD_Check_Version, 300
IMD_Check_Voltages_Vp_and_Vn, 300
IMD_error_flags_requested, 302
IMD_Expected_Part_Name, 302
IMD_Expected_Serial_Number, 302
IMD_Expected_Version, 303
IMD_High_Uncertainty, 303
IMD_Parse_Message, 301
IMD_Part_Name_0_Set, 303
IMD_Part_Name_1_Set, 303
IMD_Part_Name_2_Set, 303
IMD_Part_Name_3_Set, 304
IMD_Part_Name_Set, 304
IMD_Read_Part_Name, 304
IMD_Read_Serial_Number, 304
IMD_Read_Version, 304
IMD_Request_Status, 301
IMD_Serial_Number_0_Set, 305
IMD_Serial_Number_1_Set, 305
IMD_Serial_Number_2_Set, 305
IMD_Serial_Number_3_Set, 305
IMD_Serial_Number_Set, 305
IMD_Startup, 301
IMD_status_bits, 306
IMD_Temperature, 306
IMD_Version_0_Set, 306
IMD_Version_1_Set, 306
IMD_Version_2_Set, 306
IMD_Version_Set, 307
initIMD, 301
imd.h
battery_voltage, 161
Err_CH, 159
Err_clock, 159
Err_temp, 159
Err_Vexi, 159
Err_Vpwr, 159
Err_Vx1, 159
Err_Vx2, 159
Err_VxR, 159
Err_Watchdog, 159
Error_flags, 161
Exc_off, 160
Hardware_Error, 160
High_Battery_Voltage, 160
High_Uncertainty, 160
IMD_Check_Battery_Voltage, 161
IMD_Check_Error_Flags, 161
IMD_Check_Isolation_Capacitances, 161
IMD_Check_Isolation_Resistances, 162
IMD_Check_Isolation_State, 162
IMD_Check_Max_Battery_Working_Voltage, 162
IMD_Check_Part_Name, 162
IMD_Check_Safety_Touch_Current, 163
IMD_Check_Safety_Touch_Energy, 163
IMD_Check_Serial_Number, 163
IMD_Check_Status_Bits, 163
IMD_Check_Temperature, 164
IMD_Check_Uptime, 164
IMD_Check_Version, 164
IMD_Check_Voltages_Vp_and_Vn, 164
imd_error_flags, 159
imd_high_resolution_measurements, 159
imd_manufacturer_requests, 160
IMD_Parse_Message, 165
IMD_Request_Status, 165
IMD_Startup, 165
imd_status_bits, 160
imd_status_requests, 160
initIMD, 165
isolation_capacitances, 161
isolation_resistances, 161
isolation_state, 161
Isolation_status_bit0, 160
Isolation_status_bit1, 160
Low_Battery_Voltage, 160
Max_battery_working_voltage, 161
Part_name_0, 160
Part_name_1, 160
Part_name_2, 160
Part_name_3, 160
safety_touch_current, 161
safety_touch_energy, 161
Serial_number_0, 160
Serial_number_1, 160
Serial_number_2, 160
Serial_number_3, 160
Temperature, 161
Touch_energy_fault, 160
Uptime_counter, 160
uv_imd_settings, 159
Vb_hi_res, 159
Version_0, 160
Version_1, 160
Version_2, 160
Vexc_hi_res, 159
Vn_hi_res, 159
voltages_Vp_and_Vn, 161
Vp_hi_res, 159
Vpwr_hi_res, 159
IMD_ADDR
uvfr_settings.h, 230

IMD_CAN_ID_Rx
constants.h, 131
IMD_CAN_ID_Tx
constants.h, 131
IMD_Check_Battery_Voltage
imd.c, 297
imd.h, 161
IMD_Check_Error_Flags
imd.c, 297
imd.h, 161
IMD_Check_Isolation_Capacitances
imd.c, 297
imd.h, 161
IMD_Check_Isolation_Resistances
imd.c, 298
imd.h, 162
IMD_Check_Isolation_State
imd.c, 298
imd.h, 162
IMD_Check_Max_Battery_Working_Voltage
imd.c, 298
imd.h, 162
IMD_Check_Part_Name
imd.c, 298
imd.h, 162
IMD_Check_Safety_Touch_Current
imd.c, 299
imd.h, 163
IMD_Check_Safety_Touch_Energy
imd.c, 299
imd.h, 163
IMD_Check_Serial_Number
imd.c, 299
imd.h, 163
IMD_Check_Status_Bits
imd.c, 299
imd.h, 163
IMD_Check_Temperature
imd.c, 300
imd.h, 164
IMD_Check_Uptime
imd.c, 300
imd.h, 164
IMD_Check_Version
imd.c, 300
imd.h, 164
IMD_Check_Voltages_Vp_and_Vn
imd.c, 300
imd.h, 164
imd_error_flags
imd.h, 159
IMD_error_flags_requested
imd.c, 302
IMD_ERRORS
daq.h, 135
IMD_Expected_Part_Name
imd.c, 302
IMD_Expected_Serial_Number
imd.c, 302
IMD_Expected_Version
imd.c, 303
imd_high_resolution_measurements
imd.h, 159
IMD_High_Uncertainty
imd.c, 303
imd_manufacturer_requests
imd.h, 160
IMD_MGROUP
uvfr_settings.h, 231
IMD_OFFSET
uvfr_settings.h, 231
IMD_Parse_Message
imd.c, 301
imd.h, 165
IMD_Part_Name_0_Set
imd.c, 303
IMD_Part_Name_1_Set
imd.c, 303
IMD_Part_Name_2_Set
imd.c, 303
IMD_Part_Name_3_Set
imd.c, 304
IMD_Part_Name_Set
imd.c, 304
IMD_Read_Part_Name
imd.c, 304
IMD_Read_Serial_Number
imd.c, 304
IMD_Read_Version
imd.c, 304
IMD_Request_Status
imd.c, 301
imd.h, 165
IMD_Serial_Number_0_Set
imd.c, 305
IMD_Serial_Number_1_Set
imd.c, 305
IMD_Serial_Number_2_Set
imd.c, 305
IMD_Serial_Number_3_Set
imd.c, 305
IMD_Serial_Number_Set
imd.c, 305
imd_settings
uv_vehicle_settings, 116
IMD_Startup
imd.c, 301
imd.h, 165
IMD_STATUS
daq.h, 135
IMD_status_bits
imd.c, 306
imd_status_bits
imd.h, 160
imd_status_requests
imd.h, 160

IMD_Temperature
imd.c, 306
IMD_Version_0_Set
imd.c, 306
IMD_Version_1_Set
imd.c, 306
IMD_Version_2_Set
imd.c, 306
IMD_Version_Set
imd.c, 307
IMD_VOLTAGE
daq.h, 135
Implausible
driving_loop.h, 142
INCLUDE_eTaskGetState
FreeRTOSConfig.h, 153
INCLUDE_pcTaskGetTaskName
FreeRTOSConfig.h, 153
INCLUDE_uxTaskGetStackHighWaterMark
FreeRTOSConfig.h, 153
INCLUDE_uxTaskGetStackHighWaterMark2
FreeRTOSConfig.h, 153
INCLUDE_uxTaskPriorityGet
FreeRTOSConfig.h, 153
INCLUDE_vTaskCleanUpResources
FreeRTOSConfig.h, 154
INCLUDE_vTaskDelay
FreeRTOSConfig.h, 154
INCLUDE_vTaskDelayUntil
FreeRTOSConfig.h, 154
INCLUDE_vTaskDelete
FreeRTOSConfig.h, 154
INCLUDE_vTaskPrioritySet
FreeRTOSConfig.h, 154
INCLUDE_vTaskSuspend
FreeRTOSConfig.h, 154
INCLUDE_xEventGroupSetBitFromISR
FreeRTOSConfig.h, 155
INCLUDE_xQueueGetMutexHolder
FreeRTOSConfig.h, 155
INCLUDE_xSemaphoreGetMutexHolder
FreeRTOSConfig.h, 155
INCLUDE_xTaskAbortDelay
FreeRTOSConfig.h, 155
INCLUDE_xTaskDelayUntil
FreeRTOSConfig.h, 155
INCLUDE_xTaskGetCurrentTaskHandle
FreeRTOSConfig.h, 155
INCLUDE_xTaskGetHandle
FreeRTOSConfig.h, 156
INCLUDE_xTaskGetSchedulerState
FreeRTOSConfig.h, 156
INCLUDE_xTimerPendFunctionCall
FreeRTOSConfig.h, 156
INIT_CHECK_PERIOD
uvfr_utils.h, 247
init_info_queue
uv_init_task_args, 99
init_params
uv_internal_params, 101
init_settings
freertos.c, 293
init_task_handle
freertos.c, 294
uvfr_utils.c, 375
initADCTask
adc.c, 262
adc.h, 124
initDaqTask
daq.c, 277
daq.h, 136
initDrivingLoop
driving_loop.c, 285
driving_loop.h, 143
initialise_monitor_handles
syscalls.c, 343
initIMD
imd.c, 301
imd.h, 165
initOdometer
odometer.c, 317
odometer.h, 179
initPDU
pdu.c, 318
pdu.h, 183
initTempMonitor
temp_monitoring.c, 346
temp_monitoring.h, 220
INORDER
rb_tree.h, 190
insertCANMessageHandler
UVFR CANbus API, 50
insertParamToParamList
daq.c, 277
insertParamToRegister
daq.c, 278
insertRepair
rb_tree.c, 323
INSTRUCTION_CACHE_ENABLE
stm32f4xx_hal_conf.h, 202
integral_memory_max
motor_controller.h, 173
motor_controller_settings, 81
integral_time_constant
motor_controller.h, 173
motor_controller_settings, 81
intended_recipient
uv_task_msg_t, 114
internal_hardware_voltage_problem
motor_controller.h, 175
INVALID_SBLOCK
uvfr_settings.c, 352
is_accelerating
driving_loop.c, 290
is_laptop_connected
uvfr_settings.c, 364

isolation_capacitances
imd.h, 161
isolation_resistances
imd.h, 161
isolation_state
imd.h, 161
Isolation_status_bit0
imd.h, 160
Isolation_status_bit1
imd.h, 160
isPowerOfTwo
Utility Macros, 46
isPtrToFlash
uvfr_settings.h, 231

journal_tx_args
helper_task_args, 79

killEmAll
State Engine Internals, 33
killSelf
State Engine Internals, 33

laptop_CMD
uvfr_settings.h, 235
LAPTOP_TO_VCU_ID
uvfr_settings.c, 352
last_contact_with_laptop
uvfr_settings.c, 365
last_execution_time
uv_task_info, 109
last_task_end_times
State Engine, 13
last_task_start_times
State Engine, 13
leakage_inductance_ph_ph
motor_controller.h, 173
left
rbnode, 83
limp
uvfr_utils.h, 252
linear_speed_map
driving_loop.h, 143
linear_torque_map
driving_loop.h, 143
linear_torque_map_args, 79
driving_loop.h, 141
offset, 79
slope, 80
loggable_params
daq.h, 134
LOGIMAP_ERRORS
motor_controller.h, 174
LOGIMAP_IO
motor_controller.h, 174
Low_Battery_Voltage
imd.h, 160
LOW_PRIORITY
State Engine API, 25

