

Vex Team A

1.0.1

Generated by Doxygen 1.8.13

Contents

| | | |
|----------|--|----------|
| 1 | InTheZoneA | 1 |
| 2 | Data Structure Index | 3 |
| 2.1 | Data Structures | 3 |
| 3 | File Index | 5 |
| 3.1 | File List | 5 |
| 4 | Data Structure Documentation | 7 |
| 4.1 | cord Struct Reference | 7 |
| 4.1.1 | Detailed Description | 7 |
| 4.1.2 | Field Documentation | 7 |
| 4.1.2.1 | x | 8 |
| 4.1.2.2 | y | 8 |
| 4.2 | lcd_buttons Struct Reference | 8 |
| 4.2.1 | Detailed Description | 8 |
| 4.2.2 | Field Documentation | 9 |
| 4.2.2.1 | left | 9 |
| 4.2.2.2 | middle | 9 |
| 4.2.2.3 | right | 9 |
| 4.3 | menu_t Struct Reference | 9 |
| 4.3.1 | Detailed Description | 10 |
| 4.3.2 | Field Documentation | 11 |
| 4.3.2.1 | current | 11 |
| 4.3.2.2 | length | 11 |

| | | |
|----------|--------------------------------|-----------|
| 4.3.2.3 | max | 12 |
| 4.3.2.4 | max_f | 12 |
| 4.3.2.5 | min | 12 |
| 4.3.2.6 | min_f | 13 |
| 4.3.2.7 | options | 13 |
| 4.3.2.8 | prompt | 13 |
| 4.3.2.9 | step | 14 |
| 4.3.2.10 | step_f | 14 |
| 4.3.2.11 | type | 14 |
| 4.4 | polar_cord Struct Reference | 15 |
| 4.4.1 | Detailed Description | 15 |
| 4.4.2 | Field Documentation | 15 |
| 4.4.2.1 | angle | 15 |
| 4.4.2.2 | magnitue | 15 |
| 5 | File Documentation | 17 |
| 5.1 | include/API.h File Reference | 17 |
| 5.1.1 | Detailed Description | 21 |
| 5.1.2 | Macro Definition Documentation | 21 |
| 5.1.2.1 | ACCEL_X | 21 |
| 5.1.2.2 | ACCEL_Y | 22 |
| 5.1.2.3 | BOARD_NR_ADC_PINS | 22 |
| 5.1.2.4 | BOARD_NR_GPIO_PINS | 22 |
| 5.1.2.5 | EOF | 22 |
| 5.1.2.6 | HIGH | 23 |
| 5.1.2.7 | IME_ADDR_MAX | 23 |
| 5.1.2.8 | INPUT | 23 |
| 5.1.2.9 | INPUT_ANALOG | 23 |
| 5.1.2.10 | INPUT_FLOATING | 24 |
| 5.1.2.11 | INTERRUPT_EDGE_BOTH | 24 |
| 5.1.2.12 | INTERRUPT_EDGE_FALLING | 24 |

| | | |
|----------|-----------------------------------|----|
| 5.1.2.13 | INTERRUPT_EDGE_RISING | 24 |
| 5.1.2.14 | JOY_DOWN | 24 |
| 5.1.2.15 | JOY_LEFT | 25 |
| 5.1.2.16 | JOY_RIGHT | 25 |
| 5.1.2.17 | JOY_UP | 25 |
| 5.1.2.18 | LCD_BTN_CENTER | 25 |
| 5.1.2.19 | LCD_BTN_LEFT | 25 |
| 5.1.2.20 | LCD_BTN_RIGHT | 26 |
| 5.1.2.21 | LOW | 26 |
| 5.1.2.22 | OUTPUT | 26 |
| 5.1.2.23 | OUTPUT_OD | 26 |
| 5.1.2.24 | SEEK_CUR | 27 |
| 5.1.2.25 | SEEK_END | 27 |
| 5.1.2.26 | SEEK_SET | 27 |
| 5.1.2.27 | SERIAL_8N1 | 27 |
| 5.1.2.28 | SERIAL_DATABITS_8 | 27 |
| 5.1.2.29 | SERIAL_DATABITS_9 | 28 |
| 5.1.2.30 | SERIAL_PARITY_EVEN | 28 |
| 5.1.2.31 | SERIAL_PARITY_NONE | 28 |
| 5.1.2.32 | SERIAL_PARITY_ODD | 28 |
| 5.1.2.33 | SERIAL_STOPBITS_1 | 28 |
| 5.1.2.34 | SERIAL_STOPBITS_2 | 29 |
| 5.1.2.35 | stdin | 29 |
| 5.1.2.36 | stdout | 29 |
| 5.1.2.37 | TASK_DEAD | 29 |
| 5.1.2.38 | TASK_DEFAULT_STACK_SIZE | 29 |
| 5.1.2.39 | TASK_MAX | 30 |
| 5.1.2.40 | TASK_MAX_PRIORITIES | 30 |
| 5.1.2.41 | TASK_MINIMAL_STACK_SIZE | 30 |
| 5.1.2.42 | TASK_PRIORITY_DEFAULT | 30 |

| | | |
|----------|--------------------------|----|
| 5.1.2.43 | TASK_PRIORITY_HIGHEST | 31 |
| 5.1.2.44 | TASK_PRIORITY_LOWEST | 31 |
| 5.1.2.45 | TASK_RUNNABLE | 31 |
| 5.1.2.46 | TASK_RUNNING | 31 |
| 5.1.2.47 | TASK_SLEEPING | 31 |
| 5.1.2.48 | TASK_SUSPENDED | 32 |
| 5.1.2.49 | uart1 | 32 |
| 5.1.2.50 | uart2 | 32 |
| 5.1.3 | Typedef Documentation | 32 |
| 5.1.3.1 | Encoder | 32 |
| 5.1.3.2 | FILE | 33 |
| 5.1.3.3 | Gyro | 33 |
| 5.1.3.4 | InterruptHandler | 33 |
| 5.1.3.5 | Mutex | 33 |
| 5.1.3.6 | Semaphore | 34 |
| 5.1.3.7 | TaskCode | 34 |
| 5.1.3.8 | TaskHandle | 34 |
| 5.1.3.9 | Ultrasonic | 34 |
| 5.1.4 | Function Documentation | 34 |
| 5.1.4.1 | __attribute__() | 34 |
| 5.1.4.2 | analogCalibrate() | 35 |
| 5.1.4.3 | analogRead() | 35 |
| 5.1.4.4 | analogReadCalibrated() | 36 |
| 5.1.4.5 | analogReadCalibratedHR() | 36 |
| 5.1.4.6 | delay() | 37 |
| 5.1.4.7 | delayMicroseconds() | 37 |
| 5.1.4.8 | digitalRead() | 38 |
| 5.1.4.9 | digitalWrite() | 38 |
| 5.1.4.10 | encoderGet() | 39 |
| 5.1.4.11 | encoderInit() | 39 |

| | | |
|----------|------------------------------------|----|
| 5.1.4.12 | encoderReset() | 39 |
| 5.1.4.13 | encoderShutdown() | 40 |
| 5.1.4.14 | fclose() | 40 |
| 5.1.4.15 | fcount() | 40 |
| 5.1.4.16 | fdelete() | 41 |
| 5.1.4.17 | feof() | 41 |
| 5.1.4.18 | fflush() | 42 |
| 5.1.4.19 | fgetc() | 42 |
| 5.1.4.20 | fgets() | 42 |
| 5.1.4.21 | fopen() | 43 |
| 5.1.4.22 | fprintf() | 43 |
| 5.1.4.23 | fprintff() | 44 |
| 5.1.4.24 | fputc() | 45 |
| 5.1.4.25 | fputs() | 45 |
| 5.1.4.26 | fread() | 46 |
| 5.1.4.27 | fseek() | 46 |
| 5.1.4.28 | ftell() | 47 |
| 5.1.4.29 | fwrite() | 47 |
| 5.1.4.30 | getchar() | 47 |
| 5.1.4.31 | gyroGet() | 48 |
| 5.1.4.32 | gyroInit() | 48 |
| 5.1.4.33 | gyroReset() | 49 |
| 5.1.4.34 | gyroShutdown() | 49 |
| 5.1.4.35 | i2cRead() | 49 |
| 5.1.4.36 | i2cReadRegister() | 50 |
| 5.1.4.37 | i2cWrite() | 50 |
| 5.1.4.38 | i2cWriteRegister() | 50 |
| 5.1.4.39 | imeGet() | 51 |
| 5.1.4.40 | imeGetVelocity() | 51 |
| 5.1.4.41 | imeInitializeAll() | 52 |

| | | |
|----------|-----------------------|----|
| 5.1.4.42 | imeReset() | 53 |
| 5.1.4.43 | imeShutdown() | 53 |
| 5.1.4.44 | ioClearInterrupt() | 53 |
| 5.1.4.45 | ioSetInterrupt() | 54 |
| 5.1.4.46 | isAutonomous() | 54 |
| 5.1.4.47 | isEnabled() | 54 |
| 5.1.4.48 | isJoystickConnected() | 55 |
| 5.1.4.49 | isOnline() | 55 |
| 5.1.4.50 | joystickGetAnalog() | 55 |
| 5.1.4.51 | joystickGetDigital() | 56 |
| 5.1.4.52 | lcdClear() | 56 |
| 5.1.4.53 | lcdInit() | 57 |
| 5.1.4.54 | lcdReadButtons() | 58 |
| 5.1.4.55 | lcdSetBacklight() | 58 |
| 5.1.4.56 | lcdSetText() | 59 |
| 5.1.4.57 | lcdShutdown() | 60 |
| 5.1.4.58 | micros() | 60 |
| 5.1.4.59 | millis() | 60 |
| 5.1.4.60 | motorGet() | 60 |
| 5.1.4.61 | motorSet() | 61 |
| 5.1.4.62 | motorStop() | 62 |
| 5.1.4.63 | motorStopAll() | 62 |
| 5.1.4.64 | mutexCreate() | 62 |
| 5.1.4.65 | mutexDelete() | 63 |
| 5.1.4.66 | mutexGive() | 63 |
| 5.1.4.67 | mutexTake() | 64 |
| 5.1.4.68 | pinMode() | 65 |
| 5.1.4.69 | powerLevelBackup() | 65 |
| 5.1.4.70 | powerLevelMain() | 66 |
| 5.1.4.71 | print() | 66 |

| | | |
|-----------|----------------------|----|
| 5.1.4.72 | printf() | 66 |
| 5.1.4.73 | putchar() | 67 |
| 5.1.4.74 | puts() | 68 |
| 5.1.4.75 | semaphoreCreate() | 68 |
| 5.1.4.76 | semaphoreDelete() | 68 |
| 5.1.4.77 | semaphoreGive() | 69 |
| 5.1.4.78 | semaphoreTake() | 69 |
| 5.1.4.79 | setTeamName() | 69 |
| 5.1.4.80 | snprintf() | 70 |
| 5.1.4.81 | speakerInit() | 70 |
| 5.1.4.82 | speakerPlayArray() | 71 |
| 5.1.4.83 | speakerPlayRtttl() | 71 |
| 5.1.4.84 | speakerShutdown() | 71 |
| 5.1.4.85 | sprintf() | 72 |
| 5.1.4.86 | taskCreate() | 72 |
| 5.1.4.87 | taskDelay() | 73 |
| 5.1.4.88 | taskDelayUntil() | 73 |
| 5.1.4.89 | taskDelete() | 74 |
| 5.1.4.90 | taskGetCount() | 74 |
| 5.1.4.91 | taskGetState() | 74 |
| 5.1.4.92 | taskPriorityGet() | 75 |
| 5.1.4.93 | taskPrioritySet() | 75 |
| 5.1.4.94 | taskResume() | 76 |
| 5.1.4.95 | taskRunLoop() | 76 |
| 5.1.4.96 | taskSuspend() | 77 |
| 5.1.4.97 | ultrasonicGet() | 77 |
| 5.1.4.98 | ultrasonicInit() | 78 |
| 5.1.4.99 | ultrasonicShutdown() | 78 |
| 5.1.4.100 | usartInit() | 78 |
| 5.1.4.101 | usartShutdown() | 79 |

| | | |
|-----------|-------------------------------------|----|
| 5.1.4.102 | wait() | 79 |
| 5.1.4.103 | waitUntil() | 80 |
| 5.1.4.104 | watchdogInit() | 80 |
| 5.1.5 | Variable Documentation | 80 |
| 5.1.5.1 | formatString | 80 |
| 5.1.5.2 | line | 81 |
| 5.2 | include/battery.h File Reference | 81 |
| 5.3 | include/controller.h File Reference | 81 |
| 5.3.1 | Detailed Description | 82 |
| 5.3.2 | Macro Definition Documentation | 83 |
| 5.3.2.1 | LEFT_JOY_X | 83 |
| 5.3.2.2 | LEFT_JOY_Y | 83 |
| 5.3.2.3 | MASTER | 84 |
| 5.3.2.4 | PARTNER | 84 |
| 5.3.2.5 | RIGHT_JOY_X | 84 |
| 5.3.2.6 | RIGHT_JOY_Y | 85 |
| 5.3.3 | Enumeration Type Documentation | 85 |
| 5.3.3.1 | joystick | 85 |
| 5.3.4 | Function Documentation | 86 |
| 5.3.4.1 | get_joystick_cord() | 86 |
| 5.4 | include/drive.h File Reference | 87 |
| 5.4.1 | Detailed Description | 88 |
| 5.4.2 | Macro Definition Documentation | 88 |
| 5.4.2.1 | DEADSPOT | 88 |
| 5.4.3 | Typedef Documentation | 88 |
| 5.4.3.1 | side_t | 89 |
| 5.4.4 | Enumeration Type Documentation | 89 |
| 5.4.4.1 | side | 89 |
| 5.4.5 | Function Documentation | 89 |
| 5.4.5.1 | set_side_speed() | 90 |

| | | |
|---------|---|-----|
| 5.4.5.2 | update_drive_motors() | 90 |
| 5.5 | include/encoders.h File Reference | 91 |
| 5.5.1 | Detailed Description | 92 |
| 5.5.2 | Macro Definition Documentation | 92 |
| 5.5.2.1 | IME_NUMBER | 93 |
| 5.5.3 | Function Documentation | 93 |
| 5.5.3.1 | get_encoder_ticks() | 93 |
| 5.5.3.2 | get_encoder_velocity() | 94 |
| 5.5.3.3 | init_encoders() | 95 |
| 5.6 | include/lcd.h File Reference | 96 |
| 5.6.1 | Detailed Description | 97 |
| 5.6.2 | Macro Definition Documentation | 97 |
| 5.6.2.1 | BOTTOM_ROW | 97 |
| 5.6.2.2 | TOP_ROW | 98 |
| 5.6.3 | Enumeration Type Documentation | 98 |
| 5.6.3.1 | button_state | 98 |
| 5.6.4 | Function Documentation | 99 |
| 5.6.4.1 | init_main_lcd() | 99 |
| 5.6.4.2 | lcd_clear() | 100 |
| 5.6.4.3 | lcd_get_pressed_buttons() | 101 |
| 5.6.4.4 | lcd_print() | 102 |
| 5.6.4.5 | lcd_printf() | 103 |
| 5.6.4.6 | lcd_set_backlight() | 104 |
| 5.6.4.7 | prompt_confirmation() | 105 |
| 5.7 | include/log.h File Reference | 106 |
| 5.7.1 | Detailed Description | 108 |
| 5.7.2 | Macro Definition Documentation | 108 |
| 5.7.2.1 | DEBUG | 108 |
| 5.7.2.2 | ERROR | 109 |
| 5.7.2.3 | INFO | 109 |

| | | |
|---------|--------------------------------|-----|
| 5.7.2.4 | NONE | 109 |
| 5.7.2.5 | WARNING | 110 |
| 5.7.3 | Function Documentation | 110 |
| 5.7.3.1 | debug() | 110 |
| 5.7.3.2 | error() | 111 |
| 5.7.3.3 | info() | 112 |
| 5.7.3.4 | init_error() | 113 |
| 5.7.3.5 | warning() | 114 |
| 5.8 | include/main.h File Reference | 115 |
| 5.8.1 | Detailed Description | 116 |
| 5.8.2 | Function Documentation | 117 |
| 5.8.2.1 | autonomous() | 117 |
| 5.8.2.2 | initialize() | 118 |
| 5.8.2.3 | initializelO() | 119 |
| 5.8.2.4 | operatorControl() | 119 |
| 5.9 | include/menu.h File Reference | 120 |
| 5.9.1 | Detailed Description | 122 |
| 5.9.2 | Typedef Documentation | 122 |
| 5.9.2.1 | menu_t | 122 |
| 5.9.3 | Enumeration Type Documentation | 122 |
| 5.9.3.1 | menu_type | 123 |
| 5.9.4 | Function Documentation | 123 |
| 5.9.4.1 | calculate_current_display() | 123 |
| 5.9.4.2 | create_menu() | 124 |
| 5.9.4.3 | denint_menu() | 124 |
| 5.9.4.4 | display_menu() | 125 |
| 5.9.4.5 | init_menu_float() | 126 |
| 5.9.4.6 | init_menu_int() | 127 |
| 5.9.4.7 | init_menu_var() | 128 |
| 5.10 | include/ports.h File Reference | 130 |