LSE_STARTUP_TIMEOUT
stm32f4xx_hal_conf.h, 203
LSE_VALUE
stm32f4xx_hal_conf.h, 203
LSI_VALUE
stm32f4xx_hal_conf.h, 203

MAC_ADDR0
stm32f4xx_hal_conf.h, 203
MAC_ADDR1
stm32f4xx_hal_conf.h, 203
MAC_ADDR2
stm32f4xx_hal_conf.h, 204
MAC_ADDR3
stm32f4xx_hal_conf.h, 204
MAC_ADDR4
stm32f4xx_hal_conf.h, 204
MAC_ADDR5
stm32f4xx_hal_conf.h, 204
MAGIC_NUMBER
uvfr_settings.h, 231
main
main.c, 310
main.c
adc1_APPS1, 311
adc1_APPS2, 311
adc1_BPS1, 312
adc1_BPS2, 312
adc2_CoolantFlow, 312
adc2_CoolantTemp, 312
adc_buf1, 312
adc_buf2, 313
DEBUG_CAN_IN_MAIN, 308
Error_Handler, 308
HAL_ADC_ConvCpltCallback, 309
HAL_ADC_LevelOutOfWindowCallback, 309
HAL_GPIO_EXTI_Callback, 309
HAL_TIM_PeriodElapsedCallback, 309
main, 310
MX_FREERTOS_Init, 310
SystemClock_Config, 311
main.h
Blue_LED_GPIO_Port, 167
Blue_LED_Pin, 167
Error_Handler, 168
Orange_LED_GPIO_Port, 167
Orange_LED_Pin, 167
Red_LED_GPIO_Port, 167
Red_LED_Pin, 167
Start_Button_Input_EXTI_IRQn, 168
Start_Button_Input_GPIO_Port, 168
Start_Button_Input_Pin, 168
mains_voltage_max_limit
motor_controller.h, 175
mains_voltage_min_limit
motor_controller.h, 175
map_fn_params
drivingMode, 76
map_mode

driving_loop.h, 142
mapThrottleToTorque
driving_loop.c, 285
max_acc_pwr
drivingMode, 76
max_accum_current_5s
driving_loop_args, 72
max_apps1_value
driving_loop_args, 72
max_apps2_value
driving_loop_args, 72
max_apps_offset
driving_loop_args, 73
Max_battery_working_voltage
imd.h, 161
max_BPS_value
driving_loop_args, 73
MAX_CELL_TEMP
daq.h, 135
max_current
drivingMode, 77
max_imd_temperature
uv_imd_settings, 97
MAX_INIT_TIME
uvfr_utils.h, 247
MAX_LOGGABLE_PARAMS
daq.h, 136
max_motor_torque
drivingMode, 77
MAX_NUM_MANAGED_TASKS
State Engine, 10
max_svc_task_period
uv_os_settings, 103
max_task_period
uv_os_settings, 103
max_temp
bms_settings_t, 62
MC_CAN_ID_Rx
constants.h, 131
MC_CAN_ID_Tx
constants.h, 131
mc_CAN_timeout
motor_controller_settings, 81
MC_CURRENT
daq.h, 135
mc_default_settings
motor_controller.c, 316
motor_controller.h, 178
uvfr_settings.c, 365
MC_ERRORS
daq.h, 135
MC_POWER
driving_loop.h, 141
MC_Request_Data
motor_controller.c, 314
motor_controller.h, 176
mc_rfe_channel
uv19_pdu_settings, 93
MC_RPM
driving_loop.h, 141
MC_Set_Param
motor_controller.c, 314
mc_settings
uv_vehicle_settings, 116
MC_Startup
motor_controller.c, 314
motor_controller.h, 176
MC_TEMP
daq.h, 135
MC_Torque
driving_loop.h, 142
MC_VOLTAGE
daq.h, 135
MEDIUM_PRIORITY
State Engine API, 25
MemManage_Handler
stm32f4xx_it.c, 336
stm32f4xx_it.h, 219
message_size
uv_task_msg_t, 114
message_type
uv_task_msg_t, 114
meta_id
uv_scd_response, 106
meta_task_handle
daq_child_task, 64
setting_helper_task, 88
state_change_daemon_args, 89
uv_init_task_args, 99
min
rbtree, 85
min_apps1_value
driving_loop_args, 73
min_apps2_value
driving_loop_args, 73
min_apps_offset
driving_loop_args, 73
min_apps_value
driving_loop_args, 74
min_BPS_value
driving_loop_args, 74
MIN_CELL_TEMP
daq.h, 135
min_isolation_resistances
uv_imd_settings, 97
min_task_period
uv_os_settings, 103
minimum_daq_period
daq_loop_args, 67
minimum_magnetising_current
motor_controller.h, 173
MOTOR_ADDR
uvfr_settings.h, 231
motor_continuous_current
motor_controller.h, 173
MOTOR_CONTROLLER

uvfr_utils.h, 252
motor_controller.c
mc_default_settings, 316
MC_Request_Data, 314
MC_Set_Param, 314
MC_Startup, 314
MotorControllerErrorHandler_16bitLE, 314
MotorControllerSpinTest, 315
Parse_Bamocar_Response, 315
ProcessMotorControllerResponse, 315
motor_controller.h
AC_current_offset_fault, 175
accelerate_ramp, 173
ADC_measurement_problem, 175
ADC_sequencer_problem, 175
air_temperature, 176
auxiliary_voltage_min_limit, 175
bleed_resistor_overload, 175
bleeder_resistor_warning, 175
CAN_timeout_error, 175
check_ecode_ID, 175
clear_errors, 174
critical_AC_current, 175
CURRENT_ACTUAL, 172
current_derate_temperature, 176
current_feed_forward, 173
DC_bus_voltage, 172
dismantling_ramp, 173
ecode_timeout_error, 175
eprom_read_error, 175
feedback_signal_error, 175
feedback_signal_problem, 175
firmware_version, 174
FIRMWARE_VERSION_REGISTER, 170
hardware_fault, 175
IGBT_temp_max_limit, 175
igbt_temperature, 176
IGBT_temperature_warning, 175
integral_memory_max, 173
integral_time_constant, 173
internal_hardware_voltage_problem, 175
leakage_inductance_ph_ph, 173
LOGIMAP_ERRORS, 174
LOGIMAP_IO, 174
mains_voltage_max_limit, 175
mains_voltage_min_limit, 175
mc_default_settings, 178
MC_Request_Data, 176
MC_Startup, 176
minimum_magnetising_current, 173
motor_continuous_current, 173
motor_controller_current_parameters, 171
motor_controller_errors_warnings, 175
motor_controller_io, 172
motor_controller_limp_mode, 172
motor_controller_measurements, 172
motor_controller_motor_constants, 172
motor_controller_PI_values, 173
motor_controller_repeating_time, 173
motor_controller_settings, 171
motor_controller_speed_parameters, 174
motor_controller_startup, 174
motor_controller_status, 174
motor_controller_status_information_errors_warnings, 175
motor_controller_temperatures, 175
motor_ke_constant, 173
motor_kt_constant, 173
motor_magnetising_inductance, 173
motor_max_current, 173
motor_pole_number, 173
motor_temp_max_limit, 175
motor_temperature, 176
motor_temperature_switch_off_point, 173
motor_temperature_warning, 175
MotorControllerSpinTest, 177
N_actual, 174
N_cmd, 174
N_error, 174
N_lim, 172
N_lim_minus, 172
N_lim_plus, 172
N_set, 174
N_set_cmd, 171
nominal_magnitizing_current, 173
nominal_motor_frequency, 173
nominal_motor_voltage, 173
none, 174
one_hundred_ms, 174
parameter_conflict_detected, 175
Parse_Bamocar_Response, 177
POS_ACTUAL, 174
power_factor, 173
ProcessMotorControllerResponse, 177
proportional_gain, 173
proportional_gain_2, 173
race_away_detected, 175
ramp_set_current, 173
rated_motor_speed, 173
recuperation_ramp, 173
rotate_field_enable_not_present_norun, 175
rotate_field_enable_not_present_run, 175
rotor_resistance, 173
SERIAL_NUMBER_REGISTER, 171
special_CPU_fault, 175
speed_actual_resolution_limit, 175
stator_leakage_inductance, 173
stator_resistance_ph_ph, 173
temp_sensor_pt1, 176
temp_sensor_pt2, 176
temp_sensor_pt3, 176
temp_sensor_pt4, 176
time_constant_rotor, 173
time_constant_stator, 173
todo6969, 172
tripzone_glitch_detected, 175

uv_CAN_msg, 171
Vout_saturation_max_limit, 175
warning_5, 175
warning_9, 175
watchdog_reset, 175
motor_controller_current_parameters
motor_controller.h, 171
motor_controller_errors_warnings
motor_controller.h, 175
motor_controller_io
motor_controller.h, 172
motor_controller_limp_mode
motor_controller.h, 172
motor_controller_measurements
motor_controller.h, 172
motor_controller_motor_constants
motor_controller.h, 172
motor_controller_PI_values
motor_controller.h, 173
motor_controller_repeating_time
motor_controller.h, 173
motor_controller_settings, 80
can_id_rx, 80
can_id_tx, 80
integral_memory_max, 81
integral_time_constant, 81
mc_CAN_timeout, 81
motor_controller.h, 171
proportional_gain, 81
uvfr_settings.h, 235
motor_controller_speed_parameters
motor_controller.h, 174
motor_controller_startup
motor_controller.h, 174
motor_controller_status
motor_controller.h, 174
motor_controller_status_information_errors_warnings
motor_controller.h, 175
motor_controller_temperatures
motor_controller.h, 175
MOTOR_CURRENT
daq.h, 135
motor_ke_constant
motor_controller.h, 173
motor_kt_constant
motor_controller.h, 173
motor_magnetising_inductance
motor_controller.h, 173
motor_max_current
motor_controller.h, 173
MOTOR_MGROUP
uvfr_settings.h, 231
MOTOR_OFFSET
uvfr_settings.h, 232
motor_pole_number
motor_controller.h, 173
MOTOR_RPM
daq.h, 135
MOTOR_TEMP
daq.h, 135
motor_temp_max_limit
motor_controller.h, 175
motor_temperature
motor_controller.h, 176
motor_temperature_switch_off_point
motor_controller.h, 173
motor_temperature_warning
motor_controller.h, 175
MotorControllerErrorHandler_16bitLE
motor_controller.c, 314
MotorControllerSpinTest
motor_controller.c, 315
motor_controller.h, 177
msg_contents
uv_task_msg_t, 114
msg_id
uv_CAN_msg, 96
mutex
access_control_info, 61
MX_ADC1_Init
adc.c, 262
adc.h, 124
MX_ADC2_Init
adc.c, 262
adc.h, 124
MX_CAN2_Init
can.c, 271
can.h, 130
MX_DMA_Init
dma.c, 283
dma.h, 139
MX_FREERTOS_Init
freertos.c, 291
main.c, 310
MX_GPIO_Init
gpio.c, 295
gpio.h, 157
MX_SPI1_Init
spi.c, 329
spi.h, 195
MX_TIM3_Init
tim.c, 349
tim.h, 222