| | |
|---|-----|
| 5.10.1 Detailed Description | 130 |
| 5.10.2 Macro Definition Documentation | 131 |
| 5.10.2.1 IME_FRONT_RIGHT | 131 |
| 5.10.2.2 MOTOR_BACK_LEFT | 131 |
| 5.10.2.3 MOTOR_BACK_RIGHT | 132 |
| 5.10.2.4 MOTOR_FRONT_LEFT | 132 |
| 5.10.2.5 MOTOR_FRONT_RIGHT | 132 |
| 5.10.2.6 MOTOR_MIDDLE_LEFT | 133 |
| 5.10.2.7 MOTOR_MIDDLE_RIGHT | 133 |
| 5.11 include/slew.h File Reference | 134 |
| 5.11.1 Detailed Description | 135 |
| 5.11.2 Macro Definition Documentation | 135 |
| 5.11.2.1 MOTOR_PORTS | 135 |
| 5.11.2.2 RAMP_PROPORTION | 136 |
| 5.11.2.3 UPDATE_PERIOD_MS | 136 |
| 5.11.3 Function Documentation | 136 |
| 5.11.3.1 deinit_slew() | 137 |
| 5.11.3.2 init_slew() | 137 |
| 5.11.3.3 set_motor_slew() | 139 |
| 5.11.3.4 updateMotors() | 140 |
| 5.12 include/vlib.h File Reference | 141 |
| 5.12.1 Detailed Description | 142 |
| 5.12.2 Function Documentation | 142 |
| 5.12.2.1 calloc_real() | 143 |
| 5.12.2.2 ftoa() | 143 |
| 5.12.2.3 itoa() | 144 |
| 5.12.2.4 reverse() | 146 |
| 5.13 include/vmath.h File Reference | 146 |
| 5.13.1 Detailed Description | 148 |
| 5.13.2 Function Documentation | 148 |

| | | |
|----------|--|-----|
| 5.13.2.1 | <code>cartesian_cord_to_polar()</code> | 148 |
| 5.13.2.2 | <code>cartesian_to_polar()</code> | 149 |
| 5.14 | README.md File Reference | 150 |
| 5.15 | src/auto.c File Reference | 150 |
| 5.15.1 | Detailed Description | 151 |
| 5.15.2 | Function Documentation | 151 |
| 5.15.2.1 | <code>autonomous()</code> | 152 |
| 5.16 | src/battery.c File Reference | 152 |
| 5.17 | src/controller.c File Reference | 152 |
| 5.17.1 | Function Documentation | 153 |
| 5.17.1.1 | <code>get_joystick_cord()</code> | 153 |
| 5.18 | src/drive.c File Reference | 154 |
| 5.18.1 | Function Documentation | 155 |
| 5.18.1.1 | <code>deadspot()</code> | 155 |
| 5.18.1.2 | <code>joystick_interpolate()</code> | 155 |
| 5.18.1.3 | <code>set_side_speed()</code> | 155 |
| 5.18.1.4 | <code>update_drive_motors()</code> | 156 |
| 5.19 | src/encoders.c File Reference | 157 |
| 5.19.1 | Function Documentation | 158 |
| 5.19.1.1 | <code>get_encoder_ticks()</code> | 158 |
| 5.19.1.2 | <code>get_encoder_velocity()</code> | 159 |
| 5.19.1.3 | <code>init_encoders()</code> | 159 |
| 5.20 | src/init.c File Reference | 160 |
| 5.20.1 | Detailed Description | 161 |
| 5.20.2 | Function Documentation | 161 |
| 5.20.2.1 | <code>initialize()</code> | 162 |
| 5.20.2.2 | <code>initialzeIO()</code> | 163 |
| 5.21 | src/lcd.c File Reference | 163 |
| 5.21.1 | Function Documentation | 164 |
| 5.21.1.1 | <code>init_main_lcd()</code> | 164 |

| | | |
|----------|---|-----|
| 5.21.1.2 | <code>lcd_assert()</code> | 165 |
| 5.21.1.3 | <code>lcd_clear()</code> | 167 |
| 5.21.1.4 | <code>lcd_get_pressed_buttons()</code> | 167 |
| 5.21.1.5 | <code>lcd_print()</code> | 169 |
| 5.21.1.6 | <code>lcd_printf()</code> | 170 |
| 5.21.1.7 | <code>lcd_set_backlight()</code> | 170 |
| 5.21.1.8 | <code>prompt_confirmation()</code> | 171 |
| 5.21.2 | Variable Documentation | 172 |
| 5.21.2.1 | <code>lcd_port</code> | 173 |
| 5.22 | <code>src/log.c</code> File Reference | 173 |
| 5.22.1 | Function Documentation | 174 |
| 5.22.1.1 | <code>debug()</code> | 174 |
| 5.22.1.2 | <code>error()</code> | 175 |
| 5.22.1.3 | <code>info()</code> | 176 |
| 5.22.1.4 | <code>init_error()</code> | 177 |
| 5.22.1.5 | <code>log_info()</code> | 178 |
| 5.22.1.6 | <code>warning()</code> | 179 |
| 5.22.2 | Variable Documentation | 180 |
| 5.22.2.1 | <code>log_lcd</code> | 181 |
| 5.22.2.2 | <code>log_level</code> | 181 |
| 5.23 | <code>src/menu.c</code> File Reference | 181 |
| 5.23.1 | Function Documentation | 182 |
| 5.23.1.1 | <code>calculate_current_display()</code> | 182 |
| 5.23.1.2 | <code>create_menu()</code> | 183 |
| 5.23.1.3 | <code>denint_menu()</code> | 184 |
| 5.23.1.4 | <code>display_menu()</code> | 185 |
| 5.23.1.5 | <code>init_menu_float()</code> | 186 |
| 5.23.1.6 | <code>init_menu_int()</code> | 187 |
| 5.23.1.7 | <code>init_menu_var()</code> | 188 |
| 5.24 | <code>src/opcontrol.c</code> File Reference | 189 |

| | |
|--|-----|
| 5.24.1 Detailed Description | 190 |
| 5.24.2 Function Documentation | 190 |
| 5.24.2.1 operatorControl() | 190 |
| 5.25 src/slew.c File Reference | 191 |
| 5.25.1 Function Documentation | 192 |
| 5.25.1.1 deinitslew() | 192 |
| 5.25.1.2 init_slew() | 193 |
| 5.25.1.3 set_motor_slew() | 194 |
| 5.25.1.4 updateMotors() | 195 |
| 5.25.2 Variable Documentation | 196 |
| 5.25.2.1 initialized | 197 |
| 5.25.2.2 motors_set_speeds | 197 |
| 5.25.2.3 mutex | 197 |
| 5.25.2.4 slew | 198 |
| 5.26 src/vlib.c File Reference | 198 |
| 5.26.1 Function Documentation | 199 |
| 5.26.1.1 calloc_real() | 199 |
| 5.26.1.2 ftoa() | 199 |
| 5.26.1.3 itoa() | 200 |
| 5.26.1.4 reverse() | 202 |
| 5.27 src/vmath.c File Reference | 202 |
| 5.27.1 Function Documentation | 203 |
| 5.27.1.1 cartesian_cord_to_polar() | 203 |
| 5.27.1.2 cartesian_to_polar() | 204 |

Chapter 1

InTheZoneA

Team A code for In The Zone

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

| | | |
|-----------------------------|--|----|
| cord | A struct that contains cartesian coordinates | 7 |
| lcd_buttons | State of the lcd buttons | 8 |
| menu_t | Represents the different types of menus | 9 |
| polar_cord | A struct that contains polar coordinates | 15 |

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

| | |
|---|-----|
| include/battery.h | 81 |
| include/controller.h | 81 |
| Controller definitions, macros | |
| include/drive.h | 87 |
| Drive base definitions and enumerations | |
| include/encoders.h | 91 |
| Wrapper around encoder functions | |
| include/lcd.h | 96 |
| LCD wrapper functions and macros | |
| include/log.h | 106 |
| Contains logging functions | |
| include/main.h | 115 |
| Header file for global functions | |
| include/menu.h | 120 |
| Contains menu functionality and abstraction | |
| include/ports.h | 130 |
| Port macros for sensors | |
| include/slew.h | 134 |
| Contains the slew rate controller wrapper for the motors | |
| include/vlib.h | 141 |
| Contains misc helpful functions | |
| include/vmath.h | 146 |
| Vex Specific Math Functions, includes: Cartesian to polar coordinates | |
| src/auto.c | 150 |
| File for autonomous code | |
| src/battery.c | 152 |
| src/controller.c | 152 |
| src/drive.c | 154 |
| src/encoders.c | 157 |
| src/init.c | 160 |
| File for initialization code | |
| src/lcd.c | 163 |
| src/log.c | 173 |
| src/menu.c | 181 |
| src/opcontrol.c | 189 |
| File for operator control code | |

| | |
|--|-----|
| src/ slew.c | 191 |
| src/ vlib.c | 198 |
| src/ vmath.c | 202 |

Chapter 4

Data Structure Documentation

4.1 cord Struct Reference

A struct that contains cartesian coordinates.

```
#include <vmath.h>
```

Data Fields

- float [x](#)
- float [y](#)

4.1.1 Detailed Description

A struct that contains cartesian coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line 31 of file vmath.h.

4.1.2 Field Documentation

4.1.2.1 x

```
float cord::x
```

the x coordinate

Definition at line 33 of file vmath.h.

Referenced by `get_joystick_cord()`.

4.1.2.2 y

```
float cord::y
```

the y coordinate

Definition at line 35 of file vmath.h.

Referenced by `get_joystick_cord()`.

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

- [include/vmath.h](#)

4.2 lcd_buttons Struct Reference

represents the state of the lcd buttons

```
#include <lcd.h>
```

Data Fields

- [button_state left](#)
- [button_state middle](#)
- [button_state right](#)

4.2.1 Detailed Description

represents the state of the lcd buttons

Author

Chris Jerrett

Date

9/9/2017

Definition at line 48 of file lcd.h.

4.2.2 Field Documentation

4.2.2.1 left

```
button_state lcd_buttons::left
```

Definition at line 49 of file lcd.h.

Referenced by `lcd_get_pressed_buttons()`.

4.2.2.2 middle

```
button_state lcd_buttons::middle
```

Definition at line 50 of file lcd.h.

Referenced by `lcd_get_pressed_buttons()`.

4.2.2.3 right

```
button_state lcd_buttons::right
```

Definition at line 51 of file lcd.h.

Referenced by `lcd_get_pressed_buttons()`.

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

- `include/lcd.h`

4.3 menu_t Struct Reference

Represents the different types of menus.

```
#include <menu.h>
```

Data Fields

- int [current](#)
contains the current index of menu.
- unsigned int [length](#)
*contains the length of options char**.*
- int [max](#)
contains the maximum int value of menu. Defaults to minimum int value
- float [max_f](#)
contains the maximum float value of menu. Defaults to minimum int value
- int [min](#)
contains the minimum int value of menu. Defaults to minimum int value
- float [min_f](#)
contains the minimum float value of menu. Defaults to minimum int value
- char ** [options](#)
contains the array of string options.
- char [prompt](#) [16]
contains the prompt to display on the first line. Step is how much the int menu will increase of decrease with each press. Defaults to one
- int [step](#)
contains the step int value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to one
- float [step_f](#)
contains the step float value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to 1.0f
- enum [menu_type](#) type
contains the type of menu.

4.3.1 Detailed Description

Represents the different types of menus.

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#)
[menu_t](#)
[create_menu](#)
[init_menu](#)
[display_menu](#)
[menu_type](#)

Definition at line 62 of file menu.h.

4.3.2 Field Documentation

4.3.2.1 current

```
int menu_t::current
```

contains the current index of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line 136 of file menu.h.

Referenced by calculate_current_display(), and display_menu().

4.3.2.2 length

```
unsigned int menu_t::length
```

contains the length of options char**.

Author

Chris Jerrett

Date

9/8/17

Definition at line 82 of file menu.h.

Referenced by calculate_current_display(), and init_menu_var().

4.3.2.3 max

```
int menu_t::max
```

contains the maximum int value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line 98 of file menu.h.

Referenced by `calculate_current_display()`, `create_menu()`, and `init_menu_int()`.

4.3.2.4 max_f

```
float menu_t::max_f
```

contains the maximum float value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line 122 of file menu.h.

Referenced by `calculate_current_display()`, `create_menu()`, and `init_menu_float()`.

4.3.2.5 min

```
int menu_t::min
```

contains the minimum int value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line 90 of file menu.h.

Referenced by `calculate_current_display()`, `create_menu()`, and `init_menu_int()`.

4.3.2.6 min_f

```
float menu_t::min_f
```

contains the minimum float value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line 114 of file menu.h.

Referenced by `calculate_current_display()`, `create_menu()`, and `init_menu_float()`.

4.3.2.7 options

```
char** menu_t::options
```

contains the array of string options.

Author

Chris Jerrett

Date

9/8/17

Definition at line 75 of file menu.h.

Referenced by `calculate_current_display()`, `denint_menu()`, and `init_menu_var()`.

4.3.2.8 prompt

```
char menu_t::prompt[16]
```

contains the prompt to display on the first line. Step is how much the int menu will increase of decrease with each press. Defaults to one

Author

Chris Jerrett

Date

9/8/17

Definition at line 143 of file menu.h.

Referenced by `create_menu()`, `denint_menu()`, and `display_menu()`.

4.3.2.9 step

```
int menu_t::step
```

contains the step int value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to one

Author

Chris Jerrett

Date

9/8/17

Definition at line 106 of file menu.h.

Referenced by `calculate_current_display()`, `create_menu()`, and `init_menu_int()`.

4.3.2.10 step_f

```
float menu_t::step_f
```

contains the step float value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to 1.0f

Author

Chris Jerrett

Date

9/8/17

Definition at line 130 of file menu.h.

Referenced by `calculate_current_display()`, `create_menu()`, and `init_menu_float()`.

4.3.2.11 type

```
enum menu_type menu_t::type
```

contains the type of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line 68 of file menu.h.

Referenced by `calculate_current_display()`, and `create_menu()`.

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

- `include/menu.h`

4.4 polar_cord Struct Reference

A struct that contains polar coordinates.

```
#include <vmath.h>
```

Data Fields

- float [angle](#)
- float [magnitue](#)

4.4.1 Detailed Description

A struct that contains polar coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line 19 of file vmath.h.

4.4.2 Field Documentation

4.4.2.1 angle

```
float polar_cord::angle
```

the angle of the vector

Definition at line 21 of file vmath.h.

Referenced by cartesian_to_polar().

4.4.2.2 magnitue

```
float polar_cord::magnitue
```

the magnitude of the vector

Definition at line 23 of file vmath.h.

Referenced by cartesian_to_polar().

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

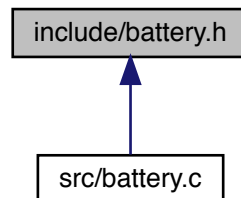
- include/[vmath.h](#)

Chapter 5

File Documentation

5.1 include/battery.h File Reference

This graph shows which files directly or indirectly include this file:

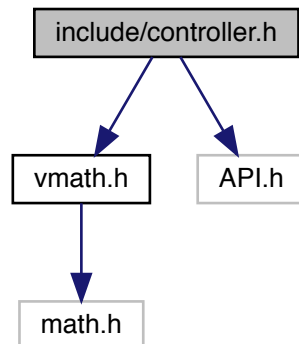


5.2 include/controller.h File Reference

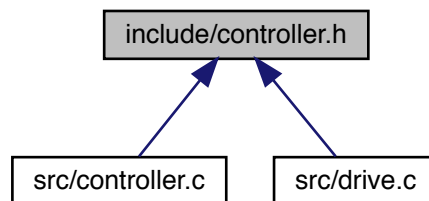
controller definitions, macros

```
#include "vmath.h"  
#include <API.h>
```

Include dependency graph for controller.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define LEFT_JOY_X 4`
the left x joystick on controller
- `#define LEFT_JOY_Y 3`
the left y joystick on controller
- `#define MASTER 1`
the master controller
- `#define PARTNER 2`
the slave/partner controller
- `#define RIGHT_JOY_X 1`
the right x joystick on controller
- `#define RIGHT_JOY_Y 2`
the right y joystick on controller

Enumerations

- enum joystick { RIGHT_JOY, LEFT_JOY }

Represents a joystick on the controller.

Functions

- struct cord get_joystick_cord (enum joystick side, int controller)

5.2.1 Detailed Description

controller definitions, macros

Author

Chris Jerrett

Date

9/9/2017

5.2.2 Macro Definition Documentation

5.2.2.1 LEFT_JOY_X

```
#define LEFT_JOY_X 4
```

the left x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 47 of file controller.h.

Referenced by get_joystick_cord().

5.2.2.2 LEFT_JOY_Y

```
#define LEFT_JOY_Y 3
```

the left y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 54 of file controller.h.

Referenced by get_joystick_cord().

5.2.2.3 MASTER

```
#define MASTER 1
```

the master controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 19 of file controller.h.

Referenced by update_drive_motors().

5.2.2.4 PARTNER

```
#define PARTNER 2
```

the slave/partner controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 26 of file controller.h.

5.2.2.5 RIGHT_JOY_X

```
#define RIGHT_JOY_X 1
```

the right x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 33 of file controller.h.

Referenced by `get_joystick_cord()`.

5.2.2.6 RIGHT_JOY_Y

```
#define RIGHT_JOY_Y 2
```

the right y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 40 of file controller.h.

Referenced by `get_joystick_cord()`.

5.2.3 Enumeration Type Documentation

5.2.3.1 joystick

```
enum joystick
```

Represents a joystick on the controller.

Date

9/10/2017

Author

Chris Jerrett

Enumerator

| | |
|-----------|--------------------|
| RIGHT_JOY | The right joystick |
| LEFT_JOY | The left joystick |

Definition at line 61 of file controller.h.

```

61         {
62     RIGHT_JOY,
63     LEFT_JOY,
64 };

```

5.2.4 Function Documentation

5.2.4.1 get_joystick_cord()

```

struct cord get_joystick_cord (
    enum joystick side,
    int controller )

```

Definition at line 3 of file controller.c.

References LEFT_JOY_X, LEFT_JOY_Y, RIGHT_JOY, RIGHT_JOY_X, RIGHT_JOY_Y, cord::x, and cord::y.

Referenced by update_drive_motors().

```

3                                     {
4     int x;
5     int y;
6     if(side == RIGHT_JOY) {
7         y = joystickGetAnalog(controller, RIGHT_JOY_X);
8         x = joystickGetAnalog(controller, RIGHT_JOY_Y);
9     } else {
10        y = joystickGetAnalog(controller, LEFT_JOY_X);
11        x = joystickGetAnalog(controller, LEFT_JOY_Y);
12    }
13    struct cord c;
14    c.x = x;
15    c.y = y;
16    return c;
17 }

```

Here is the caller graph for this function:

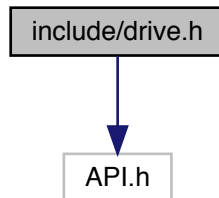


5.3 include/drive.h File Reference

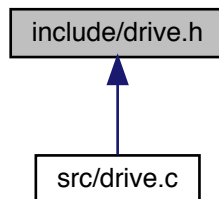
Drive base definitions and enumerations.

```
#include <API.h>
```

Include dependency graph for drive.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define DEADSPOT 30`

Typedefs

- `typedef enum side side_t`
enumeration indication side of the robot.

Enumerations

- `enum side { LEFT, BOTH, RIGHT }`
enumeration indication side of the robot.

Functions

- void `set_side_speed` (`side_t` side, int speed)
sets the speed of one side of the robot.
- void `update_drive_motors` ()
Updates the drive motors during teleop.

5.3.1 Detailed Description

Drive base definitions and enumerations.

Author

Christian Desimone

Date

9/9/2017

5.3.2 Macro Definition Documentation

5.3.2.1 DEADSPOT

```
#define DEADSPOT 30
```

Definition at line 13 of file drive.h.

Referenced by `deadspot()`.

5.3.3 Typedef Documentation

5.3.3.1 side_t

```
typedef enum side side_t
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

9/7/2017 Side can be right, both of left. Contained in side typedef, so enum is unnecessary.

5.3.4 Enumeration Type Documentation

5.3.4.1 side

enum `side`

enumeration indication side of the robot.

Author

Christian Desimone

Date

9/7/2017 Side can be right, both of left. Contained in side typedef, so enum is unnecessary.

Enumerator

| | |
|-------|--|
| LEFT | |
| BOTH | |
| RIGHT | |

Definition at line 21 of file drive.h.

```
21          {  
22      LEFT,  
23      BOTH,  
24      RIGHT  
25 } side_t;
```

5.3.5 Function Documentation

5.3.5.1 set_side_speed()

```
void set_side_speed (  
    side_t side,  
    int speed )
```

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

| | |
|--------------|--|
| <i>side</i> | a side enum which indicates the size. |
| <i>speed</i> | the speed of the side. Can range from -127 - 127 negative being back and positive forwards |

Definition at line 7 of file drive.c.

References BOTH, LEFT, MOTOR_BACK_LEFT, MOTOR_BACK_RIGHT, MOTOR_FRONT_RIGHT, MOTOR_MIDDLE_RIGHT, and RIGHT.

```

7
8  if(side == RIGHT || side == BOTH){
9      motorSet(MOTOR_BACK_RIGHT, speed);
10     motorSet(MOTOR_FRONT_RIGHT, speed);
11     motorSet(MOTOR_MIDDLE_RIGHT, speed);
12 }
13 if(side == LEFT || side == BOTH){
14     motorSet(MOTOR_BACK_LEFT, speed);
15     motorSet(MOTOR_BACK_LEFT, speed);
16     motorSet(MOTOR_BACK_LEFT, speed);
17 }
18 }
```

5.3.5.2 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

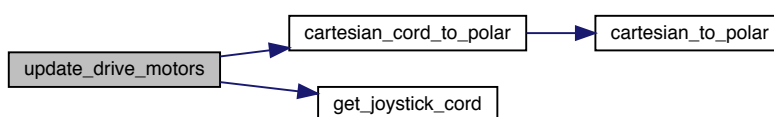
Definition at line 28 of file drive.c.

References cartesian_cord_to_polar(), get_joystick_cord(), MASTER, and RIGHT_JOY.

```

28
29  struct polar_cord cord = cartesian_cord_to_polar(
30      get_joystick_cord(RIGHT_JOY, MASTER));
31 }
```

Here is the call graph for this function:

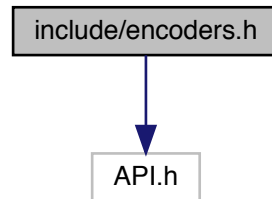


5.4 include/encoders.h File Reference

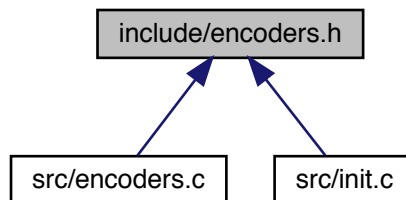
wrapper around encoder functions

```
#include <API.h>
```

Include dependency graph for encoders.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define IME_NUMBER 0`
The number of IMEs. This number is compared against the number detect in `init_encoders`.

Functions

- `int get_encoder_ticks` (unsigned char address)
Gets the encoder ticks since last reset.
- `int get_encoder_velocity` (unsigned char address)
Gets the encoder reads.
- `bool init_encoders` ()
Initializes all motor encoders.

5.4.1 Detailed Description

wrapper around encoder functions

Author

Chris Jerrett

Date

9/9/2017

5.4.2 Macro Definition Documentation

5.4.2.1 IME_NUMBER

```
#define IME_NUMBER 0
```

The number of IMEs. This number is compared against the number detect in `init_encoders`.

See also

[init_encoders\(\)](#)

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#)

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

Referenced by `init_encoders()`.

5.4.3 Function Documentation

5.4.3.1 get_encoder_ticks()

```
int get_encoder_ticks (
    unsigned char address )
```

Gets the encoder ticks since last reset.

Author

Chris Jerrett

Date

9/15/2017

Definition at line 12 of file encoders.c.

```
12                                     {
13     int i = 0;
14     imeGet(address, &i);
15     return i;
16 }
```

5.4.3.2 get_encoder_velocity()

```
int get_encoder_velocity (
    unsigned char address )
```

Gets the encoder reads.

Author

Chris Jerrett

Date

9/15/2017

Definition at line 18 of file encoders.c.

```
18                                     {
19     int i = 0;
20     imeGetVelocity(address, &i);
21     return i;
22 }
```

5.4.3.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#)

Definition at line 4 of file encoders.c.

References [IME_NUMBER](#).

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

```
4      {
5  #ifdef IME_NUMBER
6  return imeInitializeAll() == IME_NUMBER;
7  #else
8  return imeInitializeAll();
9  #endif
10 }
```

Here is the caller graph for this function:

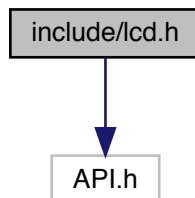


5.5 include/lcd.h File Reference

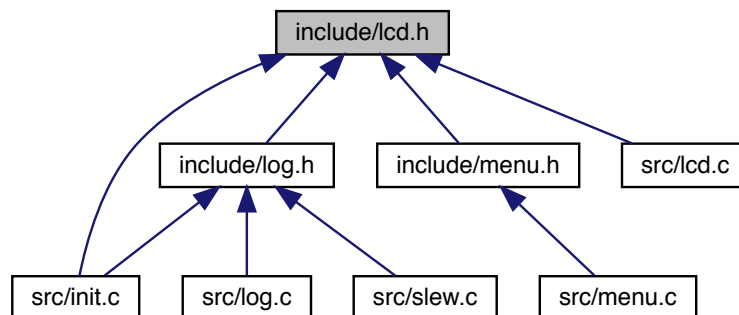
LCD wrapper functions and macros.

```
#include <API.h>
```

Include dependency graph for lcd.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [lcd_buttons](#)
represents the state of the lcd buttons

Macros

- #define [BOTTOM_ROW](#) 2
The bottom row on the lcd screen.
- #define [TOP_ROW](#) 1
The top row on the lcd screen.

Enumerations

- enum `button_state` { `RELEASED` = false, `PRESSED` = true }
Represents the state of a button.

Functions

- void `init_main_lcd` (FILE *lcd)
Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- void `lcd_clear` ()
Clears the lcd.
- `lcd_buttons` `lcd_get_pressed_buttons` ()
Returns the pressed buttons.
- void `lcd_print` (unsigned int line, const char *str)
prints a string to a line on the lcd
- void `lcd_printf` (unsigned int line, const char *format_str,...)
prints a formatted string to a line on the lcd. Similar to printf
- void `lcd_set_backlight` (bool state)
sets the backlight of the lcd
- void `prompt_confirmation` (const char *confirm_text)
Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

5.5.1 Detailed Description

LCD wrapper functions and macros.

Author

Chris Jerrett

Date

9/9/2017

5.5.2 Macro Definition Documentation

5.5.2.1 BOTTOM_ROW

```
#define BOTTOM_ROW 2
```

The bottom row on the lcd screen.

Author

Chris Jerrett

Date

9/9/2017

Definition at line 25 of file lcd.h.

Referenced by `log_info()`.

5.5.2.2 TOP_ROW

```
#define TOP_ROW 1
```

The top row on the lcd screen.

Author

Chris Jerrett

Date

9/9/2017

Definition at line 18 of file lcd.h.

Referenced by display_menu(), and log_info().

5.5.3 Enumeration Type Documentation

5.5.3.1 button_state

```
enum button_state
```

Represents the state of a button.

A button can be pressed or RELEASED. Release is false which is also 0. PRESSED is true or 1.

Author

Chris Jerrett

Date

9/9/2017

Enumerator

| | |
|----------|-------------------|
| RELEASED | A released button |
| PRESSED | A pressed button |

Definition at line 36 of file lcd.h.

```
36      {  
38  RELEASED = false,  
40  PRESSED  = true,  
41  } button_state;
```

5.5.4 Function Documentation

5.5.4.1 init_main_lcd()

```
void init_main_lcd (
    FILE * lcd )
```

Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.

Parameters

| | |
|------------|---------------------------------|
| <i>lcd</i> | the uart port of the lcd screen |
|------------|---------------------------------|

See also

uart1
uart2

Author

Chris Jerrett

Date

9/9/2017

Definition at line 39 of file lcd.c.

References lcd_port.

Referenced by initialize().

```
39                                     {
40     lcdInit(lcd);
41     lcd_port = lcd;
42 }
```

Here is the caller graph for this function:



5.5.4.2 lcd_clear()

```
void lcd_clear ( )
```

Clears the lcd.

Author

Chris Jerrett

Date

9/9/2017

Definition at line 34 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.

```
34      {  
35  lcd_assert();  
36  lcdClear(lcd_port);  
37 }
```

Here is the call graph for this function:



5.5.4.3 lcd_get_pressed_buttons()

```
lcd_buttons lcd_get_pressed_buttons ( )
```

Returns the pressed buttons.

Returns

a struct containing the states of all three buttons.

Author

Chris Jerrett

Date

9/9/2017

See also

[lcd_buttons](#)

Definition at line 20 of file lcd.c.

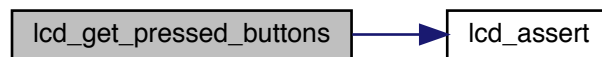
References `lcd_assert()`, `lcd_port`, `lcd_buttons::left`, `lcd_buttons::middle`, `PRESSED`, `RELEASED`, and `lcd_buttons::right`.

Referenced by `display_menu()`, and `prompt_confirmation()`.

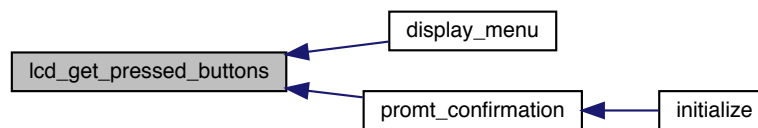
```

20                                     {
21     lcd_assert();
22     unsigned int btn_binary = lcdReadButtons(lcd_port);
23     bool left = btn_binary & 0x1;
24     bool middle = btn_binary & 0x2;
25     bool right = btn_binary & 0x4;
26     lcd_buttons btns;
27     btns.left = left ? PRESSED : RELEASED;
28     btns.middle = middle ? PRESSED : RELEASED;
29     btns.right = right ? PRESSED : RELEASED;
30
31     return btns;
32 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.4.4 lcd_print()

```

void lcd_print (
    unsigned int line,
    const char * str )
```

prints a string to a line on the lcd

Parameters

| | |
|-------------|----------------------|
| <i>line</i> | the line to print on |
| <i>str</i> | string to print |

Author

Chris Jerrett

Date

9/9/2017

Definition at line 44 of file lcd.c.

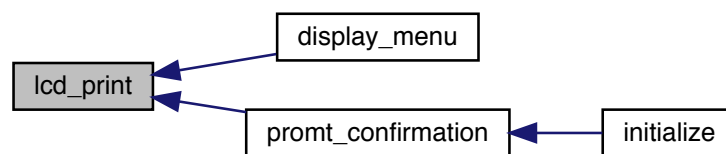
References `lcd_assert()`, and `lcd_port`.Referenced by `display_menu()`, and `prompt_confirmation()`.

```
44                                     {  
45     lcd_assert();  
46     lcdSetText(lcd_port, line, str);  
47 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.4.5 lcd_printf()

```
void lcd_printf (  
    unsigned int line,  
    const char * format_str,  
    ... )
```

prints a formatted string to a line on the lcd. Smilar to printf

Parameters

| | |
|-------------------|-------------------------------|
| <i>line</i> | the line to print on |
| <i>format_str</i> | format string string to print |

Author

Chris Jerrett

Date

9/9/2017

Definition at line 49 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.

```
49                                     {
50     lcd_assert();
51     lcdPrint(lcd_port, line, format_str);
52 }
```

Here is the call graph for this function:

**5.5.4.6 lcd_set_backlight()**

```
void lcd_set_backlight (
    bool state )
```

sets the backlight of the lcd

Parameters

| | |
|--------------|--|
| <i>state</i> | a boolean representing the state of the backlight. true = on, false = off. |
|--------------|--|

Author

Chris Jerrett

Date

9/9/2017

Definition at line 54 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.

```
54                                     {  
55     lcd_assert();  
56     lcdSetBacklight(lcd_port, state);  
57 }
```

Here is the call graph for this function:

**5.5.4.7 `prompt_confirmation()`**

```
void prompt_confirmation (  
    const char * confirm_text )
```

Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Parameters

| | |
|---------------------------|-----------------------------------|
| <code>confirm_text</code> | the text for the user to confirm. |
|---------------------------|-----------------------------------|

Author

Chris Jerrett

Date

9/9/2017

Definition at line 59 of file lcd.c.

References `lcd_assert()`, `lcd_get_pressed_buttons()`, `lcd_print()`, and `PRESSED`.

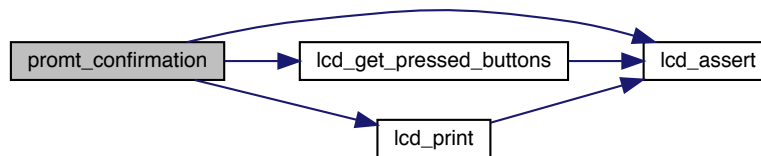
Referenced by `initialize()`.

```

59                                     {
60     lcd_assert();
61     lcd_print(1, confirm_text);
62     while(lcd_get_pressed_buttons().middle != PRESSED){
63         delay(200);
64     }
65 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.6 include/log.h File Reference

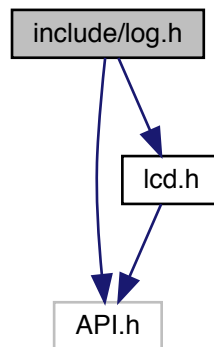
Contains logging functions.

```

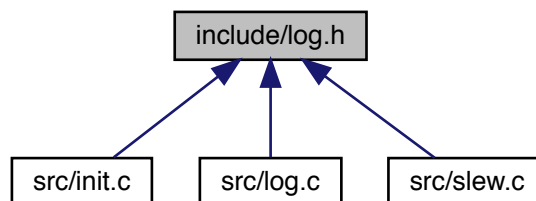
#include <API.h>
#include "lcd.h"

```


Include dependency graph for log.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define` `DEBUG` 4
logging only info debug. most verbose level
- `#define` `ERROR` 1
logging only errors. Also displays error to lcd
- `#define` `INFO` 3
logging only info messages and higher.
- `#define` `NONE` 0
No logging. Should be used in competition to reduce serial communication.
- `#define` `WARNING` 2
logs errors and warnings. Also displays error to lcd

Functions

- void `debug` (const char *debug_message)
prints a info message
- void `error` (const char *error_message)
prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE
- void `info` (const char *info_message)
prints a info message
- void `init_error` (bool use_lcd, FILE *lcd)
Initializes the error lcd system Only required if using lcd.
- void `warning` (const char *warning_message)
prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

5.6.1 Detailed Description

Contains logging functions.

Author

Chris Jerrett

Date

9/16/2017

5.6.2 Macro Definition Documentation

5.6.2.1 DEBUG

```
#define DEBUG 4
```

logging only info debug. most verbose level

Author

Chris Jerrett

Date

9/10/17

Definition at line 51 of file log.h.