N_actual
motor_controller.h, 174
N_cmd
motor_controller.h, 174
N_error
motor_controller.h, 174
N_lim
motor_controller.h, 172
N_lim_minus
motor_controller.h, 172
N_lim_plus
motor_controller.h, 172
N_set

motor_controller.h, 174
N_set_cmd
motor_controller.h, 171
nc_throttling
State Engine, 13
nchar
uv_init_task_response, 100
next
CAN_Callback, 63
daq_param_list_node, 68
next_task
daq_child_task, 64
nil
rbtree, 85
NMI_Handler
stm32f4xx_it.c, 336
stm32f4xx_it.h, 219
nominal_magnitizing_current
motor_controller.h, 173
nominal_motor_frequency
motor_controller.h, 173
nominal_motor_voltage
motor_controller.h, 173
none
motor_controller.h, 174
normal
uvfr_utils.h, 252
nuke_hash_table
can.c, 271
can.h, 130
nukeSettings
uvfr_settings.c, 354
uvfr_settings.h, 236
num_driving_modes
driving_loop_args, 74

odometer.c
initOdometer, 317
odometerTask, 317
odometer.h
initOdometer, 179
odometerTask, 179
uv_persistant_data_frame, 179
odometerTask
odometer.c, 317
odometer.h, 179
offset
exp_torque_map_args, 78
linear_torque_map_args, 79
oled.h
amogus, 180
oled_config, 180
oled_Write, 180
oled_Write_Cmd, 180
oled_Write_Data, 180
refresh_OLED, 181
wait, 181
oled_config
oled.h, 180
oled_Write
oled.h, 180
oled_Write_Cmd
oled.h, 180
oled_Write_Data
oled.h, 180
one_hundred_ms
motor_controller.h, 174
Orange_LED_GPIO_Port
main.h, 167
Orange_LED_Pin
main.h, 167
os_flag
State Engine API, 25
os_flags
uv_os_settings, 103
os_settings
State Engine, 13
uv_vehicle_settings, 117
OS_SETTINGS_ADDR
uvfr_settings.h, 232
OS_SETTINGS_MGROUP
uvfr_settings.h, 232
OS_SETTINGS_OFFSET
uvfr_settings.h, 232

p_status, 81
activation_time, 82
peripheral_status, 82
uvfr_utils.h, 249
param
daq_datapoint, 65
daq_param_list_node, 68
param_bank
daq.c, 280
param_list
daq_child_task, 64
param_ptrs
daq.c, 280
parameter_conflict_detected
motor_controller.h, 175
parent
rbnode, 83
uv_task_info, 109
parent_msg_queue
task_management_info, 89
Parse_Bamocar_Response
motor_controller.c, 315
motor_controller.h, 177
Part_name_0
imd.h, 160
Part_name_1
imd.h, 160
Part_name_2
imd.h, 160
Part_name_3
imd.h, 160
PDU
uvfr_utils.h, 252

pdu.c
default_pdu_settings, 321
initPDU, 318
PDU_disable_brake_light, 318
PDU_disable_coolant_pump, 319
PDU_disable_cooling_fans, 319
PDU_disable_motor_controller, 319
PDU_disable_shutdown_circuit, 319
PDU_enable_brake_light, 319
PDU_enable_coolant_pump, 320
PDU_enable_cooling_fans, 320
PDU_enable_motor_controller, 320
PDU_enable_shutdown_circuit, 320
PDU_speaker_chirp, 320
pdu.h
disable_brake_light_msg, 183
disable_coolant_pump_msg, 182
disable_left_cooling_fan_msg, 182
disable_motor_controller_msg, 183
disable_right_cooling_fan_msg, 182
disable_shutdown_circuit_msg, 183
disable_speaker_msg, 183
enable_brake_light_msg, 183
enable_coolant_pump_msg, 182
enable_left_cooling_fan_msg, 182
enable_motor_controller_msg, 183
enable_right_cooling_fan_msg, 182
enable_shutdown_circuit_msg, 183
enable_speaker_msg, 183
initPDU, 183
PDU_disable_brake_light, 183
PDU_disable_coolant_pump, 183
PDU_disable_cooling_fans, 183
PDU_disable_motor_controller, 184
PDU_disable_shutdown_circuit, 184
PDU_enable_brake_light, 184
PDU_enable_coolant_pump, 184
PDU_enable_cooling_fans, 184
PDU_enable_motor_controller, 185
PDU_enable_shutdown_circuit, 185
pdu_messages_20A, 182
pdu_messages_5A, 182
PDU_speaker_chirp, 185
uv19_pdu_settings, 182
PDU_ADDR
uvfr_settings.h, 232
PDU_CAN_ID_Tx
constants.h, 131
PDU_disable_brake_light
pdu.c, 318
pdu.h, 183
PDU_disable_coolant_pump
pdu.c, 319
pdu.h, 183
PDU_disable_cooling_fans
pdu.c, 319
pdu.h, 183
PDU_disable_motor_controller
pdu.c, 319
pdu.h, 184
PDU_disable_shutdown_circuit
pdu.c, 319
pdu.h, 184
PDU_enable_brake_light
pdu.c, 319
pdu.h, 184
PDU_enable_coolant_pump
pdu.c, 320
pdu.h, 184
PDU_enable_cooling_fans
pdu.c, 320
pdu.h, 184
PDU_enable_motor_controller
pdu.c, 320
pdu.h, 185
PDU_enable_shutdown_circuit
pdu.c, 320
pdu.h, 185
pdu_messages_20A
pdu.h, 182
pdu_messages_5A
pdu.h, 182
PDU_MGROUP
uvfr_settings.h, 232
PDU_OFFSET
uvfr_settings.h, 233
PDU_rx_addr
uv19_pdu_settings, 94
pdu_settings
uv_vehicle_settings, 117
PDU_speaker_chirp
pdu.c, 320
pdu.h, 185
PDU_tx_addr
uv19_pdu_settings, 94
performSafetyChecks
driving_loop.c, 286
period
daq_child_task, 64
daq_datapoint, 66
driving_loop_args, 74
peripheral_status
p_status, 82
uv_internal_params, 101
PERSISTENT_DATA_MGROUP
uvfr_settings.h, 233
PHY_AUTONEGO_COMPLETE
stm32f4xx_hal_conf.h, 204
PHY_AUTONEGOTIATION
stm32f4xx_hal_conf.h, 204
PHY_BCR
stm32f4xx_hal_conf.h, 205
PHY_BSR
stm32f4xx_hal_conf.h, 205
PHY_CONFIG_DELAY
stm32f4xx_hal_conf.h, 205

PHY_DUPLEX_STATUS
stm32f4xx_hal_conf.h, 205
PHY_FULLDUPLEX_100M
stm32f4xx_hal_conf.h, 205
PHY_FULLDUPLEX_10M
stm32f4xx_hal_conf.h, 206
PHY_HALFDUPLEX_100M
stm32f4xx_hal_conf.h, 206
PHY_HALFDUPLEX_10M
stm32f4xx_hal_conf.h, 206
PHY_ISOLATE
stm32f4xx_hal_conf.h, 206
PHY_JABBER_DETECTION
stm32f4xx_hal_conf.h, 206
PHY_LINKED_STATUS
stm32f4xx_hal_conf.h, 207
PHY_LOOPBACK
stm32f4xx_hal_conf.h, 207
PHY_POWERDOWN
stm32f4xx_hal_conf.h, 207
PHY_READ_TO
stm32f4xx_hal_conf.h, 207
PHY_RESET
stm32f4xx_hal_conf.h, 207
PHY_RESET_DELAY
stm32f4xx_hal_conf.h, 208
PHY_RESTART_AUTONEGOTIATION
stm32f4xx_hal_conf.h, 208
PHY_SPEED_STATUS
stm32f4xx_hal_conf.h, 208
PHY_SR
stm32f4xx_hal_conf.h, 208
PHY_WRITE_TO
stm32f4xx_hal_conf.h, 208
Plausible
driving_loop.h, 142
POS_ACTUAL
motor_controller.h, 174
POST_CHECKSUM
uvfr_settings.c, 353
POSTORDER
rb_tree.h, 190
POWER_DERATE_FACTOR
daq.h, 135
power_factor
motor_controller.h, 173
PRE_CHECKSUM
uvfr_settings.c, 353
PREFETCH_ENABLE
stm32f4xx_hal_conf.h, 209
PREORDER
rb_tree.h, 190
previous_state
State Engine, 14
print
rb_tree.c, 323
rbtree, 86
proccessSCDMsg
State Engine Internals, 34
processADCBuffer
adc.c, 263
adc.h, 124
ProcessMotorControllerResponse
motor_controller.c, 315
motor_controller.h, 177
PROGRAMMING
State Engine API, 27
proportional_gain
motor_controller.h, 173
motor_controller_settings, 81
proportional_gain_2
motor_controller.h, 173
ptask_handle
uvfr_settings.c, 365

race_away_detected
motor_controller.h, 175
ramp_set_current
motor_controller.h, 173
rated_motor_speed
motor_controller.h, 173
RB_APPLY
rb_tree.h, 187
RB_DUP
rb_tree.h, 187
RB_FIRST
rb_tree.h, 187
RB_ISEMPTY
rb_tree.h, 187
RB_MIN
rb_tree.h, 187
RB_MINIMAL
rb_tree.h, 188
RB_NIL
rb_tree.h, 188
RB_ROOT
rb_tree.h, 188
rb_tree.c
checkBlackHeight, 322
checkOrder, 322
deleteRepair, 322
destroyAllNodes, 323
insertRepair, 323
print, 323
rbApplyNode, 323
rbCheckBlackHeight, 324
rbCheckOrder, 324
rbCreate, 324
rbDelete, 325
rbDestroy, 325
rbFind, 325
rbInsert, 326
rbPrint, 326
rbSuccessor, 327
rotateLeft, 327
rotateRight, 327
rb_tree.h

BLACK, 186
INORDER, 190
POSTORDER, 190
PREORDER, 190
RB_APPLY, 187
RB_DUP, 187
RB_FIRST, 187
RB_ISEMPTY, 187
RB_MIN, 187
RB_MINIMAL, 188
RB_NIL, 188
RB_ROOT, 188
rbApplyNode, 190
rbCheckBlackHeight, 190
rbCheckOrder, 191
rbCreate, 191
rbDelete, 192
rbDestroy, 192
rbFind, 192
rbInsert, 193
rbnode, 188
rbPrint, 193
rbSuccessor, 194
rbtraversal, 189
RED, 188
rbApplyNode
rb_tree.c, 323
rb_tree.h, 190
rbCheckBlackHeight
rb_tree.c, 324
rb_tree.h, 190
rbCheckOrder
rb_tree.c, 324
rb_tree.h, 191
rbCreate
rb_tree.c, 324
rb_tree.h, 191
rbDelete
rb_tree.c, 325
rb_tree.h, 192
rbDestroy
rb_tree.c, 325
rb_tree.h, 192
rbFind
rb_tree.c, 325
rb_tree.h, 192
rbInsert
rb_tree.c, 326
rb_tree.h, 193
rbnode, 82
color, 83
data, 83
left, 83
parent, 83
rb_tree.h, 188
right, 84
rbPrint
rb_tree.c, 326
rb_tree.h, 193
rbSuccessor
rb_tree.c, 327
rb_tree.h, 194
rbtraversal
rb_tree.h, 189
rbtree, 84
compare, 85
count, 85
destroy, 85
min, 85
nil, 85
print, 86
root, 86
REALTIME_PRIORITY
State Engine API, 25
recuperation_ramp
motor_controller.h, 173
RED
rb_tree.h, 188
Red_LED_GPIO_Port
main.h, 167
Red_LED_Pin
main.h, 167
refresh_OLED
oled.h, 181
regen_rpm_cutoff
driving_loop_args, 74
REQUEST_ALL_JOURNAL_ENTRIES
uvfr_settings.h, 236
REQUEST_ALL_SETTINGS
uvfr_settings.h, 236
REQUEST_JOURNAL_ENTRIES_BY_TIME
uvfr_settings.h, 236
REQUEST_SPECIFIC_SETTING
uvfr_settings.h, 236
REQUEST_VCU_FIRMWARE_VERSION
uvfr_settings.h, 236
REQUEST_VCU_STATUS
uvfr_settings.h, 236
reset_handle
uvfr_utils.c, 375
response_val
uv_scd_response, 106
right
rbnode, 84
root
rbtree, 86
rotate_field_enable_not_present_norun
motor_controller.h, 175
rotate_field_enable_not_present_run
motor_controller.h, 175
rotateLeft
rb_tree.c, 327
rotateRight
rb_tree.c, 327
rotor_resistance
motor_controller.h, 173

Rx_msg_queue
can.c, 272
RxData
constants.c, 273
constants.h, 131
RxHeader
constants.c, 273
constants.h, 131

s_curve_speed_map
driving_loop.h, 143
s_curve_torque_map
driving_loop.h, 143
s_curve_torque_map_args, 86
a, 87
b, 87
c, 87
driving_loop.h, 142
safePtrRead
Utility Macros, 46
safePtrWrite
Utility Macros, 46
safety_touch_current
imd.h, 161
safety_touch_energy
imd.h, 161
SAVE_AND_APPLY_NEW_SETTINGS
uvfr_settings.h, 236
SBLOCK_CRC_REGION_OFFSET
uvfr_settings.h, 233
SBLOCK_CSR_OFFSET
uvfr_settings.h, 233
sblock_origin
tx_all_settings_args, 91
SCD_active
State Engine, 14
scd_handle_ptr
State Engine, 14
sdc_channel
uv19_pdu_settings, 94
semaphore
access_control_info, 61
sendAllParamsFromList
daq.c, 278
sendAllSettingsWorker
uvfr_settings.c, 354
sendDaqMsg
daq.c, 278
sender
uv_task_msg_t, 114
sendJournalWorker
uvfr_settings.c, 355
sendTorqueToMotorController
driving_loop.c, 286
Serial_number_0
imd.h, 160
Serial_number_1
imd.h, 160
Serial_number_2
imd.h, 160
Serial_number_3
imd.h, 160
SERIAL_NUMBER_REGISTER
motor_controller.h, 171
serializeBigE16
Utility Macros, 46
serializeBigE32
Utility Macros, 47
serializeSmallE16
Utility Macros, 47
serializeSmallE32
Utility Macros, 47
SET_SPECIFIC_PARAM
uvfr_settings.h, 236
setBits
Utility Macros, 47
SETTING_BRANCH_SIZE
uvfr_settings.h, 233
setting_helper_task, 87
args, 88
meta_task_handle, 88
status, 88
uvfr_settings.c, 353
setting_tx_args
helper_task_args, 79
settingCopy
uvfr_settings.c, 355
settings_queue
uvfr_settings.c, 365
setup_extern_devices
uvfr_utils.c, 371
setupDefaultSettings
uvfr_settings.c, 355
uvfr_settings.h, 236
size
daq_param_list_node, 69
SIZE_OF_MGROUP
uvfr_settings.h, 233
SIZEOF_SBLOCK
uvfr_settings.h, 234
SIZEOF_USER_FLASH
uvfr_settings.h, 234
slope
linear_torque_map_args, 80
special_CPU_fault
motor_controller.h, 175
specific_args
uv_init_task_args, 99
speed_actual_resolution_limit
motor_controller.h, 175
spi.c
HAL_SPI_MspDeInit, 328
HAL_SPI_MspInit, 329
hspi1, 329
MX_SPI1_Init, 329
spi.h
hspi1, 195

MX_SPI1_Init, 195
SRC_UVFR_SETTINGS_C_
uvfr_settings.c, 353
stack_size
uv_task_info, 109
Start_Button_Input_EXTI_IRQn
main.h, 168
Start_Button_Input_GPIO_Port
main.h, 168
Start_Button_Input_Pin
main.h, 168
START_OF_USER_FLASH
uvfr_settings.h, 234
start_time
tx_journal_args, 92
StartADCTask
adc.c, 263
startDaqSubTasks
daq.c, 278
StartDefaultTask
freertos.c, 292
StartDrivingLoop
driving_loop.c, 287
driving_loop.h, 143
State Engine, 9
_next_svc_task_id, 12
_next_task_id, 12
_task_register, 12
compareTaskByName, 11
default_os_settings, 12
error_bitfield, 13
last_task_end_times, 13
last_task_start_times, 13
MAX_NUM_MANAGED_TASKS, 10
nc_throttling, 13
os_settings, 13
previous_state, 14
SCD_active, 14
scd_handle_ptr, 14
state_change_daemon_args, 10
state_change_queue, 14
svc_task_manager, 14
task_manager, 15
task_name_lut, 15
task_name_tree, 15
task_tardiness, 15
throttle_daq, 15
uvSVCTaskManager, 11
uvTaskPeriodEnd, 11
vehicle_state, 15
State Engine API, 16
ABOVE_NORMAL, 25
BELOW_NORMAL, 25
changeVehicleState, 27
HIGH_PRIORITY, 25
IDLE_TASK_PRIORITY, 25
LOW_PRIORITY, 25
MEDIUM_PRIORITY, 25
os_flag, 25
PROGRAMMING, 27
REALTIME_PRIORITY, 25
task_management_info, 23
task_priority, 23, 25
task_status_block, 23
UV_BOOT, 27
UV_COULDNT_DELETE, 26
UV_COULDNT_SUSPEND, 26
UV_DRIVING, 27
UV_ERROR_STATE, 27
UV_HALT, 27
UV_INIT, 27
UV_KILL_CMD, 26
UV_LAUNCH_CONTROL, 27
UV_NO_CMD, 26
UV_OS_ATTEMPT_RESTART_NC_TASK, 25
UV_OS_ENABLE_NONCRIT_TASK_THROTTLE, 25
UV_OS_LOG_MEM, 25
UV_OS_LOG_TASK_END_TIME, 25
uv_os_settings, 24
UV_READY, 27
uv_scd_response, 24
uv_scd_response_e, 25
UV_SUCCESSFUL_DELETION, 26
UV_SUCCESSFUL_SUSPENSION, 26
UV_SUSPEND_CMD, 26
UV_SUSPENDED, 27
UV_TASK_AWAITING_DELETION, 18
uv_task_cmd, 24
uv_task_cmd_e, 26
UV_TASK_DEADLINE_FIRM, 18
UV_TASK_DEADLINE_HARD, 18
UV_TASK_DEADLINE_MASK, 18
UV_TASK_DEADLINE_NOT_ENFORCED, 19
UV_TASK_DEFER_DELETION, 19
UV_TASK_DELAYING, 19
UV_TASK_DELETED, 26
UV_TASK_DORMANT_SVC, 19
UV_TASK_ERR_IN_CHILD, 19
UV_TASK_GENERIC_SVC, 19
uv_task_info, 24
UV_TASK_IS_CHILD, 20
UV_TASK_IS_ORPHAN, 20
UV_TASK_IS_PARENT, 20
UV_TASK_LOG_MEM_USAGE, 20
UV_TASK_LOG_START_STOP_TIME, 20
UV_TASK_MANAGER_MASK, 20
UV_TASK_MISSION_CRITICAL, 21
UV_TASK_NOT_STARTED, 26
UV_TASK_PERIODIC_SVC, 21
UV_TASK_PRIO_INCREMENTATION, 21
UV_TASK_RUNNING, 26
UV_TASK_SCD_IGNORE, 21
UV_TASK_START_CMD, 26
uv_task_state_t, 26
uv_task_status, 24