5.6.2.2 ERROR

```
#define ERROR 1
```

logging only errors. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line 28 of file log.h.

Referenced by debug(), and info().

5.6.2.3 INFO

```
#define INFO 3
```

logging only info messages and higher.

Author

Chris Jerrett

Date

9/10/17

Definition at line 43 of file log.h.

5.6.2.4 NONE

```
#define NONE 0
```

No logging. Should be used in competition to reduce serial communication.

Author

Chris Jerrett

Date

9/10/17

Definition at line 20 of file log.h.

Referenced by error().

5.6.2.5 WARNING

```
#define WARNING 2
```

logs errors and warnings. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line 36 of file log.h.

Referenced by warning().

5.6.3 Function Documentation

5.6.3.1 debug()

```
void debug (
    const char * debug_message )
```

prints a info message

Only will print and display if log_level is greater than info

See also

[log_level](#)

Parameters

| | |
|----------------------|-------------|
| <i>debug_message</i> | the message |
|----------------------|-------------|

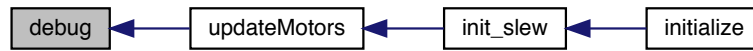
Definition at line 37 of file log.c.

References ERROR, and log_level.

Referenced by updateMotors().

```
37                                     {
38     if(log_level>ERROR) {
39         printf("[INFO]: %s\n", debug_message);
40     }
41 }
```

Here is the caller graph for this function:



5.6.3.2 error()

```
void error (  
    const char * error_message )
```

prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

[log_level](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

| | |
|----------------------|-------------|
| <i>error_message</i> | the message |
|----------------------|-------------|

Definition at line 21 of file log.c.

References [log_info\(\)](#), [log_level](#), and [NONE](#).

```
21                                     {  
22     if(log_level>NONE)  
23         log_info("ERROR", error_message);  
24 }
```

Here is the call graph for this function:



5.6.3.3 info()

```
void info (
    const char * info_message )
```

prints a info message

Only will print and display if log_level is greater than ERROR

See also

[log_level](#)

Parameters

| | |
|---------------------|-------------|
| <i>info_message</i> | the message |
|---------------------|-------------|

Definition at line 31 of file log.c.

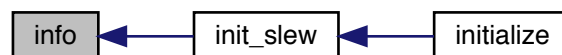
References `ERROR`, and `log_level`.

Referenced by `init_slew()`.

```

31                                     {
32     if(log_level>ERROR) {
33         printf("[INFO]: %s\n", info_message);
34     }
35 }
```

Here is the caller graph for this function:



5.6.3.4 init_error()

```
void init_error (
    bool use_lcd,
    FILE * lcd )
```

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

| | |
|----------------|------------------------|
| <i>use_lcd</i> | whether to use the lcd |
| <i>lcd</i> | the lcd |

Definition at line 6 of file log.c.

References `log_lcd`.

Referenced by `initialize()`.

```
6                                     {
7     if (use_lcd) {
8         lcdInit (lcd);
9         log_lcd = lcd;
10    }
11 }
```

Here is the caller graph for this function:



5.6.3.5 warning()

```
void warning (
    const char * warning_message )
```

prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

[log_level](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

| | |
|------------------------|-------------|
| <i>warning_message</i> | the message |
|------------------------|-------------|

Definition at line 26 of file log.c.

References [log_info\(\)](#), [log_level](#), and [WARNING](#).

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

```
26                                     {
27     if(log_level>WARNING)
28         log_info("WARNING", warning_message);
29 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

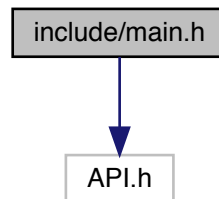


5.7 include/main.h File Reference

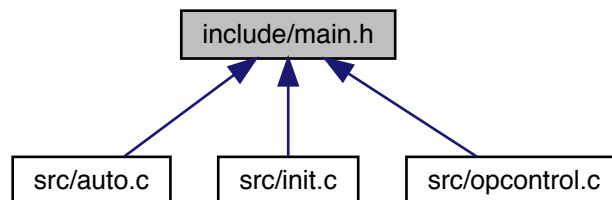
Header file for global functions.

```
#include <API.h>
```

Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [autonomous](#) ()
- void [initialize](#) ()
- void [initializeIO](#) ()
- void [operatorControl](#) ()

5.7.1 Detailed Description

Header file for global functions.

Any experienced C or C++ programmer knows the importance of header files. For those who do not, a header file allows multiple files to reference functions in other files without necessarily having to see the code (and therefore

causing a multiple definition). To make a function in "opcontrol.c", "auto.c", "main.c", or any other C file visible to the core implementation files, prototype it here.

This file is included by default in the predefined stubs in each VEX Cortex PROS Project.

Copyright (c) 2011-2014, Purdue University ACM SIG BOTS. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Purdue University ACM SIG BOTS nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL PURDUE UNIVERSITY ACM SIG BOTS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Purdue Robotics OS contains FreeRTOS (<http://www.freertos.org>) whose source code may be obtained from <http://sourceforge.net/projects/freertos/files/> or on request.

5.7.2 Function Documentation

5.7.2.1 autonomous()

```
void autonomous ( )
```

Runs the user autonomous code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is lost, the autonomous task will be stopped by the kernel. Re-enabling the robot will restart the task, not re-start it from where it left off.

Code running in the autonomous task cannot access information from the VEX Joystick. However, the autonomous function can be invoked from another task if a VEX Competition Switch is not available, and it can access joystick information if called in this way.

The autonomous task may exit, unlike [operatorControl\(\)](#) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

Definition at line 29 of file auto.c.

```
29         {
30     }
```

5.7.2.2 initialize()

```
void initialize ( )
```

Runs user initialization code. This function will be started in its own task with the default priority and stack size once when the robot is starting up. It is possible that the VEXnet communication link may not be fully established at this time, so reading from the VEX Joystick may fail.

This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global variables, and IMEs.

This function must exit relatively promptly, or the [operatorControl\(\)](#) and [autonomous\(\)](#) tasks will not start. An autonomous mode selection menu like the `pre_auton()` in other environments can be implemented in this task if desired.

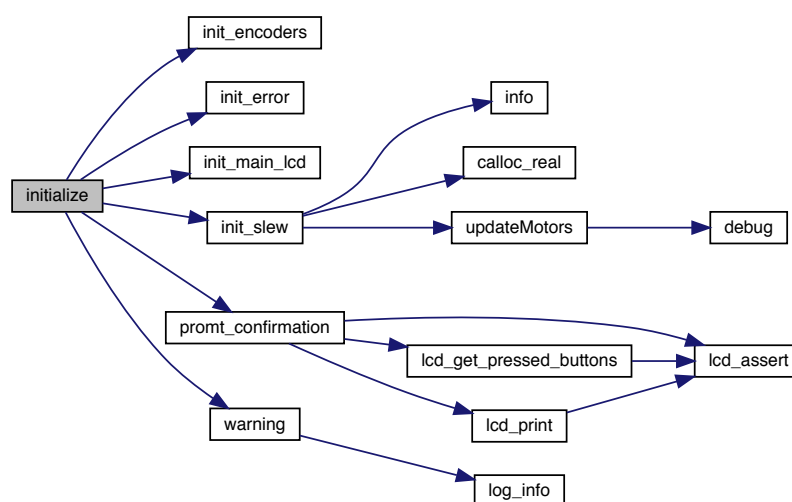
Definition at line 45 of file `init.c`.

References `init_encoders()`, `init_error()`, `init_main_lcd()`, `init_slew()`, `prompt_confirmation()`, and `warning()`.

```

45     {
46     setTeamName("9228A");
47     init_slew();
48     init_main_lcd(uart1);
49     init_error(true, uart2);
50     if(!init_encoders()) {
51         prompt_confirmation("Check IME");
52         warning("CHECK IME");
53     }
54
55     if(powerLevelBackup()/1000 == 0) {
56         prompt_confirmation("Check Backup");
57         warning("Checkbackup bat");
58     }
59 }
```

Here is the call graph for this function:



5.7.2.3 initializeIO()

```
void initializeIO ( )
```

Runs pre-initialization code. This function will be started in kernel mode one time while the VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.

The purpose of this function is solely to set the default pin modes (`pinMode()`) and port states (`digitalWrite()`) of limit switches, push buttons, and solenoids. It can also safely configure a UART port (`usartOpen()`) but cannot set up an LCD (`lcdInit()`).

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

```
28      {  
29          watchdogInit();  
30      }
```

5.7.2.4 operatorControl()

```
void operatorControl ( )
```

Runs the user operator control code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the operator control mode. If the robot is disabled or communications is lost, the operator control task will be stopped by the kernel. Re-enabling the robot will restart the task, not resume it from where it left off.

If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will run the operator control task. Be warned that this will also occur if the VEX Cortex is tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.

Code running in this task can take almost any action, as the VEX Joystick is available and the scheduler is operational. However, proper use of `delay()` or `taskDelayUntil()` is highly recommended to give other tasks (including system tasks such as updating LCDs) time to run.

This task should never exit; it should end with some kind of infinite loop, even if empty.

Definition at line 33 of file `opcontrol.c`.

References `set_motor_slew()`.

```
33      {  
34          while (1) {  
35              set_motor_slew(2, 100);  
36              delay(20);  
37          }  
38      }
```

Here is the call graph for this function:

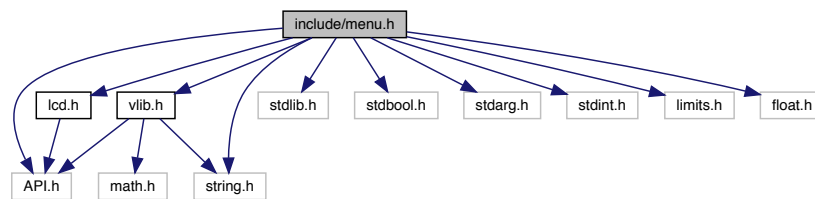


5.8 include/menu.h File Reference

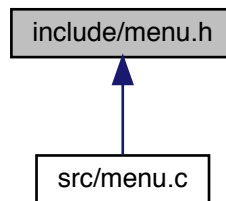
Contains menu functionality and abstraction.

```
#include "lcd.h"
#include "API.h"
#include <string.h>
#include <limits.h>
#include <float.h>
#include <vlib.h>
```

Include dependency graph for menu.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `menu_t`
Represents the different types of menus.

Typedefs

- typedef struct `menu_t menu_t`
Represents the different types of menus.

Enumerations

- enum `menu_type` { `INT_TYPE`, `FLOAT_TYPE`, `STRING_TYPE` }
Represents the different types of menus.

Functions

- static void [calculate_current_display](#) (char *rtn, [menu_t](#) *menu)
Static function that calculates the string from menu.
- static [menu_t](#) * [create_menu](#) (enum [menu_type](#) type, const char *prompt)
Static function that handles creation of menu. Menu must be freed or will cause memory leak
- void [denint_menu](#) ([menu_t](#) *menu)
Destroys a menu Menu must be freed or will cause memory leak
- int [display_menu](#) ([menu_t](#) *menu)
Displays a menu context, but does not display. Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.
- [menu_t](#) * [init_menu_float](#) (enum [menu_type](#) type, float min, float max, float step, char *prompt)
Creates a menu context, but does not display. Menu must be freed or will cause memory leak!
- [menu_t](#) * [init_menu_int](#) (enum [menu_type](#) type, int min, int max, int step, char *prompt)
Creates a menu context, but does not display. Menu must be freed or will cause memory leak
- [menu_t](#) * [init_menu_var](#) (enum [menu_type](#) type, unsigned int nums, char *prompt, char *options,...)
Creates a menu context, but does not display. Menu must be freed or will cause memory leak

5.8.1 Detailed Description

Contains menu functionality and abstraction.

Author

Chris Jerrett

Date

9/9/2017

5.8.2 Typedef Documentation

5.8.2.1 [menu_t](#)

```
typedef struct menu\_t menu\_t
```

Represents the different types of menus.

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#)
[menu_t](#)
[create_menu](#)
[init_menu](#)
[display_menu](#)
[menu_type](#)

5.8.3 Enumeration Type Documentation

5.8.3.1 menu_type

enum [menu_type](#)

Represents the different types of menus.

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#)
[menu_t](#)
[create_menu](#)
[init_menu](#)
[display_menu](#)
[menu_type](#)

Enumerator

| | |
|-------------|--|
| INT_TYPE | Menu type allowing user to select a integer. The integer type menu has a max, min and a step value. Each step is calculated. Will return the index of the selected value. Example: User goes forwards twice then it will return 2. |
| FLOAT_TYPE | Menu type allowing user to select a float The float type menu has a max, min and a step value. Each step is calculated. Will return the index of the selected value. Example: User goes forwards twice then it will return 2. |
| STRING_TYPE | Menu type allowing user to select a string from a array of strings. Will return the index of the selected value. Example: User goes forwards twice then it will return 2. |

Definition at line 28 of file menu.h.

```
28         {  
35     INT_TYPE,  
42     FLOAT_TYPE,  
48     STRING_TYPE  
49 };
```

5.8.4 Function Documentation

5.8.4.1 calculate_current_display()

```
static void calculate_current_display (
    char * rtn,
    menu_t * menu ) [static]
```

Static function that calculates the string from menu.

Parameters

| | |
|-------------|---|
| <i>rtn</i> | the string to be written to |
| <i>menu</i> | the menu for prompt to be calculated from |

Author

Chris Jerrett

Date

9/8/17

5.8.4.2 create_menu()

```
static menu_t* create_menu (
    enum menu_type type,
    const char * prompt ) [static]
```

Static function that handles creation of menu. *Menu must be freed or will cause memory leak*

Author

Chris Jerrett

Date

9/8/17

5.8.4.3 denint_menu()

```
void denint_menu (
    menu_t * menu )
```

Destroys a menu *Menu must be freed or will cause memory leak*

Parameters

| | |
|-------------|------------------|
| <i>menu</i> | the menu to free |
|-------------|------------------|

See also

[menu](#)

Author

Chris Jerrett

Date

9/8/17

Definition at line 92 of file menu.c.

References [menu_t::options](#), and [menu_t::prompt](#).

```
92     {
93     free(menu->prompt);
94     if(menu->options != NULL) free(menu->options);
95     free(menu);
96 }
```

5.8.4.4 display_menu()

```
int display_menu (
    menu_t * menu )
```

Displays a menu context, but does not display. *Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.*

Parameters

| | |
|-------------|---------------------|
| <i>menu</i> | the menu to display |
|-------------|---------------------|

See also

[menu_type](#)

Author

Chris Jerrett

Date

9/8/17

Definition at line 74 of file menu.c.

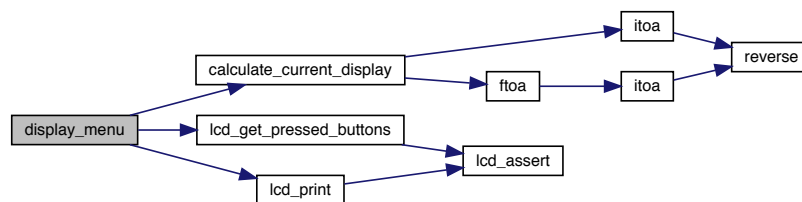
References `calculate_current_display()`, `menu_t::current`, `lcd_get_pressed_buttons()`, `lcd_print()`, `PRESSED`, `menu_t::prompt`, `RELEASED`, and `TOP_ROW`.

```

74     {
75         lcd_print(TOP_ROW, menu->prompt);
76         //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
77         while(lcd_get_pressed_buttons().middle == RELEASED && !isEnabled()) {
78             char val[16];
79             calculate_current_display(val, menu);
80
81             if(lcd_get_pressed_buttons().right == PRESSED) {
82                 menu->current += 1;
83             }
84             if(lcd_get_pressed_buttons().left == PRESSED) {
85                 menu->current -= 1;
86             }
87             delay(500);
88         }
89         return menu->current;
90     }

```

Here is the call graph for this function:

5.8.4.5 `init_menu_float()`

```

menu_t* init_menu_float (
    enum menu_type type,
    float min,
    float max,
    float step,
    char * prompt )

```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak!*

Parameters

| | |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

[menu_type](#)

Parameters

| | |
|---------------|-------------------------------|
| <i>min</i> | the minimum value |
| <i>max</i> | the maximum value |
| <i>step</i> | the step value |
| <i>prompt</i> | the prompt to display to user |

Author

Chris Jerrett

Date

9/8/17

Definition at line 39 of file menu.c.

References [create_menu\(\)](#), [menu_t::max_f](#), [menu_t::min_f](#), and [menu_t::step_f](#).

```
39                                     {
40  menu_t* menu = create_menu(type, prompt);
41  menu->min_f = min;
42  menu->max_f = max;
43  menu->step_f = step;
44  return menu;
45 }
```

Here is the call graph for this function:



5.8.4.6 init_menu_int()

```
menu_t* init_menu_int (
    enum menu_type type,
    int min,
    int max,
    int step,
    char * prompt )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

| | |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

[menu_type](#)

Parameters

| | |
|---------------|-------------------------------|
| <i>min</i> | the minimum value |
| <i>max</i> | the maximum value |
| <i>step</i> | the step value |
| <i>prompt</i> | the prompt to display to user |

Author

Chris Jerrett

Date

9/8/17

Definition at line 31 of file menu.c.

References `create_menu()`, `menu_t::max`, `menu_t::min`, and `menu_t::step`.

```

31
32  menu_t* menu = create_menu(type, prompt);
33  menu->min = min;
34  menu->max = max;
35  menu->step = step;
36  return menu;
37 }
```

Here is the call graph for this function:



5.8.4.7 init_menu_var()

```

menu_t* init_menu_var (
    enum menu_type type,
    unsigned int nums,
    char * prompt,
    char * options,
    ... )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

| | |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

[menu_type](#)

Parameters

| | |
|----------------|---|
| <i>nums</i> | the number of elements passed to function |
| <i>prompt</i> | the prompt to display to user |
| <i>options</i> | the options to display for user |

Author

Chris Jerrett

Date

9/8/17

Definition at line 17 of file menu.c.

References [create_menu\(\)](#), [menu_t::length](#), and [menu_t::options](#).

```
17                                     {
18  menu_t* menu = create_menu(type, prompt);
19  va_list values;
20  char **options_array = (char**)malloc(sizeof(char*) * nums);
21  va_start(values, options);
22  for(unsigned int i = 0; i < nums; i++){
23      options_array[i] = va_arg(values, char*);
24  }
25  va_end(values);
26  menu->options = options_array;
27  menu->length = nums;
28  return menu;
29 }
```

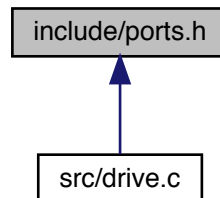
Here is the call graph for this function:



5.9 include/ports.h File Reference

contains port macros for sensors

This graph shows which files directly or indirectly include this file:



Macros

- `#define` [IME_FRONT_RIGHT](#) 0
Number of integrated motor encoders Used when checking to see if all imes are plugged in.
- `#define` [MOTOR_BACK_LEFT](#) 5
Back left drive motor of robot base.
- `#define` [MOTOR_BACK_RIGHT](#) 4
Back right drive motor of robot base.
- `#define` [MOTOR_FRONT_LEFT](#) 1
Front left drive motor of robot base.
- `#define` [MOTOR_FRONT_RIGHT](#) 0
Front right drive motor of robot base.
- `#define` [MOTOR_MIDDLE_LEFT](#) 3
Middle left drive motor of robot base.
- `#define` [MOTOR_MIDDLE_RIGHT](#) 2
Middle right drive motor of robot base.

5.9.1 Detailed Description

contains port macros for sensors

Author

Chris Jerrett

Date

9/9/2017

5.9.2 Macro Definition Documentation

5.9.2.1 IME_FRONT_RIGHT

```
#define IME_FRONT_RIGHT 0
```

Number of integrated motor encoders Used when checking to see if all imes are plugged in.

See also

[init_encoders](#)

Author

Christian Desimone

Date

9/7/2017

Definition at line 18 of file ports.h.

5.9.2.2 MOTOR_BACK_LEFT

```
#define MOTOR_BACK_LEFT 5
```

Back left drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 59 of file ports.h.

Referenced by `set_side_speed()`.

5.9.2.3 MOTOR_BACK_RIGHT

```
#define MOTOR_BACK_RIGHT 4
```

Back right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 53 of file ports.h.

Referenced by set_side_speed().

5.9.2.4 MOTOR_FRONT_LEFT

```
#define MOTOR_FRONT_LEFT 1
```

Front left drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 32 of file ports.h.

5.9.2.5 MOTOR_FRONT_RIGHT

```
#define MOTOR_FRONT_RIGHT 0
```

Front right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 25 of file ports.h.

Referenced by set_side_speed().

5.9.2.6 MOTOR_MIDDLE_LEFT

```
#define MOTOR_MIDDLE_LEFT 3
```

Middle left drive motor of robot base.

Date

9/7/2017

Author

Christian Desimone

Definition at line 46 of file ports.h.

5.9.2.7 MOTOR_MIDDLE_RIGHT

```
#define MOTOR_MIDDLE_RIGHT 2
```

Middle right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 39 of file ports.h.

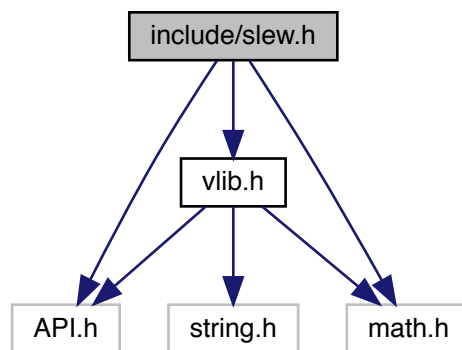
Referenced by `set_side_speed()`.

5.10 include/slew.h File Reference

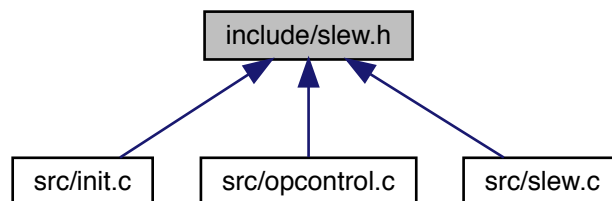
Contains the slew rate controller wrapper for the motors.

```
#include <API.h>
#include <math.h>
#include <vlib.h>
```

Include dependency graph for slew.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define MOTOR_PORTS 12`
The number of motor ports on the robot.
- `#define RAMP_PROPORTION 2`
proportion defining how quickly the motor should converge on the correct value. higher value leads to slower convergence
- `#define UPDATE_PERIOD_MS 25`
How frequently to update the motors, in milliseconds.

Functions

- void `deinitslew` ()
Deinitializes the slew rate controller and frees memory.
- void `init_slew` ()
Initializes the slew rate controller.
- void `set_motor_slew` (int motor, int speed)
Sets motor speed wrapped inside the slew rate controller.
- void `updateMotors` ()
Closes the distance between the desired motor value and the current motor value by half for each motor.

5.10.1 Detailed Description

Contains the slew rate controller wrapper for the motors.

Author

Chris Jerrett

Date

9/14/17

5.10.2 Macro Definition Documentation

5.10.2.1 MOTOR_PORTS

```
#define MOTOR_PORTS 12
```

The number of motor ports on the robot.

Author

Christian DeSimone

Date

9/14/17

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

Referenced by `init_slew()`, and `updateMotors()`.

5.10.2.2 RAMP_PROPORTION

```
#define RAMP_PROPORTION 2
```

proportion defining how quickly the motor should converge on the correct value. higher value leads to slower convergence

Author

Chris Jerrett

Date

9/14/17

Definition at line 34 of file slew.h.

Referenced by updateMotors().

5.10.2.3 UPDATE_PERIOD_MS

```
#define UPDATE_PERIOD_MS 25
```

How frequently to update the motors, in milliseconds.

Author

Chris Jerrett

Date

9/14/17

Definition at line 20 of file slew.h.

Referenced by init_slew().

5.10.3 Function Documentation

5.10.3.1 deinit_slew()

```
void deinit_slew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line 62 of file slew.c.

References `motors_set_speeds`, and `slew`.

```
62     {
63     free(motors_set_speeds);
64     taskDelete(slew);
65 }
```

5.10.3.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

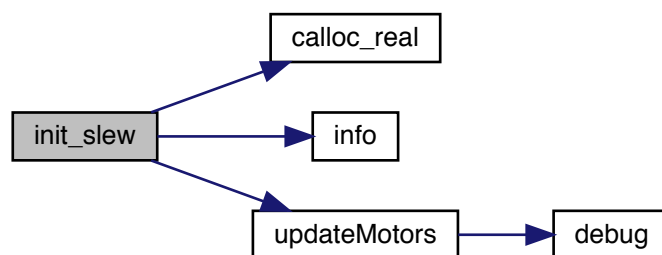
Definition at line 49 of file slew.c.

References `calloc_real()`, `info()`, `initialized`, `MOTOR_PORTS`, `mutex`, `slew`, `UPDATE_PERIOD_MS`, and `updateMotors()`.

Referenced by `initialize()`.

```
49     {
50     info("Init Slew");
51     calloc_real(MOTOR_PORTS, sizeof(char));
52     mutex = mutexCreate();
53     slew = taskRunLoop(updateMotors, UPDATE_PERIOD_MS);
54     initialized = true;
55 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.3.3 set_motor_slew()

```
void set_motor_slew (
    int motor,
    int speed )
```

Sets motor speed wrapped inside the slew rate controller.

Parameters

| | |
|--------------|--|
| <i>motor</i> | the motor port to use |
| <i>speed</i> | the speed to use, between -127 and 127 |

Author

Chris Jerrett

Date

9/14/17

Definition at line 67 of file slew.c.

References `motors_set_speeds`, and `mutex`.

Referenced by `operatorControl()`.

```
67                                     {  
68     if(mutexTake(mutex, 100)) {  
69         motors_set_speeds[motor] = speed;  
70         mutexGive(mutex);  
71     }  
72 }
```

Here is the caller graph for this function:

**5.10.3.4 updateMotors()**

```
void updateMotors ( )
```

Closes the distance between the desired motor value and the current motor value by half for each motor.

Author

Chris Jerrett

Date

9/14/17

Definition at line 31 of file slew.c.

References `debug()`, `MOTOR_PORTS`, `motors_set_speeds`, `mutex`, and `RAMP_PROPORTION`.

Referenced by `init_slew()`.

```

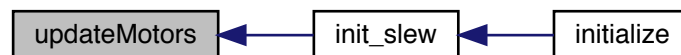
31     {
32     //Take back half approach
33     //Not linear but equal to setSpeed(1-(1/2)^x)
34     if(mutexTake(mutex, 10)) {
35         for(int i = 0; i < MOTOR_PORTS; i++) {
36             char set_speed = motors_set_speeds[i];
37             char curr_speed = motorGet(i);
38             char diff = set_speed - curr_speed;
39             int n = (int) curr_speed + ceil(diff/(float)RAMP_PROPORTION);
40             char c[16];
41             sprintf(c, "Set Motor %d: %d", i, n);
42             debug(c);
43             motorSet(i, n);
44         }
45         mutexGive(mutex);
46     }
47 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.11 include/vlib.h File Reference

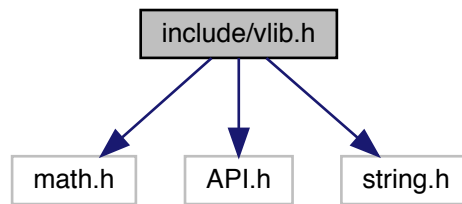
Contains misc helpful functions.

```

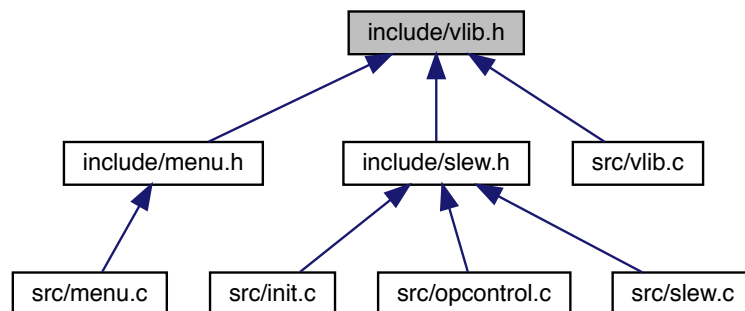
#include <math.h>
#include <API.h>
#include <string.h>

```


Include dependency graph for vlib.h:



This graph shows which files directly or indirectly include this file:



Functions

- void * [calloc_real](#) (size_t elements, size_t size)
- void [ftoa](#) (float a, char *buffer, int precision)
converts a float to string.
- int [itoa](#) (int a, char *buffer, int digits)
converts a int to string.
- void [reverse](#) (char *str, int len)
reverses a string 'str' of length 'len'

5.11.1 Detailed Description

Contains misc helpful functions.

Author

Chris Jerrett

Date

9/9/2017

5.11.2 Function Documentation

5.11.2.1 calloc_real()

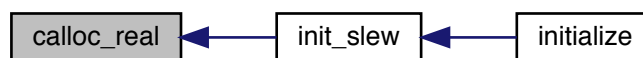
```
void* calloc_real (
    size_t elements,
    size_t size )
```

Definition at line 53 of file vlib.c.

Referenced by init_slew().

```
53 {
54     void *mem = malloc(elements * size);
55     //This is not a error. Bad ATOM!
56     memset(mem, 0, elements * size);
57     return mem;
58 }
```

Here is the caller graph for this function:



5.11.2.2 ftoa()

```
void ftoa (
    float a,
    char * buffer,
    int precision )
```

converts a float to string.

Parameters

| | |
|------------------|--|
| <i>a</i> | the float |
| <i>buffer</i> | the string the float will be written to. |
| <i>precision</i> | digits after the decimal to write |

Definition at line 30 of file vlib.c.

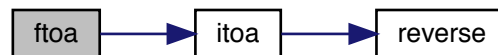
References itoa().

Referenced by calculate_current_display().

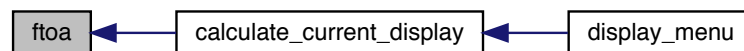
```

30                                     {
31     // Extract integer part
32     int ipart = (int)a;
33
34     // Extract floating part
35     float fpart = a - (float)ipart;
36
37     // convert integer part to string
38     int i = itoa(ipart, buffer, 0);
39
40     // check for display option after point
41     if(precision != 0) {
42         buffer[i] = '.'; // add dot
43
44         // Get the value of fraction part up to given num.
45         // of points after dot. The third parameter is needed
46         // to handle cases like 233.007
47         fpart = fpart * pow(10, precision);
48
49         itoa((int)fpart, buffer + i + 1, precision);
50     }
51 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.3 itoa()

```

int itoa (
    int a,
    char * buffer,
    int digits )
```

converts a int to string.

Parameters

| | |
|---------------|--|
| <i>a</i> | the integer |
| <i>buffer</i> | the string the int will be written to. |
| <i>digits</i> | the number of digits to be written |

Returns

the digits

Author

Chris Jerrett

Date

9/9/2017

Definition at line 13 of file vlib.c.

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

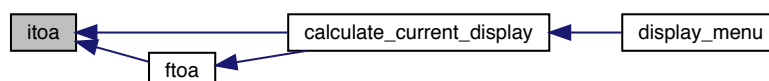
Referenced by [calculate_current_display\(\)](#), and [ftoa\(\)](#).

```
13     {
14     int i = 0;
15     while (a) {
16         buffer[i++] = (a%10) + '0';
17         a = a/10;
18     }
19
20     // If number of digits required is more, then
21     // add 0s at the beginning
22     while (i < digits)
23         buffer[i++] = '0';
24
25     reverse(buffer, i);
26     buffer[i] = '\0';
27     return i;
28 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.2.4 reverse()

```
void reverse (
    char * str,
    int len )
```

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

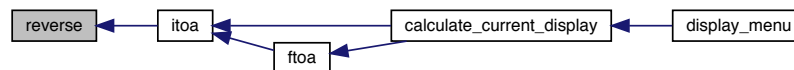
| | |
|------------|-----------------------|
| <i>str</i> | the string to reverse |
| <i>len</i> | the length |

Definition at line 3 of file vlib.c.

Referenced by itoa().

```
3      {
4      int i=0, j=len-1, temp;
5      while (i<j) {
6          temp = str[i];
7          str[i] = str[j];
8          str[j] = temp;
9          i++; j--;
10     }
11 }
```

Here is the caller graph for this function:

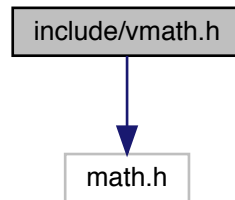


5.12 include/vmath.h File Reference

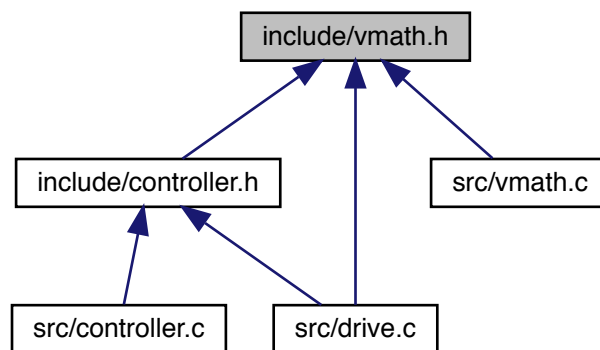
Vex Specific Math Functions, includes: Cartesian to polar coordinates.

```
#include <math.h>
```

Include dependency graph for vmath.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [cord](#)
A struct that contains cartesian coordinates.
- struct [polar_cord](#)
A struct that contains polar coordinates.

Functions

- struct [polar_cord](#) [cartesian_cord_to_polar](#) (struct [cord](#) cords)
Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- struct [polar_cord](#) [cartesian_to_polar](#) (float x, float y)
Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

5.12.1 Detailed Description

Vex Specific Math Functions, includes: Cartesian to polar coordinates.

Author

Christian Desimone
Chris Jerrett

Date

9/9/2017

5.12.2 Function Documentation

5.12.2.1 cartesian_cord_to_polar()

```
struct polar_cord cartesian_cord_to_polar (  
    struct cord cords )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

| | |
|--------------|---------------------|
| <i>cords</i> | the cartesian cords |
|--------------|---------------------|

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)
[cord](#)

Definition at line 33 of file vmath.c.

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

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

```

33                                     {
34     return cartesian_to_polar(cords.x, cords.y);
35 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.2 cartesian_to_polar()

```

struct polar_cord cartesian_to_polar (
    float x,
    float y )

```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

| | |
|----------|--|
| <i>x</i> | float value of the x cartesian coordinate. |
| <i>y</i> | float value of the y cartesian coordinate. |

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)

Definition at line 3 of file vmath.c.