UV_TASK_SUSPENDED, 26
UV_TASK_VEHICLE_APPLICATION, 21
UV_UNSAFE_STATE, 26
uv_vehicle_state, 24
uv_vehicle_state_t, 27
uvCreateTask, 28
uvDeInitStateEngine, 28
uvInitStateEngine, 28
uvStartStateMachine, 29
uvTaskDelay, 21
uvTaskDelayUntil, 22
uvTaskIsDelaying, 22
uvTaskPeriodEnd, 29
uvTaskResetDelayBit, 22
uvTaskResetDeletionBit, 22
uvTaskSetDelayBit, 23
uvTaskSetDeletionBit, 23
State Engine Internals, 30
__uvPanic, 31
_stateChangeDaemon, 31
_uvValidateSpecificTask, 33
addTaskToTaskRegister, 33
killEmAll, 33
killSelf, 33
proccessSCDMsg, 34
suspendSelf, 34
uvAbortTaskDeletion, 35
uvCreateServiceTask, 35
uvDeleteSVCTask, 35
uvDeleteTask, 36
uvGetTaskFromName, 36
uvGetTaskFromRTOSHandle, 36
uvKillTaskViolently, 37
uvLateTaskHandler, 37
uvRestartSVCTask, 37
uvScheduleTaskDeletion, 38
uvSendTaskStatusReport, 38
uvStartSVCTask, 38
uvStartTask, 38
uvSuspendSVCTask, 39
uvSuspendTask, 39
uvTaskCrashHandler, 40
uvTaskManager, 40
uvThrottleNonCritTasks, 41
uvValidateManagedTasks, 41
state_change_daemon_args, 88
meta_task_handle, 89
State Engine, 10
state_change_queue
State Engine, 14
stator_leakage_inductance
motor_controller.h, 173
stator_resistance_ph_ph
motor_controller.h, 173
status
setting_helper_task, 88
uv_init_task_response, 100
STM32_F407
uvfr_global_config.h, 223
STM32_H7xx
uvfr_global_config.h, 223
stm32f4xx_hal_conf.h
assert_param, 198
DATA_CACHE_ENABLE, 199
DP83848_PHY_ADDRESS, 199
ETH_RX_BUF_SIZE, 199
ETH_RXBUFNB, 199
ETH_TX_BUF_SIZE, 199
ETH_TXBUFNB, 199
EXTERNAL_CLOCK_VALUE, 200
HAL_ADC_MODULE_ENABLED, 200
HAL_CAN_MODULE_ENABLED, 200
HAL_CORTEX_MODULE_ENABLED, 200
HAL_DMA_MODULE_ENABLED, 200
HAL_EXTI_MODULE_ENABLED, 200
HAL_FLASH_MODULE_ENABLED, 201
HAL_GPIO_MODULE_ENABLED, 201
HAL_MODULE_ENABLED, 201
HAL_PWR_MODULE_ENABLED, 201
HAL_RCC_MODULE_ENABLED, 201
HAL_SPI_MODULE_ENABLED, 201
HAL_TIM_MODULE_ENABLED, 202
HSE_STARTUP_TIMEOUT, 202
HSE_VALUE, 202
HSI_VALUE, 202
INSTRUCTION_CACHE_ENABLE, 202
LSE_STARTUP_TIMEOUT, 203
LSE_VALUE, 203
LSI_VALUE, 203
MAC_ADDR0, 203
MAC_ADDR1, 203
MAC_ADDR2, 204
MAC_ADDR3, 204
MAC_ADDR4, 204
MAC_ADDR5, 204
PHY_AUTONEGO_COMPLETE, 204
PHY_AUTONEGOTIATION, 204
PHY_BCR, 205
PHY_BSR, 205
PHY_CONFIG_DELAY, 205
PHY_DUPLEX_STATUS, 205
PHY_FULLDUPLEX_100M, 205
PHY_FULLDUPLEX_10M, 206
PHY_HALFDUPLEX_100M, 206
PHY_HALFDUPLEX_10M, 206
PHY_ISOLATE, 206
PHY_JABBER_DETECTION, 206
PHY_LINKED_STATUS, 207
PHY_LOOPBACK, 207
PHY_POWERDOWN, 207
PHY_READ_TO, 207
PHY_RESET, 207
PHY_RESET_DELAY, 208
PHY_RESTART_AUTONEGOTIATION, 208
PHY_SPEED_STATUS, 208
PHY_SR, 208

PHY_WRITE_TO, 208
PREFETCH_ENABLE, 209
TICK_INT_PRIORITY, 209
USE_HAL_ADC_REGISTER_CALLBACKS, 209
USE_HAL_CAN_REGISTER_CALLBACKS, 209
USE_HAL_CEC_REGISTER_CALLBACKS, 209
USE_HAL_CRYP_REGISTER_CALLBACKS, 209
USE_HAL_DAC_REGISTER_CALLBACKS, 210
USE_HAL_DCMI_REGISTER_CALLBACKS, 210
USE_HAL_DFSDM_REGISTER_CALLBACKS, 210
USE_HAL_DMA2D_REGISTER_CALLBACKS, 210
USE_HAL_DSI_REGISTER_CALLBACKS, 210
USE_HAL_ETH_REGISTER_CALLBACKS, 210
USE_HAL_FMPI2C_REGISTER_CALLBACKS, 211
USE_HAL_FMPSMBUS_REGISTER_CALLBACKS, 211
USE_HAL_HASH_REGISTER_CALLBACKS, 211
USE_HAL_HCD_REGISTER_CALLBACKS, 211
USE_HAL_I2C_REGISTER_CALLBACKS, 211
USE_HAL_I2S_REGISTER_CALLBACKS, 211
USE_HAL_IRDA_REGISTER_CALLBACKS, 212
USE_HAL_LPTIM_REGISTER_CALLBACKS, 212
USE_HAL_LTDC_REGISTER_CALLBACKS, 212
USE_HAL_MMC_REGISTER_CALLBACKS, 212
USE_HAL_NAND_REGISTER_CALLBACKS, 212
USE_HAL_NOR_REGISTER_CALLBACKS, 212
USE_HAL_PCCARD_REGISTER_CALLBACKS, 213
USE_HAL_PCD_REGISTER_CALLBACKS, 213
USE_HAL_QSPI_REGISTER_CALLBACKS, 213
USE_HAL_RNG_REGISTER_CALLBACKS, 213
USE_HAL_RTC_REGISTER_CALLBACKS, 213
USE_HAL_SAI_REGISTER_CALLBACKS, 213
USE_HAL_SD_REGISTER_CALLBACKS, 214
USE_HAL_SDRAM_REGISTER_CALLBACKS, 214
USE_HAL_SMARTCARD_REGISTER_CALLBACKS, 214
USE_HAL_SMBUS_REGISTER_CALLBACKS, 214
USE_HAL_SPDIFRX_REGISTER_CALLBACKS, 214
USE_HAL_SPI_REGISTER_CALLBACKS, 214
USE_HAL_SRAM_REGISTER_CALLBACKS, 215
USE_HAL_TIM_REGISTER_CALLBACKS, 215
USE_HAL_UART_REGISTER_CALLBACKS, 215
USE_HAL_USART_REGISTER_CALLBACKS, 215
USE_HAL_WWDG_REGISTER_CALLBACKS, 215
USE_RTOS, 215
USE_SPI_CRC, 216
VDD_VALUE, 216
stm32f4xx_hal_msp.c
HAL_MspInit, 330
stm32f4xx_hal_timebase_tim.c
HAL_InitTick, 331
HAL_ResumeTick, 331
HAL_SuspendTick, 332
htim1, 333
stm32f4xx_it.c
ADC_IRQHandler, 334
BusFault_Handler, 334
CAN2_RX0_IRQHandler, 334
CAN2_RX1_IRQHandler, 335
CAN2_TX_IRQHandler, 335
DebugMon_Handler, 335
DMA2_Stream0_IRQHandler, 335
EXTI0_IRQHandler, 336
hadc1, 337
hadc2, 337
HardFault_Handler, 336
hcan2, 338
hdma_adc1, 338
htim1, 338
MemManage_Handler, 336
NMI_Handler, 336
TIM1_UP_TIM10_IRQHandler, 337
UsageFault_Handler, 337
stm32f4xx_it.h
ADC_IRQHandler, 217
BusFault_Handler, 217
CAN2_RX0_IRQHandler, 217
CAN2_RX1_IRQHandler, 217
CAN2_TX_IRQHandler, 218
DebugMon_Handler, 218
DMA2_Stream0_IRQHandler, 218
EXTI0_IRQHandler, 218
HardFault_Handler, 219
MemManage_Handler, 219
NMI_Handler, 219
TIM1_UP_TIM10_IRQHandler, 219
UsageFault_Handler, 220
Stm32f4xx_system, 52
STM32F4xx_System_Private_Defines, 55
STM32F4xx_System_Private_FunctionPrototypes, 58
STM32F4xx_System_Private_Functions, 59
SystemCoreClockUpdate, 59
SystemInit, 59
STM32F4xx_System_Private_Includes, 53
HSE_VALUE, 53
HSI_VALUE, 53
STM32F4xx_System_Private_Macros, 56
STM32F4xx_System_Private_TypesDefinitions, 54
STM32F4xx_System_Private_Variables, 57
AHBPrescTable, 57
APBPrescTable, 57
SystemCoreClock, 57
stopDaqSubTasks
daq.c, 279
SUS_DAMPER_FL
daq.h, 135
SUS_DAMPER_RL

daq.h, 135
SUS_DAMPER_RR
daq.h, 135
SUS_DUMPER_FR
daq.h, 135
suspendSelf
State Engine Internals, 34
suspension_states
uv_task_info, 109
svc_task_manager
State Engine, 14
svc_task_manager_period
uv_os_settings, 104
SVC_TASK_MAX_CHECKIN_PERIOD
uvfr_state_engine.h, 242
SYS_DATA_OFFSET
uvfr_settings.h, 234
syscalls.c
__attribute__, 339
__io_getchar, 340
__io_putchar, 340
_close, 340
_execve, 340
_exit, 340
_fork, 341
_fstat, 341
_getpid, 341
_isatty, 341
_kill, 341
_link, 342
_lseek, 342
_open, 342
_stat, 342
_times, 342
_unlink, 343
_wait, 343
environ, 343
initialise_monitor_handles, 343
sysmem.c
__sbrk_heap_end, 345
_sbrk, 344
SystemClock_Config
main.c, 311
SystemCoreClock
STM32F4xx_System_Private_Variables, 57
SystemCoreClockUpdate
STM32F4xx_System_Private_Functions, 59
SystemInit
STM32F4xx_System_Private_Functions, 59