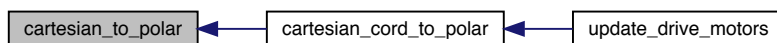
References `polar_cord::angle`, and `polar_cord::magnitue`.

Referenced by `cartesian_cord_to_polar()`.

```

3                                     {
4   float degree = 0;
5   double magnitue = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
6
7   if(x < 0){
8       degree += 180.0;
9   }
10  else if(x > 0 && y < 0){
11      degree += 360.0;
12  }
13
14  if(x != 0 && y != 0){
15      degree += atan((float)y / (float)x);
16  }
17  else if(x == 0 && y > 0){
18      degree = 90.0;
19  }
20  else if(y == 0 && x < 0){
21      degree = 180.0;
22  }
23  else if(x == 0 && y < 0){
24      degree = 270.0;
25  }
26
27  struct polar_cord p;
28  p.angle = degree;
29  p.magnitue = magnitue;
30  return p;
31 }
```

Here is the caller graph for this function:



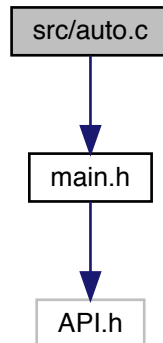
5.13 README.md File Reference

5.14 src/auto.c File Reference

File for autonomous code.

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

Include dependency graph for auto.c:



Functions

- void [autonomous](#) ()

5.14.1 Detailed Description

File for autonomous code.

This file should contain the user [autonomous\(\)](#) function and any functions related to it.

Any copyright is dedicated to the Public Domain. <http://creativecommons.org/publicdomain/zero/1.0/>

PROS contains FreeRTOS (<http://www.freertos.org>) whose source code may be obtained from <http://sourceforge.net/projects/freertos/files/> or on request.

5.14.2 Function Documentation

5.14.2.1 autonomous()

```
void autonomous ( )
```

Runs the user autonomous code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is lost, the autonomous task will be stopped by the kernel. Re-enabling the robot will restart the task, not re-start it from where it left off.

Code running in the autonomous task cannot access information from the VEX Joystick. However, the autonomous function can be invoked from another task if a VEX Competition Switch is not available, and it can access joystick information if called in this way.

The autonomous task may exit, unlike [operatorControl\(\)](#) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

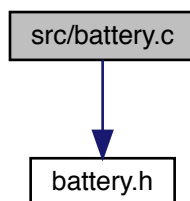
Definition at line 29 of file auto.c.

```
29         {  
30     }
```

5.15 src/battery.c File Reference

```
#include "battery.h"
```

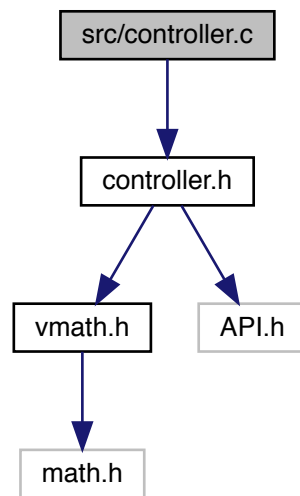
Include dependency graph for battery.c:



5.16 src/controller.c File Reference

```
#include "controller.h"
```

Include dependency graph for controller.c:



Functions

- struct `cord` `get_joystick_cord` (enum `joystick` `side`, int `controller`)

5.16.1 Function Documentation

5.16.1.1 `get_joystick_cord()`

```

struct cord get_joystick_cord (
    enum joystick side,
    int controller )

```

Definition at line 3 of file `controller.c`.

References `LEFT_JOY_X`, `LEFT_JOY_Y`, `RIGHT_JOY`, `RIGHT_JOY_X`, `RIGHT_JOY_Y`, `cord::x`, and `cord::y`.

Referenced by `update_drive_motors()`.

```

3                                     {
4   int x;
5   int y;
6   if(side == RIGHT_JOY) {
7       y = joystickGetAnalog(controller, RIGHT_JOY_X);
8       x = joystickGetAnalog(controller, RIGHT_JOY_Y);
9   } else {
10      y = joystickGetAnalog(controller, LEFT_JOY_X);
11      x = joystickGetAnalog(controller, LEFT_JOY_Y);
12  }
13  struct cord c;
14  c.x = x;
15  c.y = y;
16  return c;
17 }
```

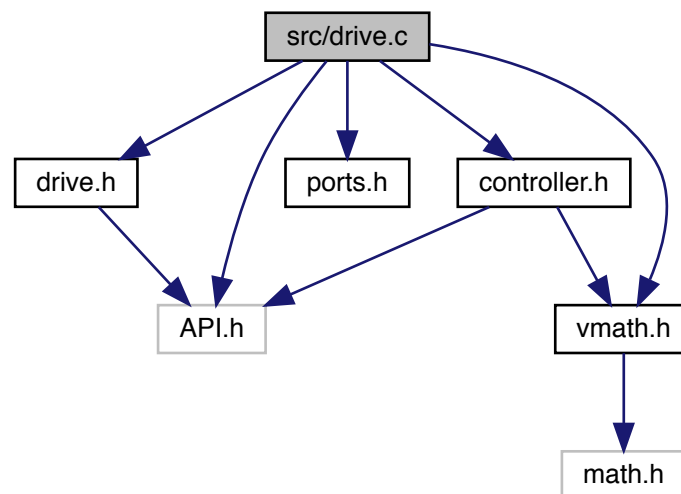
Here is the caller graph for this function:



5.17 src/drive.c File Reference

```
#include "drive.h"
#include "ports.h"
#include "vmath.h"
#include "controller.h"
#include <API.h>
```

Include dependency graph for drive.c:



Functions

- static int `deadspot` (int val)
- static int `joystick_interpolate` (int val)
- void `set_side_speed` (side_t side, int speed)
sets the speed of one side of the robot.
- void `update_drive_motors` ()
Updates the drive motors during teleop.

5.17.1 Function Documentation

5.17.1.1 deadspot()

```
static int deadspot (  
    int val ) [static]
```

Definition at line 24 of file drive.c.

References DEADSPOT.

```
24      {  
25  return abs(val) > DEADSPOT ? val : 0;  
26 }
```

5.17.1.2 joystick_interpolate()

```
static int joystick_interpolate (  
    int val ) [static]
```

Definition at line 20 of file drive.c.

```
20      {  
21  
22 }
```

5.17.1.3 set_side_speed()

```
void set_side_speed (  
    side_t side,  
    int speed )
```

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

| | |
|--------------|--|
| <i>side</i> | a side enum which indicates the size. |
| <i>speed</i> | the speed of the side. Can range from -127 - 127 negative being back and positive forwards |

Definition at line 7 of file drive.c.

References BOTH, LEFT, MOTOR_BACK_LEFT, MOTOR_BACK_RIGHT, MOTOR_FRONT_RIGHT, MOTOR_MIDDLE_RIGHT, and RIGHT.

```
7
8     if(side == RIGHT || side == BOTH) {
9         motorSet(MOTOR_BACK_RIGHT, speed);
10        motorSet(MOTOR_FRONT_RIGHT, speed);
11        motorSet(MOTOR_MIDDLE_RIGHT, speed);
12    }
13    if(side == LEFT || side == BOTH) {
14        motorSet(MOTOR_BACK_LEFT, speed);
15        motorSet(MOTOR_BACK_LEFT, speed);
16        motorSet(MOTOR_BACK_LEFT, speed);
17    }
18 }
```

5.17.1.4 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

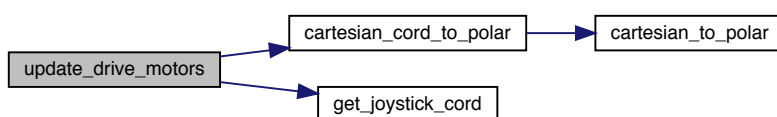
9/5/17

Definition at line 28 of file drive.c.

References cartesian_cord_to_polar(), get_joystick_cord(), MASTER, and RIGHT_JOY.

```
28
29     struct polar_cord cord = cartesian_cord_to_polar(
30         get_joystick_cord(RIGHT_JOY, MASTER));
```

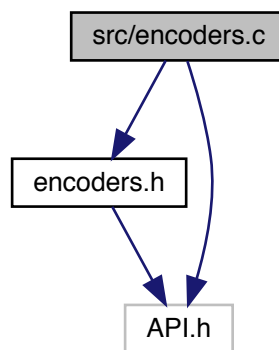
Here is the call graph for this function:



5.18 src/encoders.c File Reference

```
#include "encoders.h"  
#include <API.h>
```

Include dependency graph for encoders.c:



Functions

- int [get_encoder_ticks](#) (unsigned char address)
Gets the encoder ticks since last reset.
- int [get_encoder_velocity](#) (unsigned char address)
Gets the encoder reads.
- bool [init_encoders](#) ()
Initializes all motor encoders.

5.18.1 Function Documentation

5.18.1.1 `get_encoder_ticks()`

```
int get_encoder_ticks (  
    unsigned char address )
```

Gets the encoder ticks since last reset.

Author

Chris Jerrett

Date

9/15/2017

Definition at line 12 of file encoders.c.

```
12                                     {
13   int i = 0;
14   imeGet(address, &i);
15   return i;
16 }
```

5.18.1.2 get_encoder_velocity()

```
int get_encoder_velocity (
    unsigned char address )
```

Gets the encoder reads.

Author

Chris Jerrett

Date

9/15/2017

Definition at line 18 of file encoders.c.

```
18                                     {
19   int i = 0;
20   imeGetVelocity(address, &i);
21   return i;
22 }
```

5.18.1.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#)

Definition at line 4 of file encoders.c.

References [IME_NUMBER](#).

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

```

4      {
5  #ifdef IME_NUMBER
6  return imeInitializeAll() == IME_NUMBER;
7  #else
8  return imeInitializeAll();
9  #endif
10 }
```

Here is the caller graph for this function:

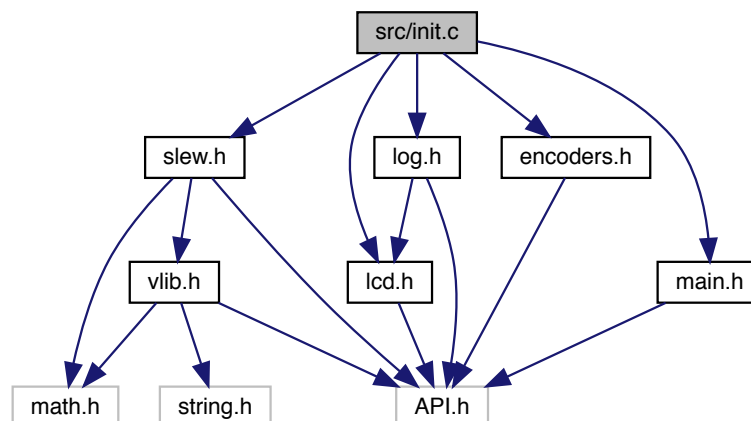


5.19 src/init.c File Reference

File for initialization code.

```

#include "main.h"
#include "slew.h"
#include "lcd.h"
#include "log.h"
#include "encoders.h"
Include dependency graph for init.c:
```



Functions

- void `initialize()`
- void `initializeIO()`

5.19.1 Detailed Description

File for initialization code.

This file should contain the user `initialize()` function and any functions related to it.

Any copyright is dedicated to the Public Domain. <http://creativecommons.org/publicdomain/zero/1.0/>

PROS contains FreeRTOS (<http://www.freertos.org>) whose source code may be obtained from <http://sourceforge.net/projects/freertos/files/> or on request.

5.19.2 Function Documentation

5.19.2.1 `initialize()`

```
void initialize ( )
```

Runs user initialization code. This function will be started in its own task with the default priority and stack size once when the robot is starting up. It is possible that the VEXnet communication link may not be fully established at this time, so reading from the VEX Joystick may fail.

This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global variables, and IMEs.

This function must exit relatively promptly, or the `operatorControl()` and `autonomous()` tasks will not start. An autonomous mode selection menu like the `pre_auton()` in other environments can be implemented in this task if desired.

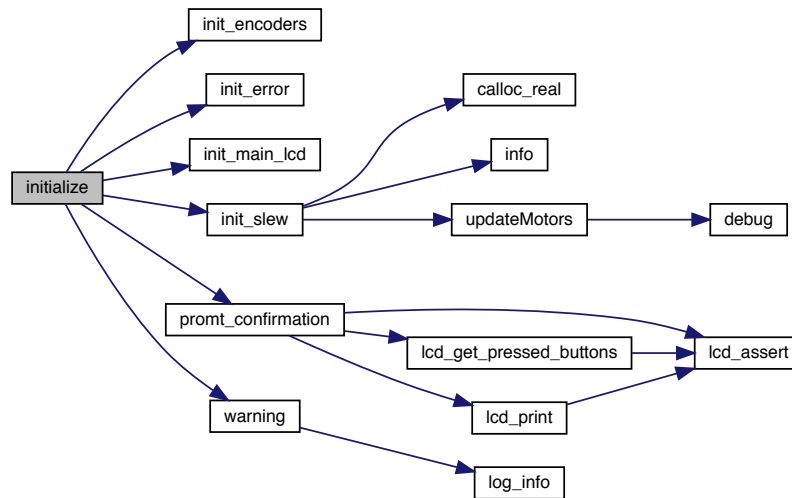
Definition at line 45 of file `init.c`.

References `init_encoders()`, `init_error()`, `init_main_lcd()`, `init_slew()`, `prompt_confirmation()`, and `warning()`.

```

45     {
46     setTeamName("9228A");
47     init_slew();
48     init_main_lcd(uart1);
49     init_error(true, uart2);
50     if(!init_encoders()) {
51         prompt_confirmation("Check IME");
52         warning("CHECK IME");
53     }
54
55     if(powerLevelBackup()/1000 == 0) {
56         prompt_confirmation("Check Backup");
57         warning("Checkbackup bat");
58     }
59 }
```

Here is the call graph for this function:



5.19.2.2 initializeIO()

```
void initializeIO ( )
```

Runs pre-initialization code. This function will be started in kernel mode one time while the VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.

The purpose of this function is solely to set the default pin modes (pinMode()) and port states (digitalWrite()) of limit switches, push buttons, and solenoids. It can also safely configure a UART port (usartOpen()) but cannot set up an LCD (lcdInit()).

Definition at line 28 of file init.c.

```

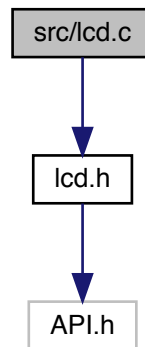
28      {
29          watchdogInit();
30      }

```

5.20 src/lcd.c File Reference

```
#include "lcd.h"
```

Include dependency graph for lcd.c:



Functions

- void [init_main_lcd](#) (FILE *lcd)
Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- static void [lcd_assert](#) ()
*Asserts the lcd is initialized Works by checking is the File *lcd_port is the default NULL value and thus not set.*
- void [lcd_clear](#) ()
Clears the lcd.
- [lcd_buttons](#) [lcd_get_pressed_buttons](#) ()
Returns the pressed buttons.
- void [lcd_print](#) (unsigned int line, const char *str)
prints a string to a line on the lcd
- void [lcd_printf](#) (unsigned int line, const char *format_str,...)
prints a formatted string to a line on the lcd. Similar to printf
- void [lcd_set_backlight](#) (bool state)
sets the backlight of the lcd
- void [prompt_confirmation](#) (const char *confirm_text)
Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Variables

- static FILE * [lcd_port](#) = NULL

5.20.1 Function Documentation

5.20.1.1 init_main_lcd()

```
void init_main_lcd (  
    FILE * lcd )
```

Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.

Parameters

| | |
|------------|---------------------------------|
| <i>lcd</i> | the uart port of the lcd screen |
|------------|---------------------------------|

See also

uart1
uart2

Author

Chris Jerrett

Date

9/9/2017

Definition at line 39 of file lcd.c.

References `lcd_port`.

Referenced by `initialize()`.

```
39                                     {  
40   lcdInit (lcd);  
41   lcd_port = lcd;  
42 }
```

Here is the caller graph for this function:

**5.20.1.2 lcd_assert()**

```
static void lcd_assert ( ) [static]
```

Asserts the lcd is initialized Works by checking is the File `*lcd_port` is the default NULL value and thus not set.

Author

Chris Jerrett

Date

9/9/2017

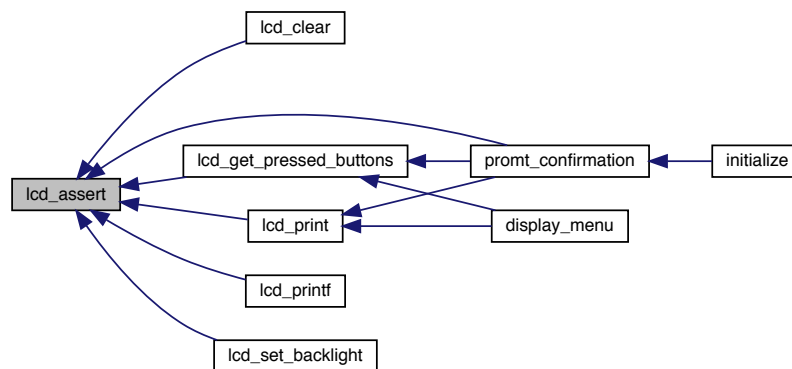
Definition at line 13 of file lcd.c.

References lcd_port.

Referenced by lcd_clear(), lcd_get_pressed_buttons(), lcd_print(), lcd_printf(), lcd_set_backlight(), and prompt_confirmation().

```
13 {  
14     if(lcd_port != NULL) {  
15         printf("LCD NULL!");  
16         exit(1);  
17     }  
18 }
```

Here is the caller graph for this function:



5.20.1.3 lcd_clear()

```
void lcd_clear ( )
```

Clears the lcd.