T_PREV
driving_loop.c, 290
T_REQ
driving_loop.c, 290
table_size
can.c, 268
task_args
uv_task_info, 110
task_flags
uv_task_info, 110
task_function
uv_task_info, 110
task_handle
task_management_info, 90
uv_task_info, 111
task_high_water_mark
task_status_block, 90
task_id
uv_task_info, 111
task_management_info, 89
parent_msg_queue, 89
State Engine API, 23
task_handle, 90
task_manager
State Engine, 15
task_manager_period
uv_os_settings, 104
task_name
uv_task_info, 111
task_name_lut
State Engine, 15
task_name_tree
State Engine, 15
task_overshoot_margin_crit
uv_os_settings, 104
task_overshoot_margin_noncrit
uv_os_settings, 104
task_period
uv_task_info, 111
task_priority
State Engine API, 23, 25
uv_task_info, 112
task_report_time
task_status_block, 91
task_rx_mailbox
uv_task_info, 112
task_state
uv_task_info, 112
task_status_block, 90
State Engine API, 23
task_high_water_mark, 90
task_report_time, 91
task_tardiness
State Engine, 15
task_throttle_increment
uv_os_settings, 104
temp_monitoring.c
initTempMonitor, 346
tempMonitorTask, 347
testfunc, 347
testfunc2, 347
temp_monitoring.h
initTempMonitor, 220
tempMonitorTask, 220
temp_sensor_pt1
motor_controller.h, 176
temp_sensor_pt2

motor_controller.h, 176
temp_sensor_pt3
motor_controller.h, 176
temp_sensor_pt4
motor_controller.h, 176
Temperature
imd.h, 161
tempMonitorTask
temp_monitoring.c, 347
temp_monitoring.h, 220
test1
veh_gen_info, 118
test2
veh_gen_info, 118
test3
veh_gen_info, 118
test4
veh_gen_info, 118
test5
veh_gen_info, 119
test6
veh_gen_info, 119
testfunc
temp_monitoring.c, 347
testfunc2
temp_monitoring.c, 347
throttle_daq
State Engine, 15
throttle_daq_to_preserve_performance
daq_loop_args, 67
throttle_factor
uv_task_info, 112
TICK_INT_PRIORITY
stm32f4xx_hal_conf.h, 209
tim.c
HAL_TIM_Base_MspDeInit, 348
HAL_TIM_Base_MspInit, 349
htim3, 349
MX_TIM3_Init, 349
tim.h
htim3, 222
MX_TIM3_Init, 222
TIM1_UP_TIM10_IRQHandler
stm32f4xx_it.c, 337
stm32f4xx_it.h, 219
time_constant_rotor
motor_controller.h, 173
time_constant_stator
motor_controller.h, 173
time_sent
uv_task_msg_t, 114
tmi
uv_task_info, 113
tmp_daq_msg
daq.c, 281
todo6969
motor_controller.h, 172
TOP_OF_FLASH_SBLOCK
uvfr_settings.h, 234
TOP_OF_USER_FLASH
uvfr_settings.h, 234
total_distance_cm
uv_persistant_data_frame, 105
total_params_logged
daq_loop_args, 67
total_time_driving
uv_persistant_data_frame, 105
total_vehicle_uptime
uv_persistant_data_frame, 105
Touch_energy_fault
imd.h, 160
tripzone_glitch_detected
motor_controller.h, 175
true
Utility Macros, 48
tx_all_settings_args, 91
sblock_origin, 91
uvfr_settings.c, 354
tx_journal_args, 92
end_time, 92
start_time, 92
uvfr_settings.c, 354
Tx_msg_queue
can.c, 272
TxData
constants.c, 273
constants.h, 132
uvfr_utils.c, 375
TxHeader
constants.c, 274
constants.h, 132
TxMailbox
constants.c, 274
constants.h, 132
type
daq_datapoint, 66

Update_Batt_Temp
dash.c, 281
dash.h, 138
Update_RPM
dash.c, 281
dash.h, 138
Update_State_Of_Charge
dash.c, 282
dash.h, 138
updateRunningTasks
uvfr_state_engine.h, 243
Uptime_counter
imd.h, 160
UsageFault_Handler
stm32f4xx_it.c, 337
stm32f4xx_it.h, 220
use_default_settings
uv_init_struct, 98
USE_HAL_ADC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 209

USE_HAL_CAN_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 209
USE_HAL_CEC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 209
USE_HAL_CRYP_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 209
USE_HAL_DAC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 210
USE_HAL_DCMI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 210
USE_HAL_DFSDM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 210
USE_HAL_DMA2D_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 210
USE_HAL_DSI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 210
USE_HAL_ETH_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 210
USE_HAL_FMPI2C_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 211
USE_HAL_FMPSMBUS_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 211
USE_HAL_HASH_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 211
USE_HAL_HCD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 211
USE_HAL_I2C_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 211
USE_HAL_I2S_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 211
USE_HAL_IRDA_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 212
USE_HAL_LPTIM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 212
USE_HAL_LTDC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 212
USE_HAL_MMC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 212
USE_HAL_NAND_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 212
USE_HAL_NOR_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 212
USE_HAL_PCCARD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 213
USE_HAL_PCD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 213
USE_HAL_QSPI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 213
USE_HAL_RNG_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 213
USE_HAL_RTC_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 213
USE_HAL_SAI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 213
USE_HAL_SD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 214
USE_HAL_SDRAM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 214
USE_HAL_SMARTCARD_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 214
USE_HAL_SMBUS_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 214
USE_HAL_SPDIFRX_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 214
USE_HAL_SPI_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 214
USE_HAL_SRAM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 215
USE_HAL_TIM_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 215
USE_HAL_UART_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 215
USE_HAL_USART_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 215
USE_HAL_WWDG_REGISTER_CALLBACKS
stm32f4xx_hal_conf.h, 215
USE_OLED_DEBUG
uvfr_utils.h, 247
USE_OS_MEM_MGMT
uvfr_global_config.h, 223
USE_RTOS
stm32f4xx_hal_conf.h, 215
USE_SPI_CRC
stm32f4xx_hal_conf.h, 216
Utility Macros, 43
_BV, 43
_BV_16, 43
_BV_32, 44
_BV_8, 44
deserializeBigE16, 44
deserializeBigE32, 44
deserializeSmallE16, 44
deserializeSmallE32, 45
endianSwap, 45
endianSwap16, 45
endianSwap32, 45
endianSwap8, 45
false, 46
isPowerOfTwo, 46
safePtrRead, 46
safePtrWrite, 46
serializeBigE16, 46
serializeBigE32, 47
serializeSmallE16, 47
serializeSmallE32, 47
setBits, 47
true, 48
UV19_PDU
uvfr_global_config.h, 224
uv19_pdu_settings, 92
brake_light_channel, 93
buzzer_channel, 93
coolant_pump_channel, 93
fan1_channel, 93
fan2_channel, 93
mc_rfe_channel, 93

pdu.h, 182
PDU_rx_addr, 94
PDU_tx_addr, 94
sdc_channel, 94
UV_ABORTED
uvfr_utils.h, 253
UV_ASSIGN_TASK
uvfr_utils.h, 253
UV_BINARY_SEMAPHORE
uvfr_utils.h, 251
uv_binary_semaphore_info, 94
handle, 95
UV_BOOT
State Engine API, 27
UV_CAN1
uvfr_utils.h, 247
UV_CAN2
uvfr_utils.h, 248
UV_CAN_CHANNEL_MASK
uvfr_utils.h, 248
UV_CAN_DYNAMIC_MEM
uvfr_utils.h, 248
UV_CAN_EXTENDED_ID
uvfr_utils.h, 248
uv_CAN_msg, 95
can.h, 128
data, 95
dlc, 95
flags, 96
motor_controller.h, 171
msg_id, 96
uvfr_utils.h, 249
UV_COMMAND_ACKNOWLEDGEMENT
uvfr_utils.h, 253
UV_COULDNT_DELETE
State Engine API, 26
UV_COULDNT_SUSPEND
State Engine API, 26
UV_DOUBLE
uvfr_utils.h, 252
UV_DRIVING
State Engine API, 27
uv_driving_mode_t
uvfr_utils.h, 252
UV_DUMB_FLAG
uvfr_utils.h, 251
UV_ERROR
uvfr_utils.h, 253
UV_ERROR_REPORT
uvfr_utils.h, 253
UV_ERROR_STATE
State Engine API, 27
uv_ext_device_id
uvfr_utils.h, 249
uv_external_device
uvfr_utils.h, 252
UV_FLOAT
uvfr_utils.h, 252
UV_HALT
State Engine API, 27
uv_imd_settings, 96
expected_isolation_capacitances, 97
imd.h, 159
max_imd_temperature, 97
min_isolation_resistances, 97
UV_INIT
State Engine API, 27
uv_init_struct, 97
use_default_settings, 98
uvfr_utils.h, 249
uv_init_task_args, 98
init_info_queue, 99
meta_task_handle, 99
specific_args, 99
uvfr_utils.h, 249
uv_init_task_response, 99
device, 100
errmsg, 100
nchar, 100
status, 100
uvfr_utils.h, 249
UV_INT16
uvfr_utils.h, 251
UV_INT32
uvfr_utils.h, 252
UV_INT64
uvfr_utils.h, 252
UV_INT8
uvfr_utils.h, 251
uv_internal_params, 101
e_code, 101
init_params, 101
peripheral_status, 101
uvfr_utils.h, 250
vehicle_settings, 101
UV_INVALID_MSG
uvfr_utils.h, 253
UV_KILL_CMD
State Engine API, 26
UV_LAUNCH_CONTROL
State Engine API, 27
UV_MALLOC_LIMIT
uvfr_global_config.h, 224
uv_msg_type
uvfr_utils.h, 250
uv_msg_type_t
uvfr_utils.h, 252
UV_MUTEX
uvfr_utils.h, 251
uv_mutex_info, 102
handle, 102
UV_NO_CMD
State Engine API, 26
UV_NONE
uvfr_utils.h, 251
UV_OK

uvfr_utils.h, 253
UV_OS_ATTEMPT_RESTART_NC_TASK
State Engine API, 25
UV_OS_ENABLE_NONCRIT_TASK_THROTTLE
State Engine API, 25
UV_OS_LOG_MEM
State Engine API, 25
UV_OS_LOG_TASK_END_TIME
State Engine API, 25
uv_os_settings, 102
max_svc_task_period, 103
max_task_period, 103
min_task_period, 103
os_flags, 103
State Engine API, 24
svc_task_manager_period, 104
task_manager_period, 104
task_overshoot_margin_crit, 104
task_overshoot_margin_noncrit, 104
task_throttle_increment, 104
UV_PARAM_READY
uvfr_utils.h, 253
UV_PARAM_REQUEST
uvfr_utils.h, 253
uv_persistant_data_frame, 105
odometer.h, 179
total_distance_cm, 105
total_time_driving, 105
total_vehicle_uptime, 105
UV_RAW_DATA_TRANSFER
uvfr_utils.h, 253
UV_READY
State Engine API, 27
UV_SC_COMMAND
uvfr_utils.h, 253
uv_scd_response, 106
meta_id, 106
response_val, 106
State Engine API, 24
uv_scd_response_e
State Engine API, 25
UV_SEMAPHORE
uvfr_utils.h, 251
uv_semaphore_info, 106
handle, 107
uv_status
can.h, 128
daq.h, 134
uvfr_settings.h, 235
uvfr_state_engine.h, 243
uvfr_utils.h, 250
uv_status_t
uvfr_utils.h, 253
UV_STRING
uvfr_utils.h, 252
UV_SUCCESSFUL_DELETION
State Engine API, 26
UV_SUCCESSFUL_SUSPENSION
State Engine API, 26
UV_SUSPEND_CMD
State Engine API, 26
UV_SUSPENDED
State Engine API, 27
UV_TASK_AWAITING_DELETION
State Engine API, 18
uv_task_cmd
State Engine API, 24
uvfr_utils.h, 250
uv_task_cmd_e
State Engine API, 26
UV_TASK_DEADLINE_FIRM
State Engine API, 18
UV_TASK_DEADLINE_HARD
State Engine API, 18
UV_TASK_DEADLINE_MASK
State Engine API, 18
UV_TASK_DEADLINE_NOT_ENFORCED
State Engine API, 19
UV_TASK_DEFER_DELETION
State Engine API, 19
UV_TASK_DELAYING
State Engine API, 19
UV_TASK_DELETE_COMMAND
uvfr_utils.h, 253
UV_TASK_DELETED
State Engine API, 26
UV_TASK_DORMANT_SVC
State Engine API, 19
UV_TASK_ERR_IN_CHILD
State Engine API, 19
UV_TASK_GENERIC_SVC
State Engine API, 19
uv_task_id
uvfr_state_engine.h, 243
uvfr_utils.h, 250
uv_task_info, 107
active_states, 108
cmd_data, 108
deletion_delay, 108
deletion_states, 108
last_execution_time, 109
parent, 109
stack_size, 109
State Engine API, 24
suspension_states, 109
task_args, 110
task_flags, 110
task_function, 110
task_handle, 111
task_id, 111
task_name, 111
task_period, 111
task_priority, 112
task_rx_mailbox, 112
task_state, 112
throttle_factor, 112

tmi, 113
UV_TASK_IS_CHILD
State Engine API, 20
UV_TASK_IS_ORPHAN
State Engine API, 20
UV_TASK_IS_PARENT
State Engine API, 20
UV_TASK_LOG_MEM_USAGE
State Engine API, 20
UV_TASK_LOG_START_STOP_TIME
State Engine API, 20
UV_TASK_MANAGER_MASK
State Engine API, 20
UV_TASK_MISSION_CRITICAL
State Engine API, 21
uv_task_msg
uvfr_utils.h, 250
uv_task_msg_t, 113
intended_recipient, 114
message_size, 114
message_type, 114
msg_contents, 114
sender, 114
time_sent, 114
UV_TASK_NOT_STARTED
State Engine API, 26
UV_TASK_PERIODIC_SVC
State Engine API, 21
UV_TASK_PRIO_INCREMENTATION
State Engine API, 21
UV_TASK_RUNNING
State Engine API, 26
UV_TASK_SCD_IGNORE
State Engine API, 21
UV_TASK_START_CMD
State Engine API, 26
UV_TASK_START_COMMAND
uvfr_utils.h, 253
uv_task_state_t
State Engine API, 26
uv_task_status
State Engine API, 24
UV_TASK_STATUS_REPORT
uvfr_utils.h, 253
UV_TASK_SUSPEND_COMMAND
uvfr_utils.h, 253
UV_TASK_SUSPENDED
State Engine API, 26
UV_TASK_VEHICLE_APPLICATION
State Engine API, 21
uv_timespan_ms
uvfr_state_engine.h, 243
uvfr_utils.h, 251
UV_UINT16
uvfr_utils.h, 251
UV_UINT32
uvfr_utils.h, 251
UV_UINT64
uvfr_utils.h, 252
UV_UINT8
uvfr_utils.h, 251
UV_UNSAFE_STATE
State Engine API, 26
UV_UTILS_SRC_IMPLIMENTATION
uvfr_utils.c, 370
uv_vehicle_settings, 115
bms_settings, 115
daq_param_list, 115
daq_settings, 116
driving_loop_settings, 116
flags, 116
imd_settings, 116
mc_settings, 116
os_settings, 117
pdu_settings, 117
uvfr_settings.h, 235
veh_info, 117
uv_vehicle_state
State Engine API, 24
uv_vehicle_state_t
State Engine API, 27
UV_WAKEUP
uvfr_utils.h, 253
UV_WARNING
uvfr_utils.h, 253
uvAbortTaskDeletion
State Engine Internals, 35
uvAwaitLaptopAckMsg
uvfr_settings.c, 356
uvComputeChunkChecksum
uvfr_settings.c, 356
uvComputeMemRegionChecksum
uvfr_settings.c, 356
uvfr_settings.h, 236
uvComputeSettingsCheckSums
uvfr_settings.h, 237
uvComputeSettingsChecksums
uvfr_settings.c, 356
uvConfigSettingTask
uvfr_settings.c, 357
uvfr_settings.h, 237
uvCreateServiceTask
State Engine Internals, 35
uvCreateTask
State Engine API, 28
uvCreateTmpSettingsCopy
uvfr_settings.c, 357
uvDeInitStateEngine
State Engine API, 28
uvDeleteSVCTask
State Engine Internals, 35
uvDeleteTask
State Engine Internals, 36
uvForceDefaultReversionUponDeviceReset
uvfr_settings.c, 357
UVFR CANbus API, 50

insertCANMessageHandler, 50
uvSendCanMSG, 50
UVFR Utilities, 42
UVFR Vehicle Commands, 49
uvfr_global_config.h
ECUMASTER_PMU, 222
FIRMWARE_MAJOR_RELEASE, 223
FIRMWARE_MINOR_RELEASE, 223
FIRMWARE_PATCH_NUM, 223
STM32_F407, 223
STM32_H7xx, 223
USE_OS_MEM_MGMT, 223
UV19_PDU, 224
UV_MALLOC_LIMIT, 224
uvfr_settings.c
_e_uvdata, 362
_s_uvdata, 362
child_task_active, 362
child_task_handle, 363
current_vehicle_settings, 363
DATA_MISMATCH, 352
default_bms_settings, 363
default_daq_settings, 363
default_datapoints, 363
default_dl_settings, 364
default_imd_settings, 364
default_pdu_settings, 364
default_vehicle, 364
DID_NOT_FINISH_PROGRAMMING, 352
FLASH_NOT_UNLOCKED, 352
FLASH_OK, 352
handleIncomingLaptopMsg, 354
helper_task_args, 353
INVALID_SBLOCK, 352
is_laptop_connected, 364
LAPTOP_TO_VCU_ID, 352
last_contact_with_laptop, 365
mc_default_settings, 365
nukeSettings, 354
POST_CHECKSUM, 353
PRE_CHECKSUM, 353
ptask_handle, 365
sendAllSettingsWorker, 354
sendJournalWorker, 355
setting_helper_task, 353
settingCopy, 355
settings_queue, 365
setupDefaultSettings, 355
SRC_UVFR_SETTINGS_C_, 353
tx_all_settings_args, 354
tx_journal_args, 354
uvAwaitLaptopAckMsg, 356
uvComputeChunkChecksum, 356
uvComputeMemRegionChecksum, 356
uvComputeSettingsChecksums, 356
uvConfigSettingTask, 357
uvCreateTmpSettingsCopy, 357
uvForceDefaultReversionUponDeviceReset, 357
uvLoadSettingsFromFlash, 357
uvOverwriteCsr, 358
uvResetFlashToDefault, 358
uvSaveSettingsToFlash, 358
uvSBlockCorruptionHandler, 359
uvSendSettingGroup, 359
uvSendSpecificParam, 359
uvSetSettingResponseReminder, 360
uvSettingsInit, 360
uvSettingsProgrammerTask, 360
uvTransmitVehicleStatus, 361
uvUpdateTmpSettings, 361
uvValidateChecksums, 361
uvValidateFlashSettings, 362
vcu_ack_failed_msg, 365
vcu_ack_msg, 366
VCU_TO_LAPTOP_ID, 353
uvfr_settings.h
BMS_ADDR, 226
BMS_MGROUP, 226
BMS_OFFSET, 227
CANNOT_PERFORM_REQUEST, 236
CLEAR_FAULTS, 236
CRC_MGROUP1, 227
CRC_MGROUP2, 227
CRC_POLY, 227
current_vehicle_settings, 238
DAQ_HEAD_ADDR, 227
DAQ_HEAD_MGROUP, 227
DAQ_HEAD_OFFSET, 228
DAQ_PARAMS1_ADDR, 228
DAQ_PARAMS1_MGROUP, 228
DAQ_PARAMS1_OFFSET, 228
DAQ_PARAMS2_ADDR, 228
DAQ_PARAMS2_MGROUP, 228
DAQ_PARAMS2_OFFSET, 229
DAQ_PARAMS3_ADDR, 229
DAQ_PARAMS3_MGROUP, 229
DAQ_PARAMS3_OFFSET, 229
DISCARD_NEW_SETTINGS, 236
DISCARD_NEW_SETTINGS_AND_EXIT, 236
DRIVING_ADDR, 229
DRIVING_MGROUP, 229
DRIVING_OFFSET, 230
ENABLE_FLASH_SETTINGS, 230
END_OF_SPECIFIC_PARAMS, 236
ENTER_PROGRAMMING_MODE, 236
FLASH_SBLOCK_START, 230
FORCE_RESTORE_FACTORY_DEFAULT, 236
GENERAL_VEH_INFO_MGROUP, 230
GENERAL_VEH_INFO_OFFSET, 230
GENERIC_ACK, 236
HANDSHAKE, 236
IMD_ADDR, 230
IMD_MGROUP, 231
IMD_OFFSET, 231
isPtrToFlash, 231
laptop_CMD, 235