Author

Chris Jerrett

Date

9/9/2017

Definition at line 34 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.

```
34      {  
35  lcd_assert();  
36  lcdClear(lcd_port);  
37  }
```

Here is the call graph for this function:

**5.20.1.4 lcd_get_pressed_buttons()**

```
lcd_buttons lcd_get_pressed_buttons ( )
```

Returns the pressed buttons.

Returns

a struct containing the states of all three buttons.

Author

Chris Jerrett

Date

9/9/2017

See also

[lcd_buttons](#)

Definition at line 20 of file lcd.c.

References [lcd_assert\(\)](#), [lcd_port](#), [lcd_buttons::left](#), [lcd_buttons::middle](#), [PRESSED](#), [RELEASED](#), and [lcd_buttons::right](#).

Referenced by [display_menu\(\)](#), and [prompt_confirmation\(\)](#).

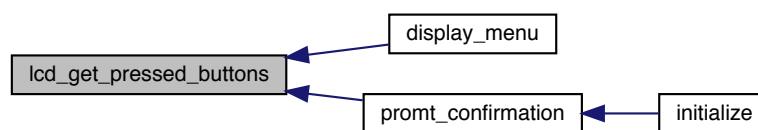
```

20                                     {
21     lcd_assert();
22     unsigned int btn_binary = lcdReadButtons(lcd_port);
23     bool left = btn_binary & 0x1;
24     bool middle = btn_binary & 0x2;
25     bool right = btn_binary & 0x4;
26     lcd_buttons btns;
27     btns.left = left ? PRESSED : RELEASED;
28     btns.middle = middle ? PRESSED : RELEASED;
29     btns.right = right ? PRESSED : RELEASED;
30
31     return btns;
32 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.20.1.5 lcd_print()

```

void lcd_print (
    unsigned int line,
    const char * str )
```

prints a string to a line on the lcd

Parameters

| | |
|-------------|----------------------|
| <i>line</i> | the line to print on |
| <i>str</i> | string to print |

Author

Chris Jerrett

Date

9/9/2017

Definition at line 44 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.Referenced by `display_menu()`, and `prompt_confirmation()`.

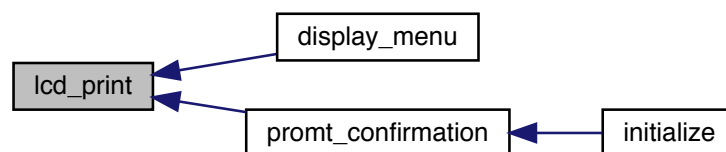
```

44                                     {
45     lcd_assert();
46     lcdSetText(lcd_port, line, str);
47 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

**5.20.1.6 lcd_printf()**

```

void lcd_printf (
    unsigned int line,
    const char * format_str,
    ... )
```

prints a formatted string to a line on the lcd. Smilar to printf

Parameters

| | |
|-------------------|-------------------------------|
| <i>line</i> | the line to print on |
| <i>format_str</i> | format string string to print |

Author

Chris Jerrett

Date

9/9/2017

Definition at line 49 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.

```
49                                     {
50     lcd_assert();
51     lcdPrint(lcd_port, line, format_str);
52 }
```

Here is the call graph for this function:



5.20.1.7 lcd_set_backlight()

```
void lcd_set_backlight (
    bool state )
```

sets the backlight of the lcd

Parameters

| | |
|--------------|--|
| <i>state</i> | a boolean representing the state of the backlight. true = on, false = off. |
|--------------|--|

Author

Chris Jerrett

Date

9/9/2017

Definition at line 54 of file lcd.c.

References `lcd_assert()`, and `lcd_port`.

```
54                                     {  
55   lcd_assert();  
56   lcdSetBacklight(lcd_port, state);  
57 }
```

Here is the call graph for this function:

**5.20.1.8 prompt_confirmation()**

```
void prompt_confirmation (  
    const char * confirm_text )
```

Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Parameters

| | |
|---------------------|-----------------------------------|
| <i>confirm_text</i> | the text for the user to confirm. |
|---------------------|-----------------------------------|

Author

Chris Jerrett

Date

9/9/2017

Definition at line 59 of file lcd.c.

References `lcd_assert()`, `lcd_get_pressed_buttons()`, `lcd_print()`, and `PRESSED`.

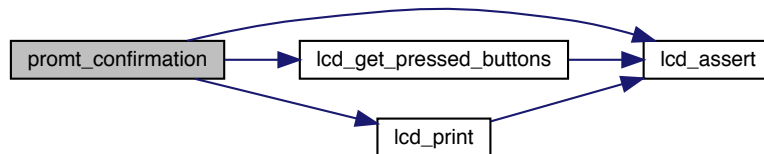
Referenced by `initialize()`.

```

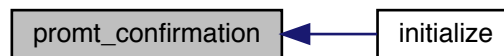
59                                     {
60     lcd_assert();
61     lcd_print(1, confirm_text);
62     while(lcd_get_pressed_buttons().middle != PRESSED){
63         delay(200);
64     }
65 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.20.2 Variable Documentation

5.20.2.1 lcd_port

```
FILE* lcd_port = NULL [static]
```

The port of the initialized lcd

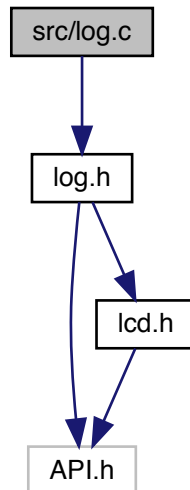
Definition at line 4 of file lcd.c.

Referenced by `init_main_lcd()`, `lcd_assert()`, `lcd_clear()`, `lcd_get_pressed_buttons()`, `lcd_print()`, `lcd_printf()`, and `lcd_set_backlight()`.

5.21 src/log.c File Reference

```
#include "log.h"
```

Include dependency graph for log.c:



Functions

- void `debug` (const char *debug_message)
prints a info message
- void `error` (const char *error_message)
prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE
- void `info` (const char *info_message)
prints a info message
- void `init_error` (bool use_lcd, FILE *lcd)
Initializes the error lcd system Only required if using lcd.
- static void `log_info` (const char *s, const char *mess)
- void `warning` (const char *warning_message)
prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

Variables

- static FILE * `log_lcd` = NULL
- unsigned int `log_level` = `DEBUG`

5.21.1 Function Documentation

5.21.1.1 debug()

```
void debug (
    const char * debug_message )
```

prints a info message

Only will print and display if log_level is greater than info

See also

[log_level](#)

Parameters

| | |
|----------------------|-------------|
| <i>debug_message</i> | the message |
|----------------------|-------------|

Definition at line 37 of file log.c.

References `ERROR`, and `log_level`.

Referenced by `updateMotors()`.

```
37                                     {
38     if(log_level>ERROR) {
39         printf("[INFO]: %s\n", debug_message);
40     }
41 }
```

Here is the caller graph for this function:



5.21.1.2 error()

```
void error (
    const char * error_message )
```

prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

[log_level](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

| | |
|----------------------|-------------|
| <i>error_message</i> | the message |
|----------------------|-------------|

Definition at line 21 of file log.c.

References `log_info()`, `log_level`, and `NONE`.

```
21                                     {  
22     if(log_level>NONE)  
23         log_info("ERROR", error_message);  
24 }
```

Here is the call graph for this function:

**5.21.1.3 info()**

```
void info (  
    const char * info_message )
```

prints a info message

Only will print and display if `log_level` is greater than `ERROR`

See also

[log_level](#)

Parameters

| | |
|---------------------|-------------|
| <i>info_message</i> | the message |
|---------------------|-------------|

Definition at line 31 of file log.c.

References `ERROR`, and `log_level`.

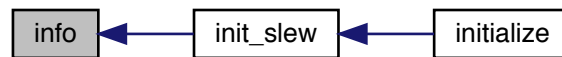
Referenced by `init_slew()`.


```

31                                     {
32     if(log_level>ERROR) {
33         printf("[INFO]: %s\n", info_message);
34     }
35 }

```

Here is the caller graph for this function:



5.21.1.4 init_error()

```

void init_error (
    bool use_lcd,
    FILE * lcd )

```

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

| | |
|----------------|------------------------|
| <i>use_lcd</i> | whether to use the lcd |
| <i>lcd</i> | the lcd |

Definition at line 6 of file log.c.

References `log_lcd`.

Referenced by `initialize()`.

```

6                                     {
7     if(use_lcd) {
8         lcdInit(lcd);
9         log_lcd = lcd;
10    }
11 }

```

Here is the caller graph for this function:



5.21.1.5 log_info()

```
static void log_info (  
    const char * s,  
    const char * mess ) [static]
```

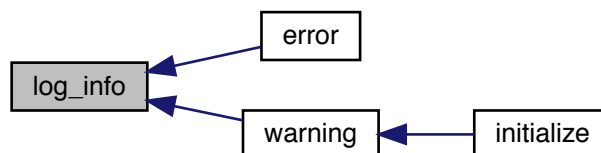
Definition at line 13 of file log.c.

References `BOTTOM_ROW`, `log_lcd`, and `TOP_ROW`.

Referenced by `error()`, and `warning()`.

```
13     {  
14     printf("[%s]: %s\n", s, mess);  
15     lcdSetBacklight(log_lcd, true);  
16     lcdClear(log_lcd);  
17     lcdPrint(log_lcd, TOP_ROW, s);  
18     lcdPrint(log_lcd, BOTTOM_ROW, mess);  
19 }
```

Here is the caller graph for this function:



5.21.1.6 warning()

```
void warning (
    const char * warning_message )
```

prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

[log_level](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

| | |
|------------------------|-------------|
| <i>warning_message</i> | the message |
|------------------------|-------------|

Definition at line 26 of file log.c.

References [log_info\(\)](#), [log_level](#), and [WARNING](#).

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

```
26                                     {
27     if(log_level>WARNING)
28         log_info("WARNING", warning_message);
29 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.21.2 Variable Documentation

5.21.2.1 log_lcd

```
FILE* log_lcd = NULL [static]
```

Definition at line 4 of file log.c.

Referenced by `init_error()`, and `log_info()`.

5.21.2.2 log_level

```
unsigned int log_level = DEBUG
```

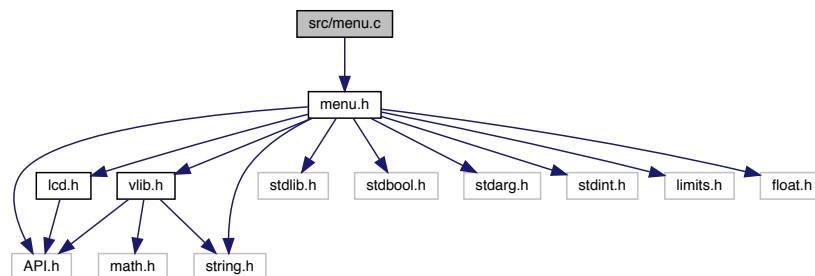
Definition at line 3 of file log.c.

Referenced by `debug()`, `error()`, `info()`, and `warning()`.

5.22 src/menu.c File Reference

```
#include "menu.h"
```

Include dependency graph for menu.c:



Functions

- static void `calculate_current_display` (char *rtn, menu_t *menu)
- static menu_t * `create_menu` (enum menu_type type, const char *prompt)
- void `denint_menu` (menu_t *menu)
Destroys a menu Menu must be freed or will cause memory leak
- int `display_menu` (menu_t *menu)
Displays a menu context, but does not display. Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.
- menu_t * `init_menu_float` (enum menu_type type, float min, float max, float step, char *prompt)
Creates a menu context, but does not display. Menu must be freed or will cause memory leak!
- menu_t * `init_menu_int` (enum menu_type type, int min, int max, int step, char *prompt)
Creates a menu context, but does not display. Menu must be freed or will cause memory leak
- menu_t * `init_menu_var` (enum menu_type type, unsigned int nums, char *prompt, char *options,...)
Creates a menu context, but does not display. Menu must be freed or will cause memory leak

5.22.1 Function Documentation

5.22.1.1 calculate_current_display()

```
static void calculate_current_display (
    char * rtn,
    menu_t * menu ) [static]
```

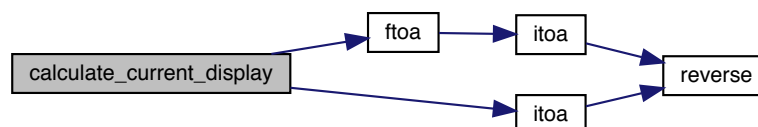
Definition at line 47 of file menu.c.

References menu_t::current, FLOAT_TYPE, ftoa(), INT_TYPE, itoa(), menu_t::length, menu_t::max, menu_t::max_f, menu_t::min, menu_t::min_f, menu_t::options, menu_t::step, menu_t::step_f, STRING_TYPE, and menu_t::type.

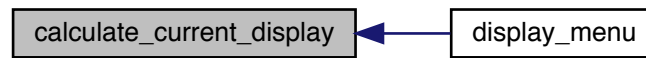
Referenced by display_menu().

```
47                                     {
48  if(menu->type == STRING_TYPE){
49      //Ignore warning
50      rtn = (menu->options[menu->current % (menu->length)]);
51  }
52  if(menu->type == INT_TYPE) {
53      int step = (menu->step);
54      int min = (menu->min);
55      int max = (menu->max);
56      int value = menu->current * step;
57      value = value < min ? min : value;
58      value = value > max ? max : value;
59      itoa(value, rtn, 4);
60  }
61  if(menu->type == FLOAT_TYPE) {
62      float step = (menu->step_f);
63      float min = (menu->min_f);
64      float max = (menu->max_f);
65      float value = menu->current * step;
66      value = value < min ? min : value;
67      value = value > max ? max : value;
68
69      ftoa(value, rtn, 5);
70  }
71 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.22.1.2 create_menu()

```
static menu_t* create_menu (
    enum menu_type type,
    const char * prompt ) [static]
```

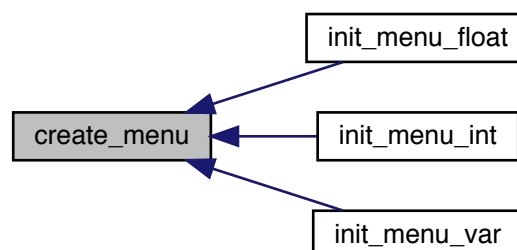
Definition at line 3 of file menu.c.

References `menu_t::max`, `menu_t::max_f`, `menu_t::min`, `menu_t::min_f`, `menu_t::prompt`, `menu_t::step`, `menu_t::step_f`, and `menu_t::type`.

Referenced by `init_menu_float()`, `init_menu_int()`, and `init_menu_var()`.

```
3
4 menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
5 menu->type = type;
6 strcpy(menu->prompt, prompt);
7 menu->max = INT_MAX;
8 menu->min = INT_MIN;
9 menu->step = 1;
10 menu->min_f = FLT_MIN;
11 menu->max_f = FLT_MAX;
12 menu->step_f = 1;
13
14 return menu;
15 }
```

Here is the caller graph for this function:



5.22.1.3 denint_menu()

```
void denint_menu (
    menu_t * menu )
```

Destroys a menu *Menu must be freed or will cause memory leak*

Parameters

| | |
|-------------|------------------|
| <i>menu</i> | the menu to free |
|-------------|------------------|

See also

[menu](#)

Author

Chris Jerrett

Date

9/8/17

Definition at line 92 of file menu.c.

References [menu_t::options](#), and [menu_t::prompt](#).

```
92     {
93     free(menu->prompt);
94     if(menu->options != NULL) free(menu->options);
95     free(menu);
96 }
```

5.22.1.4 display_menu()

```
int display_menu (
    menu_t * menu )
```

Displays a menu context, but does not display. *Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.*

Parameters

| | |
|-------------|---------------------|
| <i>menu</i> | the menu to display |
|-------------|---------------------|

See also

[menu_type](#)

Author

Chris Jerrett

Date

9/8/17

Definition at line 74 of file menu.c.

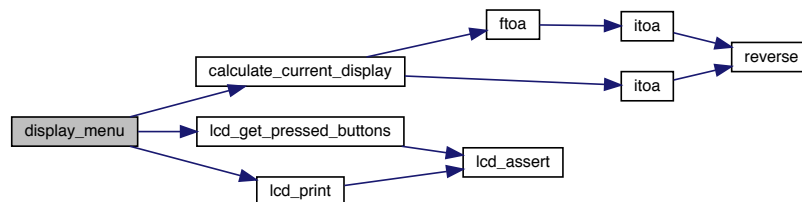
References `calculate_current_display()`, `menu_t::current`, `lcd_get_pressed_buttons()`, `lcd_print()`, `PRESSED`, `menu_t::prompt`, `RELEASED`, and `TOP_ROW`.

```

74         {
75     lcd_print(TOP_ROW, menu->prompt);
76     //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
77     while(lcd_get_pressed_buttons().middle == RELEASED && !isEnabled()) {
78         char val[16];
79         calculate_current_display(val, menu);
80
81         if(lcd_get_pressed_buttons().right == PRESSED) {
82             menu->current += 1;
83         }
84         if(lcd_get_pressed_buttons().left == PRESSED) {
85             menu->current -= 1;
86         }
87         delay(500);
88     }
89     return menu->current;
90 }

```

Here is the call graph for this function:

**5.22.1.5 init_menu_float()**

```

menu_t* init_menu_float (
    enum menu_type type,
    float min,
    float max,
    float step,
    char * prompt )

```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak!*

Parameters

| | |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

[menu_type](#)

Parameters

| | |
|---------------|-------------------------------|
| <i>min</i> | the minimum value |
| <i>max</i> | the maximum value |
| <i>step</i> | the step value |
| <i>prompt</i> | the prompt to display to user |

Author

Chris Jerrett

Date

9/8/17

Definition at line 39 of file menu.c.

References `create_menu()`, `menu_t::max_f`, `menu_t::min_f`, and `menu_t::step_f`.