MAGIC_NUMBER, 231
MOTOR_ADDR, 231
motor_controller_settings, 235
MOTOR_MGROUP, 231
MOTOR_OFFSET, 232
nukeSettings, 236
OS_SETTINGS_ADDR, 232
OS_SETTINGS_MGROUP, 232
OS_SETTINGS_OFFSET, 232
PDU_ADDR, 232
PDU_MGROUP, 232
PDU_OFFSET, 233
PERSISTENT_DATA_MGROUP, 233
REQUEST_ALL_JOURNAL_ENTRIES, 236
REQUEST_ALL_SETTINGS, 236
REQUEST_JOURNAL_ENTRIES_BY_TIME, 236
REQUEST_SPECIFIC_SETTING, 236
REQUEST_VCU_FIRMWARE_VERSION, 236
REQUEST_VCU_STATUS, 236
SAVE_AND_APPLY_NEW_SETTINGS, 236
SBLOCK_CRC_REGION_OFFSET, 233
SBLOCK_CSR_OFFSET, 233
SET_SPECIFIC_PARAM, 236
SETTING_BRANCH_SIZE, 233
setupDefaultSettings, 236
SIZE_OF_MGROUP, 233
SIZEOF_SBLOCK, 234
SIZEOF_USER_FLASH, 234
START_OF_USER_FLASH, 234
SYS_DATA_OFFSET, 234
TOP_OF_FLASH_SBLOCK, 234
TOP_OF_USER_FLASH, 234
uv_status, 235
uv_vehicle_settings, 235
uvComputeMemRegionChecksum, 236
uvComputeSettingsCheckSums, 237
uvConfigSettingTask, 237
uvSaveSettingsToFlash, 237
uvSettingsInit, 237
uvValidateChecksums, 238
uvValidateSettingsFromFlash, 238
veh_gen_info, 235
uvfr_state_engine.c
UVFR_STATE_MACHINE_IMPLIMENTATION, 369
uvfr_state_engine.h
_LONGEST_SC_TIME, 241
_SC_DAEMON_PERIOD, 242
_UV_DEFAULT_TASK_INSTANCES, 242
_UV_DEFAULT_TASK_PERIOD, 242
_UV_DEFAULT_TASK_STACK_SIZE, 242
_UV_MIN_TASK_PERIOD, 242
getSVCTaskID, 243
SVC_TASK_MAX_CHECKIN_PERIOD, 242
updateRunningTasks, 243
uv_status, 243
uv_task_id, 243
uv_timespan_ms, 243
uvGetTaskById, 244
uvRegisterTask, 244
UVFR_STATE_MACHINE_IMPLIMENTATION
uvfr_state_engine.c, 369
uvfr_utils.c
__uvFreeCritSection, 370
__uvFreeOS, 370
__uvInitPanic, 370
__uvMallocCritSection, 371
__uvMallocOS, 371
data_size, 375
init_task_handle, 375
reset_handle, 375
setup_extern_devices, 371
TxData, 375
UV_UTILS_SRC_IMPLIMENTATION, 370
uvInit, 371
uvIsPTRValid, 373
uvSysResetDaemon, 374
uvUtilsReset, 374
uvfr_utils.h
__uvInitPanic, 253
accel, 252
access_control_info, 248
access_control_t, 251
access_control_type, 248
BMS, 252
bool, 249
data_size, 256
data_type, 251
econ, 252
global_context, 256
IMD, 252
INIT_CHECK_PERIOD, 247
limp, 252
MAX_INIT_TIME, 247
MOTOR_CONTROLLER, 252
normal, 252
p_status, 249
PDU, 252
USE_OLED_DEBUG, 247
UV_ABORTED, 253
UV_ASSIGN_TASK, 253
UV_BINARY_SEMAPHORE, 251
UV_CAN1, 247
UV_CAN2, 248
UV_CAN_CHANNEL_MASK, 248
UV_CAN_DYNAMIC_MEM, 248
UV_CAN_EXTENDED_ID, 248
uv_CAN_msg, 249
UV_COMMAND_ACKNOWLEDGEMENT, 253
UV_DOUBLE, 252
uv_driving_mode_t, 252
UV_DUMB_FLAG, 251
UV_ERROR, 253
UV_ERROR_REPORT, 253
uv_ext_device_id, 249
uv_external_device, 252
UV_FLOAT, 252

uv_init_struct, 249
uv_init_task_args, 249
uv_init_task_response, 249
UV_INT16, 251
UV_INT32, 252
UV_INT64, 252
UV_INT8, 251
uv_internal_params, 250
UV_INVALID_MSG, 253
uv_msg_type, 250
uv_msg_type_t, 252
UV_MUTEX, 251
UV_NONE, 251
UV_OK, 253
UV_PARAM_READY, 253
UV_PARAM_REQUEST, 253
UV_RAW_DATA_TRANSFER, 253
UV_SC_COMMAND, 253
UV_SEMAPHORE, 251
uv_status, 250
uv_status_t, 253
UV_STRING, 252
uv_task_cmd, 250
UV_TASK_DELETE_COMMAND, 253
uv_task_id, 250
uv_task_msg, 250
UV_TASK_START_COMMAND, 253
UV_TASK_STATUS_REPORT, 253
UV_TASK_SUSPEND_COMMAND, 253
uv_timespan_ms, 251
UV_UINT16, 251
UV_UINT32, 251
UV_UINT64, 252
UV_UINT8, 251
UV_WAKEUP, 253
UV_WARNING, 253
uvInit, 254
uvIsPTRValid, 256
uvfr_vehicle_commands.c
uvSecureVehicle, 376
uvfr_vehicle_commands.h
_uvCloseSDC_canBased, 259
_uvHonkHorn_canBased, 259
_uvOpenSDC_canBased, 259
_uvSilenceHorn_canBased, 259
_uvStartCoolantPump_canBased, 259
_uvStopCoolantPump_canBased, 259
uvHonkHorn, 257
uvOpenSDC, 257
uvSecureVehicle, 259
uvSilenceHorn, 258
uvStartCoolantPump, 258
uvStartFans, 258
uvStopCoolantPump, 258
uvStopFans, 258
uvGetTaskById
uvfr_state_engine.h, 244
uvGetTaskFromName
State Engine Internals, 36
uvGetTaskFromRTOSHandle
State Engine Internals, 36
uvHonkHorn
uvfr_vehicle_commands.h, 257
uvInit
uvfr_utils.c, 371
uvfr_utils.h, 254
uvInitStateEngine
State Engine API, 28
uvIsPTRValid
uvfr_utils.c, 373
uvfr_utils.h, 256
uvKillTaskViolently
State Engine Internals, 37
uvLateTaskHandler
State Engine Internals, 37
uvLoadSettingsFromFlash
uvfr_settings.c, 357
uvOpenSDC
uvfr_vehicle_commands.h, 257
uvOverwriteCsr
uvfr_settings.c, 358
uvRegisterTask
uvfr_state_engine.h, 244
uvResetFlashToDefault
uvfr_settings.c, 358
uvRestartSVCTask
State Engine Internals, 37
uvSaveSettingsToFlash
uvfr_settings.c, 358
uvfr_settings.h, 237
uvSBlockCorruptionHandler
uvfr_settings.c, 359
uvScheduleTaskDeletion
State Engine Internals, 38
uvSecureVehicle
uvfr_vehicle_commands.c, 376
uvfr_vehicle_commands.h, 259
uvSendCanMSG
UVFR CANbus API, 50
uvSendSettingGroup
uvfr_settings.c, 359
uvSendSpecificParam
uvfr_settings.c, 359
uvSendTaskStatusReport
State Engine Internals, 38
uvSetSettingResponseReminder
uvfr_settings.c, 360
uvSettingsInit
uvfr_settings.c, 360
uvfr_settings.h, 237
uvSettingsProgrammerTask
uvfr_settings.c, 360
uvSilenceHorn
uvfr_vehicle_commands.h, 258
uvStartCoolantPump
uvfr_vehicle_commands.h, 258

uvStartFans
uvfr_vehicle_commands.h, 258
uvStartStateMachine
State Engine API, 29
uvStartSVCTask
State Engine Internals, 38
uvStartTask
State Engine Internals, 38
uvStopCoolantPump
uvfr_vehicle_commands.h, 258
uvStopFans
uvfr_vehicle_commands.h, 258
uvSuspendSVCTask
State Engine Internals, 39
uvSuspendTask
State Engine Internals, 39
uvSVCTaskManager
State Engine, 11
uvSysResetDaemon
uvfr_utils.c, 374
uvTaskCrashHandler
State Engine Internals, 40
uvTaskDelay
State Engine API, 21
uvTaskDelayUntil
State Engine API, 22
uvTaskIsDelaying
State Engine API, 22
uvTaskManager
State Engine Internals, 40
uvTaskPeriodEnd
State Engine, 11
State Engine API, 29
uvTaskResetDelayBit
State Engine API, 22
uvTaskResetDeletionBit
State Engine API, 22
uvTaskSetDelayBit
State Engine API, 23
uvTaskSetDeletionBit
State Engine API, 23
uvThrottleNonCritTasks
State Engine Internals, 41
uvTransmitVehicleStatus
uvfr_settings.c, 361
uvUpdateTmpSettings
uvfr_settings.c, 361
uvUtilsReset
uvfr_utils.c, 374
uvValidateChecksums
uvfr_settings.c, 361
uvfr_settings.h, 238
uvValidateFlashSettings
uvfr_settings.c, 362
uvValidateManagedTasks
State Engine Internals, 41
uvValidateSettingsFromFlash
uvfr_settings.h, 238
vApplicationGetIdleTaskMemory
freertos.c, 292
vApplicationGetTimerTaskMemory
freertos.c, 292
vApplicationIdleHook
freertos.c, 292
vApplicationMallocFailedHook
freertos.c, 293
vApplicationStackOverflowHook
freertos.c, 293
vApplicationTickHook
freertos.c, 293
Vb_hi_res
imd.h, 159
vcu_ack_failed_msg
uvfr_settings.c, 365
vcu_ack_msg
uvfr_settings.c, 366
VCU_CURRENT_UPTIME
daq.h, 135
VCU_ERROR_BITFIELD1
daq.h, 135
VCU_ERROR_BITFIELD2
daq.h, 135
VCU_ERROR_BITFIELD3
daq.h, 135
VCU_ERROR_BITFIELD4
daq.h, 135
VCU_TO_LAPTOP_ID
uvfr_settings.c, 353
VCU_TOTAL_UPTIME
daq.h, 135
VCU_VEHICLE_STATE
daq.h, 135
VDD_VALUE
stm32f4xx_hal_conf.h, 216
VEH_DISTANCE_RUN
daq.h, 135
VEH_DISTANCE_TOTAL
daq.h, 135
veh_gen_info, 117
drive_ratio, 118
test1, 118
test2, 118
test3, 118
test4, 118
test5, 119
test6, 119
uvfr_settings.h, 235
wheel_size, 119
veh_info
uv_vehicle_settings, 117
VEH_LAPNUM
daq.h, 135
VEH_SPEED
daq.h, 135
vehicle_settings
uv_internal_params, 101

vehicle_state
State Engine, 15
Version_0
imd.h, 160
Version_1
imd.h, 160
Version_2
imd.h, 160
Vexc_hi_res
imd.h, 159
Vn_hi_res
imd.h, 159
voltages_Vp_and_Vn
imd.h, 161
Vout_saturation_max_limit
motor_controller.h, 175
Vp_hi_res
imd.h, 159
vPortSVCHandler
FreeRTOSConfig.h, 156
Vpwr_hi_res
imd.h, 159

wait
oled.h, 181
warning_5
motor_controller.h, 175
warning_9
motor_controller.h, 175
watchdog_reset
motor_controller.h, 175
wheel_size
veh_gen_info, 119
WSS_F_AVG
daq.h, 135
WSS_FL
daq.h, 135
WSS_FR
daq.h, 135
WSS_R_AVG
daq.h, 136
WSS_RL
daq.h, 135
WSS_RR
daq.h, 135
WSS_SLIP
daq.h, 136

xIdleStack
freertos.c, 294
xIdleTaskTCBBuffer
freertos.c, 294
xPortPendSVHandler
FreeRTOSConfig.h, 156
xPortSysTickHandler
FreeRTOSConfig.h, 156
xTimerStack
freertos.c, 294
xTimerTaskTCBBuffer
freertos.c, 295