```

39
40  menu_t* menu = create_menu(type, prompt);
41  menu->min_f = min;
42  menu->max_f = max;
43  menu->step_f = step;
44  return menu;
45 }
```

Here is the call graph for this function:



5.22.1.6 init_menu_int()

```

menu_t* init_menu_int (
    enum menu_type type,
    int min,
    int max,
    int step,
    char * prompt )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

| | |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

[menu_type](#)

Parameters

| | |
|---------------|-------------------------------|
| <i>min</i> | the minimum value |
| <i>max</i> | the maximum value |
| <i>step</i> | the step value |
| <i>prompt</i> | the prompt to display to user |

Author

Chris Jerrett

Date

9/8/17

Definition at line 31 of file menu.c.

References `create_menu()`, `menu_t::max`, `menu_t::min`, and `menu_t::step`.

```

31
32  menu_t* menu = create_menu(type, prompt);
33  menu->min = min;
34  menu->max = max;
35  menu->step = step;
36  return menu;
37 }
```

Here is the call graph for this function:



5.22.1.7 init_menu_var()

```

menu_t* init_menu_var (
    enum menu_type type,
    unsigned int nums,
    char * prompt,
    char * options,
    ... )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

| | |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

[menu_type](#)

Parameters

| | |
|----------------|---|
| <i>nums</i> | the number of elements passed to function |
| <i>prompt</i> | the prompt to display to user |
| <i>options</i> | the options to display for user |

Author

Chris Jerrett

Date

9/8/17

Definition at line 17 of file menu.c.

References [create_menu\(\)](#), [menu_t::length](#), and [menu_t::options](#).

```
17                                     {
18  menu_t* menu = create_menu(type, prompt);
19  va_list values;
20  char **options_array = (char**)malloc(sizeof(char*) * nums);
21  va_start(values, options);
22  for(unsigned int i = 0; i < nums; i++){
23      options_array[i] = va_arg(values, char*);
24  }
25  va_end(values);
26  menu->options = options_array;
27  menu->length = nums;
28  return menu;
29 }
```

Here is the call graph for this function:

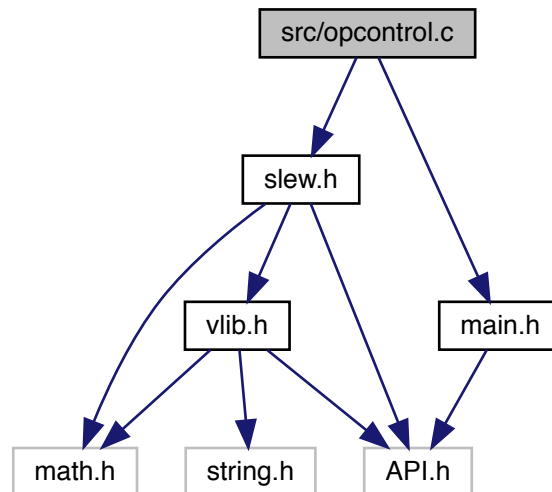


5.23 src/opcontrol.c File Reference

File for operator control code.

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

Include dependency graph for opcontrol.c:



Functions

- void `operatorControl()`

5.23.1 Detailed Description

File for operator control code.

This file should contain the user `operatorControl()` function and any functions related to it.

Any copyright is dedicated to the Public Domain. <http://creativecommons.org/publicdomain/zero/1.0/>

PROS contains FreeRTOS (<http://www.freertos.org>) whose source code may be obtained from <http://sourceforge.net/projects/freertos/files/> or on request.

5.23.2 Function Documentation

5.23.2.1 operatorControl()

```
void operatorControl ( )
```

Runs the user operator control code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the operator control mode. If the robot is disabled or communications is lost, the operator control task will be stopped by the kernel. Re-enabling the robot will restart the task, not resume it from where it left off.

If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will run the operator control task. Be warned that this will also occur if the VEX Cortex is tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.

Code running in this task can take almost any action, as the VEX Joystick is available and the scheduler is operational. However, proper use of `delay()` or `taskDelayUntil()` is highly recommended to give other tasks (including system tasks such as updating LCDs) time to run.

This task should never exit; it should end with some kind of infinite loop, even if empty.

Definition at line 33 of file `opcontrol.c`.

References `set_motor_slew()`.

```
33         {
34     while (1) {
35         set_motor_slew(2, 100);
36         delay(20);
37     }
38 }
```

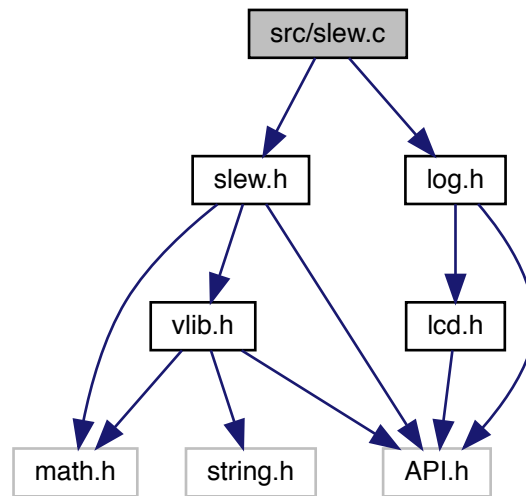
Here is the call graph for this function:



5.24 src/slew.c File Reference

```
#include "slew.h"
#include "log.h"
```

Include dependency graph for `slew.c`:



Functions

- void `deinitslew` ()
Deinitializes the slew rate controller and frees memory.
- void `init_slew` ()
Initializes the slew rate controller.
- void `set_motor_slew` (int motor, int speed)
Sets motor speed wrapped inside the slew rate controller.
- void `updateMotors` ()
Closes the distance between the desired motor value and the current motor value by half for each motor.

Variables

- static bool `initialized` = false
Boolean indicating whether or not the slew rate controller has been initialized.
- static signed char * `motors_set_speeds` = NULL
Array of motor speed values to set the motors to.
- static Mutex `mutex`
mutex to protect the data in the array of speeds from being read or written to simultaneously.
- static TaskHandle `slew` = NULL
Task that will handle updating the motors on a routine period.

5.24.1 Function Documentation

5.24.1.1 deinit_slew()

```
void deinit_slew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line 62 of file slew.c.

References `motors_set_speeds`, and `slew`.

```
62     {
63     free(motors_set_speeds);
64     taskDelete(slew);
65 }
```

5.24.1.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

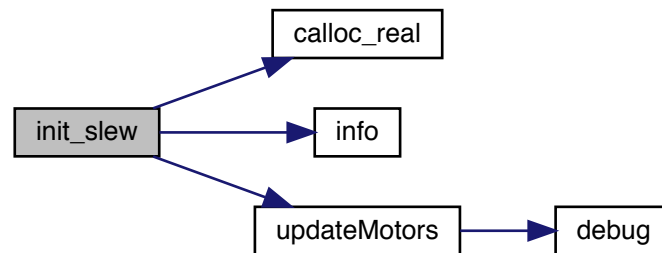
Definition at line 49 of file slew.c.

References `calloc_real()`, `info()`, `initialized`, `MOTOR_PORTS`, `mutex`, `slew`, `UPDATE_PERIOD_MS`, and `updateMotors()`.

Referenced by `initialize()`.

```
49     {
50     info("Init Slew");
51     calloc_real(MOTOR_PORTS, sizeof(char));
52     mutex = mutexCreate();
53     slew = taskRunLoop(updateMotors, UPDATE_PERIOD_MS);
54     initialized = true;
55 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.24.1.3 set_motor_slew()

```
void set_motor_slew (
    int motor,
    int speed )
```

Sets motor speed wrapped inside the slew rate controller.

Parameters

| | |
|--------------|--|
| <i>motor</i> | the motor port to use |
| <i>speed</i> | the speed to use, between -127 and 127 |

Author

Chris Jerrett

Date

9/14/17

Definition at line 67 of file slew.c.

References `motors_set_speeds`, and `mutex`.

Referenced by `operatorControl()`.

```
67                                     {
68     if(mutexTake(mutex, 100)) {
69         motors_set_speeds[motor] = speed;
70         mutexGive(mutex);
71     }
72 }
```

Here is the caller graph for this function:

**5.24.1.4 updateMotors()**

```
void updateMotors ( )
```

Closes the distance between the desired motor value and the current motor value by half for each motor.

Author

Chris Jerrett

Date

9/14/17

Definition at line 31 of file slew.c.

References `debug()`, `MOTOR_PORTS`, `motors_set_speeds`, `mutex`, and `RAMP_PROPORTION`.

Referenced by `init_slew()`.

```

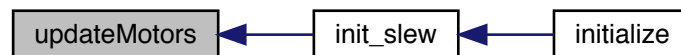
31         {
32     //Take back half approach
33     //Not linear but equal to setSpeed(1-(1/2)^x)
34     if(mutexTake(mutex, 10)) {
35         for(int i = 0; i < MOTOR_PORTS; i++) {
36             char set_speed = motors_set_speeds[i];
37             char curr_speed = motorGet(i);
38             char diff = set_speed - curr_speed;
39             int n = (int) curr_speed + ceil(diff/(float)RAMP_PROPORTION);
40             char c[16];
41             sprintf(c, "Set Motor %d: %d", i, n);
42             debug(c);
43             motorSet(i, n);
44         }
45         mutexGive(mutex);
46     }
47 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.24.2 Variable Documentation

5.24.2.1 initialized

```
bool initialized = false [static]
```

Boolean indicating whether or not the slew rate controller has been initialized.

Author

Chris Jerrett

Date

9/14/17

Definition at line 29 of file slew.c.

Referenced by init_slew().

5.24.2.2 motors_set_speeds

```
signed char* motors_set_speeds = NULL [static]
```

Array of motor speed values to set the motors to.

Author

Chris Jerrett

Date

9/14/17

Definition at line 15 of file slew.c.

Referenced by deinitslew(), set_motor_slew(), and updateMotors().

5.24.2.3 mutex

```
Mutex mutex [static]
```

mutex to protect the data in the array of speeds from being read or written to simultaneously.

Author

Chris Jerrett

Date

9/14/17

Definition at line 8 of file slew.c.

Referenced by init_slew(), set_motor_slew(), and updateMotors().

5.24.2.4 slew

```
TaskHandle slew = NULL [static]
```

Task that will handle updating the motors on a routine period.

Author

Chris Jerrett

Date

9/14/17

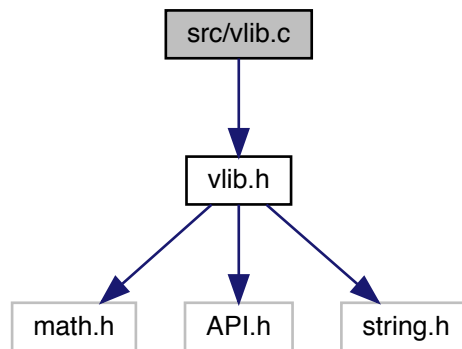
Definition at line 22 of file slew.c.

Referenced by deinitslew(), and init_slew().

5.25 src/vlib.c File Reference

```
#include "vlib.h"
```

Include dependency graph for vlib.c:



Functions

- void * [calloc_real](#) (size_t elements, size_t size)
- void [ftoa](#) (float a, char *buffer, int precision)
converts a float to string.
- int [itoa](#) (int a, char *buffer, int digits)
converts a int to string.
- void [reverse](#) (char *str, int len)
reverses a string 'str' of length 'len'

5.25.1 Function Documentation

5.25.1.1 calloc_real()

```
void* calloc_real (
    size_t elements,
    size_t size )
```

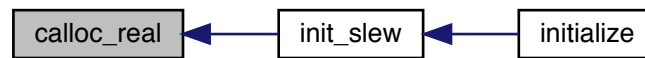
Definition at line 53 of file vlib.c.

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

```

53                                     {
54     void *mem = malloc(elements * size);
55     //This is not a error. Bad ATOM!
56     memset(mem, 0, elements * size);
57     return mem;
58 }
```

Here is the caller graph for this function:



5.25.1.2 ftoa()

```
void ftoa (
    float a,
    char * buffer,
    int precision )
```

converts a float to string.

Parameters

| | |
|------------------|--|
| <i>a</i> | the float |
| <i>buffer</i> | the string the float will be written to. |
| <i>precision</i> | digits after the decimal to write |

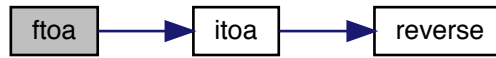
Definition at line 30 of file vlib.c.

References `itoa()`.

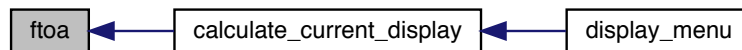
Referenced by `calculate_current_display()`.

```
30                                     {
31     // Extract integer part
32     int ipart = (int)a;
33
34     // Extract floating part
35     float fpart = a - (float)ipart;
36
37     // convert integer part to string
38     int i = itoa(ipart, buffer, 0);
39
40     // check for display option after point
41     if(precision != 0) {
42         buffer[i] = '.'; // add dot
43
44         // Get the value of fraction part up to given num.
45         // of points after dot. The third parameter is needed
46         // to handle cases like 233.007
47         fpart = fpart * pow(10, precision);
48
49         itoa((int)fpart, buffer + i + 1, precision);
50     }
51 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.1.3 itoa()

```
int itoa (
    int a,
    char * buffer,
    int digits )
```

converts a int to string.

Parameters

| | |
|---------------|--|
| <i>a</i> | the integer |
| <i>buffer</i> | the string the int will be written to. |
| <i>digits</i> | the number of digits to be written |

Returns

the digits

Author

Chris Jerrett

Date

9/9/2017

Definition at line 13 of file vlib.c.

References `reverse()`.

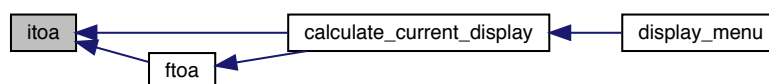
Referenced by `calculate_current_display()`, and `ftoa()`.

```
13                                     {
14     int i = 0;
15     while (a) {
16         buffer[i++] = (a%10) + '0';
17         a = a/10;
18     }
19
20     // If number of digits required is more, then
21     // add 0s at the beginning
22     while (i < digits)
23         buffer[i++] = '0';
24
25     reverse(buffer, i);
26     buffer[i] = '\0';
27     return i;
28 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.1.4 reverse()

```
void reverse (
    char * str,
    int len )
```

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

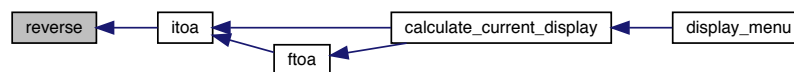
| | |
|------------|-----------------------|
| <i>str</i> | the string to reverse |
| <i>len</i> | the length |

Definition at line 3 of file vlib.c.

Referenced by itoa().

```
3
4     int i=0, j=len-1, temp;      {
5     while (i<j) {
6         temp = str[i];
7         str[i] = str[j];
8         str[j] = temp;
9         i++; j--;
10    }
11 }
```

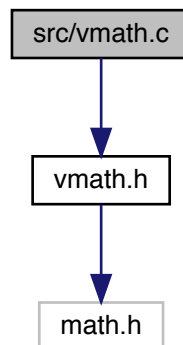
Here is the caller graph for this function:



5.26 src/vmath.c File Reference

```
#include "vmath.h"
```


Include dependency graph for vmath.c:



Functions

- struct [polar_cord](#) [cartesian_cord_to_polar](#) (struct [cord](#) cords)
Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- struct [polar_cord](#) [cartesian_to_polar](#) (float x, float y)
Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

5.26.1 Function Documentation

5.26.1.1 cartesian_cord_to_polar()

```
struct polar\_cord cartesian_cord_to_polar (  
    struct cord cords )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

| | |
|--------------|---------------------|
| <i>cords</i> | the cartesian cords |
|--------------|---------------------|

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)
[cord](#)

Definition at line 33 of file vmath.c.

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

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

```
33                                     {  
34   return cartesian_to_polar(cords.x, cords.y);  
35 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

**5.26.1.2 cartesian_to_polar()**

```
struct polar_cord cartesian_to_polar (  
    float x,  
    float y )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

| | |
|----------|--|
| <i>x</i> | float value of the x cartesian coordinate. |
| <i>y</i> | float value of the y cartesian coordinate. |

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)

Definition at line 3 of file vmath.c.

References `polar_cord::angle`, and `polar_cord::magnitue`.

Referenced by `cartesian_cord_to_polar()`.

```

3                                     {
4     float degree = 0;
5     double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
6
7     if(x < 0){
8         degree += 180.0;
9     }
10    else if(x > 0 && y < 0){
11        degree += 360.0;
12    }
13
14    if(x != 0 && y != 0){
15        degree += atan((float)y / (float)x);
16    }
17    else if(x == 0 && y > 0){
18        degree = 90.0;
19    }
20    else if(y == 0 && x < 0){
21        degree = 180.0;
22    }
23    else if(x == 0 && y < 0){
24        degree = 270.0;
25    }
26
27    struct polar_cord p;
28    p.angle = degree;
29    p.magnitue = magnitude;
30    return p;
31 }
```

Here is the caller graph for this function:

