

Vex Team 9228A

1.1.4

Generated by Doxygen 1.8.13

Tue Nov 28 2017 23:39:31

Contents

1	InTheZoneA	1
2	Data Structure Index	1
2.1	Data Structures	1
3	File Index	2
3.1	File List	2
4	Data Structure Documentation	4
4.1	_matrix Struct Reference	4
4.1.1	Detailed Description	4
4.1.2	Field Documentation	4
4.2	accelerometer_odometry Struct Reference	5
4.2.1	Detailed Description	5
4.2.2	Field Documentation	5
4.3	cord Struct Reference	6
4.3.1	Detailed Description	6
4.3.2	Field Documentation	6
4.4	encoder_odemtry Struct Reference	7
4.4.1	Detailed Description	7
4.4.2	Field Documentation	7
4.5	lcd_buttons Struct Reference	8
4.5.1	Detailed Description	8
4.5.2	Field Documentation	9
4.6	location Struct Reference	9
4.6.1	Detailed Description	10
4.6.2	Field Documentation	10
4.7	menu_t Struct Reference	10
4.7.1	Detailed Description	11
4.7.2	Field Documentation	12
4.8	polar_cord Struct Reference	16
4.8.1	Detailed Description	16
4.8.2	Field Documentation	17

5 File Documentation	17
5.1 include/API.h File Reference	17
5.1.1 Detailed Description	22
5.1.2 Macro Definition Documentation	22
5.1.3 Typedef Documentation	34
5.1.4 Function Documentation	36
5.1.5 Variable Documentation	90
5.2 API.h	90
5.3 include/auto.h File Reference	94
5.3.1 Detailed Description	95
5.3.2 Macro Definition Documentation	95
5.4 auto.h	97
5.5 include/battery.h File Reference	97
5.5.1 Detailed Description	98
5.5.2 Macro Definition Documentation	99
5.5.3 Function Documentation	99
5.6 battery.h	102
5.7 include/claw.h File Reference	102
5.7.1 Detailed Description	104
5.7.2 Macro Definition Documentation	104
5.7.3 Enumeration Type Documentation	107
5.7.4 Function Documentation	108
5.8 claw.h	113
5.9 include/controller.h File Reference	113
5.9.1 Detailed Description	115
5.9.2 Macro Definition Documentation	115
5.9.3 Enumeration Type Documentation	118
5.9.4 Function Documentation	119

5.10 controller.h	120
5.11 include/drive.h File Reference	120
5.11.1 Detailed Description	122
5.11.2 Macro Definition Documentation	122
5.11.3 Typedef Documentation	123
5.11.4 Enumeration Type Documentation	123
5.11.5 Function Documentation	124
5.12 drive.h	127
5.13 include/encoders.h File Reference	127
5.13.1 Detailed Description	129
5.13.2 Macro Definition Documentation	129
5.13.3 Function Documentation	130
5.14 encoders.h	132
5.15 include/lcd.h File Reference	133
5.15.1 Detailed Description	134
5.15.2 Macro Definition Documentation	135
5.15.3 Enumeration Type Documentation	135
5.15.4 Function Documentation	136
5.16 lcd.h	143
5.17 include/lifter.h File Reference	143
5.17.1 Detailed Description	145
5.17.2 Macro Definition Documentation	146
5.17.3 Function Documentation	148
5.18 lifter.h	155
5.19 include/localization.h File Reference	156
5.19.1 Detailed Description	157
5.19.2 Macro Definition Documentation	158
5.19.3 Function Documentation	158

5.20 localization.h	159
5.21 include/log.h File Reference	160
5.21.1 Detailed Description	162
5.21.2 Macro Definition Documentation	162
5.21.3 Function Documentation	164
5.22 log.h	170
5.23 include/main.h File Reference	170
5.23.1 Detailed Description	171
5.23.2 Function Documentation	172
5.24 main.h	177
5.25 include/matrix.h File Reference	177
5.25.1 Detailed Description	179
5.25.2 Typedef Documentation	179
5.25.3 Function Documentation	179
5.26 matrix.h	195
5.27 include/menu.h File Reference	196
5.27.1 Detailed Description	198
5.27.2 Typedef Documentation	198
5.27.3 Enumeration Type Documentation	198
5.27.4 Function Documentation	199
5.28 menu.h	207
5.29 include/mobile_goal_intake.h File Reference	208
5.29.1 Function Documentation	209
5.30 mobile_goal_intake.h	210
5.31 include/motor_ports.h File Reference	210
5.31.1 Detailed Description	211
5.31.2 Macro Definition Documentation	212
5.32 motor_ports.h	216

5.33 include/partner.h File Reference	216
5.33.1 Enumeration Type Documentation	217
5.33.2 Function Documentation	218
5.34 partner.h	219
5.35 include/potentiometer.h File Reference	220
5.35.1 Macro Definition Documentation	220
5.36 potentiometer.h	221
5.37 include/sensor_ports.h File Reference	221
5.37.1 Macro Definition Documentation	222
5.38 sensor_ports.h	223
5.39 include/slew.h File Reference	223
5.39.1 Detailed Description	224
5.39.2 Macro Definition Documentation	225
5.39.3 Function Documentation	226
5.40 slew.h	232
5.41 include/toggle.h File Reference	233
5.41.1 Enumeration Type Documentation	234
5.41.2 Function Documentation	235
5.42 toggle.h	239
5.43 include/vlib.h File Reference	240
5.43.1 Detailed Description	241
5.43.2 Function Documentation	241
5.44 vlib.h	245
5.45 include/vmath.h File Reference	245
5.45.1 Detailed Description	247
5.45.2 Macro Definition Documentation	247
5.45.3 Function Documentation	247
5.46 vmath.h	252

5.47 README.md File Reference	252
5.48 README.md	252
5.49 src/auto.c File Reference	252
5.49.1 Detailed Description	253
5.49.2 Function Documentation	253
5.50 auto.c	255
5.51 src/battery.c File Reference	256
5.51.1 Function Documentation	257
5.52 battery.c	259
5.53 src/claw.c File Reference	260
5.53.1 Function Documentation	260
5.54 claw.c	266
5.55 src/controller.c File Reference	266
5.55.1 Function Documentation	267
5.56 controller.c	268
5.57 src/drive.c File Reference	268
5.57.1 Function Documentation	269
5.57.2 Variable Documentation	274
5.58 drive.c	274
5.59 src/encoders.c File Reference	275
5.59.1 Function Documentation	276
5.60 encoders.c	279
5.61 src/init.c File Reference	279
5.61.1 Detailed Description	280
5.61.2 Function Documentation	280
5.62 init.c	281
5.63 src/lcd.c File Reference	282
5.63.1 Function Documentation	283

5.63.2 Variable Documentation	292
5.64 lcd.c	292
5.65 src/lifter.c File Reference	293
5.65.1 Function Documentation	294
5.66 lifter.c	303
5.67 src/localization.c File Reference	304
5.67.1 Function Documentation	306
5.67.2 Variable Documentation	310
5.68 localization.c	311
5.69 src/log.c File Reference	312
5.69.1 Function Documentation	314
5.69.2 Variable Documentation	321
5.70 log.c	322
5.71 src/matrix.c File Reference	322
5.71.1 Function Documentation	323
5.72 matrix.c	340
5.73 src/menu.c File Reference	344
5.73.1 Function Documentation	345
5.74 menu.c	354
5.75 src/mobile_goal_intake.c File Reference	355
5.75.1 Function Documentation	356
5.76 mobile_goal_intake.c	360
5.77 src/opcontrol.c File Reference	360
5.77.1 Detailed Description	361
5.77.2 Function Documentation	361
5.78 opcontrol.c	362
5.79 src/partner.c File Reference	363
5.79.1 Function Documentation	363

5.79.2 Variable Documentation	365
5.80 partner.c	365
5.81 src/slew.c File Reference	366
5.81.1 Function Documentation	367
5.81.2 Variable Documentation	373
5.82 slew.c	375
5.83 src/toggle.c File Reference	375
5.83.1 Function Documentation	376
5.83.2 Variable Documentation	380
5.84 toggle.c	380
5.85 src/vlib.c File Reference	382
5.85.1 Function Documentation	383
5.86 vlib.c	386
5.87 src/vmath.c File Reference	387
5.87.1 Function Documentation	388
5.88 vmath.c	393

1 InTheZoneA

Team A code for In The Zone

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_matrix	4
accelerometer_odometry	5
cord A struct that contains cartesian coordinates	6

encoder_odemtry	7
lcd_buttons	
State of the lcd buttons	8
location	9
menu_t	
Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via deinit_menu	10
polar_cord	
A struct that contains polar coordinates	16

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/ API.h	
Provides the high-level user functionality intended for use by typical VEX Cortex programmers	17
include/ auto.h	
Autonomous declarations and macros	94
include/ battery.h	
Battery management related functions	97
include/ claw.h	
Code for controlling the claw that grabs the cones	102
include/ controller.h	
Controller definitions, macros and functions to assist with usig the vex controllers	113
include/ drive.h	
Drive base definitions and enumerations	120
include/ encoders.h	
Wrapper around encoder functions	127
include/ lcd.h	
LCD wrapper functions and macros	133
include/ lifter.h	
Declarations and macros for controlling and manipulating the lifter	143
include/ localization.h	
Declarations and macros for determining the location of the robot. [WIP]	156
include/ log.h	
Contains logging functions	160

include/ main.h	
Header file for global functions	170
None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevalent, be sure to use the function to reclaim some of that memory 177	
include/ menu.h	
Contains menu functionality and abstraction	196
include/ mobile_goal_intake.h	208
Macros for the different motors ports 210	
include/ partner.h	216
include/ potentiometer.h	220
include/ sensor_ports.h	221
include/ slew.h	
Contains the slew rate controller wrapper for the motors	223
include/ toggle.h	233
include/ vlib.h	
Contains misc helpful functions	240
include/ vmath.h	
Vex Specific Math Functions, includes: Cartesian to polar coordinates	245
src/ auto.c	
File for autonomous code	252
src/ battery.c	256
src/ claw.c	260
src/ controller.c	266
src/ drive.c	268
src/ encoders.c	275
src/ init.c	
File for initialization code	279
src/ lcd.c	282
src/ lifter.c	293
src/ localization.c	304
src/ log.c	312
src/ matrix.c	322
src/ menu.c	344
src/ mobile_goal_intake.c	355

src/ opcontrol.c	
File for operator control code	360
src/ partner.c	363
src/ slew.c	366
src/ toggle.c	375
src/ vlib.c	382
src/ vmath.c	387

4 Data Structure Documentation

4.1 `_matrix` Struct Reference

```
#include "matrix.h"
```

Data Fields

- `double * data`
- `int height`
- `int width`

4.1.1 Detailed Description

A struct representing a matrix

Definition at line **14** of file **matrix.h**.

4.1.2 Field Documentation

4.1.2.1 `data`

```
double* _matrix::data
```

Definition at line **17** of file **matrix.h**.

Referenced by `covarianceMatrix()`, `dotDiagonalMatrix()`, `dotProductMatrix()`, `freeMatrix()`, `identityMatrix()`, `makeMatrix()`, `meanMatrix()`, `multiplyMatrix()`, `printMatrix()`, `rowSwap()`, `scaleMatrix()`, `traceMatrix()`, and `transposeMatrix()`.

4.1.2.2 height

```
int _matrix::height
```

Definition at line **15** of file **matrix.h**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, **printMatrix()**, **rowSwap()**, **scaleMatrix()**, **traceMatrix()**, and **transposeMatrix()**.

4.1.2.3 width

```
int _matrix::width
```

Definition at line **16** of file **matrix.h**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, **printMatrix()**, **rowSwap()**, **scaleMatrix()**, **traceMatrix()**, and **transposeMatrix()**.

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

- include/ **matrix.h**

4.2 accelerometer_odometry Struct Reference

Data Fields

- double **x**
- double **y**

4.2.1 Detailed Description

Definition at line **17** of file **localization.c**.

4.2.2 Field Documentation

4.2.2.1 x

```
double accelerometer_odometry::x
```

Definition at line **18** of file **localization.c**.

4.2.2.2 y

```
double accelerometer_odometry::y
```

Definition at line **19** of file **localization.c**.

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

- src/ **localization.c**

4.3 cord Struct Reference

A struct that contains cartesian coordinates.

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

Data Fields

- float **x**
- float **y**

4.3.1 Detailed Description

A struct that contains cartesian coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line **32** of file **vmath.h**.

4.3.2 Field Documentation

4.3.2.1 x

float cord::x

the x coordinate

Definition at line **34** of file **vmath.h**.

Referenced by **get_joystick_cord()**, and **update_drive_motors()**.

4.3.2.2 y

float cord::y

the y coordinate

Definition at line **36** of file **vmath.h**.

Referenced by **get_joystick_cord()**, and **update_drive_motors()**.

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

- include/**vmath.h**

4.4 encoder_odemtry Struct Reference

Data Fields

- double **theta**
- double **x**
- double **y**

4.4.1 Detailed Description

Definition at line **11** of file **localization.c**.

4.4.2 Field Documentation

4.4.2.1 theta

```
double encoder_odemtry::theta
```

Definition at line **14** of file **localization.c**.

Referenced by **integrate_gyro_w()**.

4.4.2.2 x

```
double encoder_odemtry::x
```

Definition at line **12** of file **localization.c**.

4.4.2.3 y

```
double encoder_odemtry::y
```

Definition at line **13** of file **localization.c**.

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

- src/ **localization.c**

4.5 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**

4.5.1 Detailed Description

represents the state of the lcd buttons

Author

Chris Jerrett

Date

9/9/2017

Definition at line **48** of file **lcd.h**.

4.5.2 Field Documentation

4.5.2.1 left

```
button_state lcd_buttons::left
```

Definition at line **49** of file **lcd.h**.

Referenced by **lcd_get_pressed_buttons()**.

4.5.2.2 middle

```
button_state lcd_buttons::middle
```

Definition at line **50** of file **lcd.h**.

Referenced by **lcd_get_pressed_buttons()**.

4.5.2.3 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**

4.6 location Struct Reference

```
#include "localization.h"
```

Data Fields

- int **theta**
- int **x**
- int **y**

4.6.1 Detailed Description

Vector storing the cartesian cords and an angle

Definition at line **23** of file **localization.h**.

4.6.2 Field Documentation

4.6.2.1 theta

```
int location::theta
```

Definition at line **26** of file **localization.h**.

4.6.2.2 x

```
int location::x
```

Definition at line **24** of file **localization.h**.

4.6.2.3 y

```
int location::y
```

Definition at line **25** of file **localization.h**.

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

- include/ **localization.h**

4.7 menu_t Struct Reference

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

```
#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**
*contains the prompt to display on the first line. Step is how much the int menu will increase or 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 or 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 or decrease with each press. Defaults to 1.0f
- enum **menu_type** **type**
contains the type of menu.

4.7.1 Detailed Description

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via `denint_menu`.

Author

Chris Jerrett

Date

9/8/17

See also

- [menu.h](#) (p. 196)
- [menu_t](#) (p. 10)
- [create_menu](#) (p. 347)
- [init_menu](#)
- [display_menu](#) (p. 348)
- [menu_type](#) (p. 198)
- [denint_menu](#) (p. 348)

Definition at line **66** of file **menu.h**.

4.7.2 Field Documentation

4.7.2.1 current

```
int menu_t::current
```

contains the current index of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line **140** of file **menu.h**.

Referenced by **calculate_current_display()**, and **display_menu()**.

4.7.2.2 length

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

contains the length of options char**.

Author

Chris Jerrett

Date

9/8/17

Definition at line **86** of file **menu.h**.

Referenced by **calculate_current_display()**, and **init_menu_var()**.

4.7.2.3 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 **102** of file **menu.h**.

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

4.7.2.4 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 **126** of file **menu.h**.

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

4.7.2.5 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 **94** of file **menu.h**.

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

4.7.2.6 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 **118** of file **menu.h**.

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

4.7.2.7 options

char** menu_t::options

contains the array of string options.

Author

Chris Jerrett

Date

9/8/17

Definition at line **79** of file **menu.h**.

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

4.7.2.8 prompt

char* menu_t::prompt

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **147** of file **menu.h**.

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

4.7.2.9 step

```
int menu_t::step
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **110** of file **menu.h**.

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

4.7.2.10 step_f

```
float menu_t::step_f
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **134** of file **menu.h**.

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

4.7.2.11 type

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

contains the type of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line **72** 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**

4.8 polar_cord Struct Reference

A struct that contains polar coordinates.

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

Data Fields

- float **angle**
- float **magnitude**

4.8.1 Detailed Description

A struct that contains polar coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line **20** of file **vmath.h**.

4.8.2 Field Documentation

4.8.2.1 angle

```
float polar_cord::angle
```

the angle of the vector

Definition at line **22** of file **vmath.h**.

Referenced by **cartesian_to_polar()**.

4.8.2.2 magnitue

```
float polar_cord::magnitue
```

the magnitude of the vector

Definition at line **24** of file **vmath.h**.

Referenced by **cartesian_to_polar()**.

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

- include/ **vmath.h**

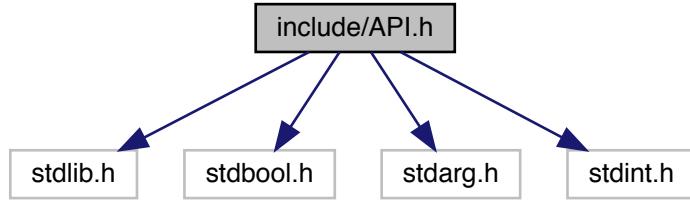
5 File Documentation

5.1 include/API.h File Reference

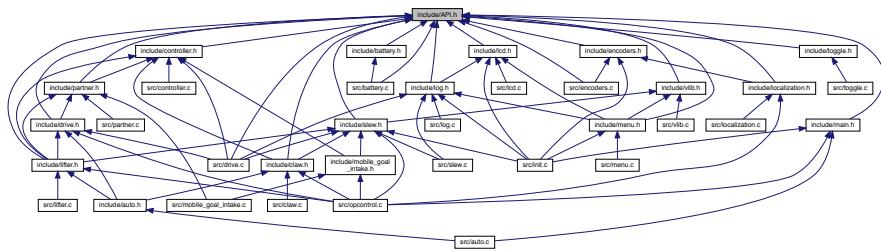
Provides the high-level user functionality intended for use by typical VEX Cortex programmers.

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdarg.h>
```

```
#include <stdint.h>
Include dependency graph for API.h:
```



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



Macros

- #define **ACCEL_X** 5
- #define **ACCEL_Y** 6
- #define **BOARD_NR_ADC_PINS** 8
- #define **BOARD_NR_GPIO_PINS** 27
- #define **EOF** ((int)-1)
- #define **FILE PROS_FILE**
- #define **HIGH** 1
- #define **IME_ADDR_MAX** 0x1F
- #define **INPUT** 0x0A
- #define **INPUT_ANALOG** 0x00
- #define **INPUT_FLOATING** 0x04
- #define **INTERRUPT_EDGE_BOTH** 3
- #define **INTERRUPT_EDGE_FALLING** 2
- #define **INTERRUPT_EDGE_RISING** 1
- #define **JOY_DOWN** 1
- #define **JOY_LEFT** 2
- #define **JOY_RIGHT** 8
- #define **JOY_UP** 4

- #define **LCD_BTN_CENTER** 2
- #define **LCD_BTN_LEFT** 1
- #define **LCD_BTN_RIGHT** 4
- #define **LOW** 0
- #define **OUTPUT** 0x01
- #define **OUTPUT_OD** 0x05
- #define **SEEK_CUR** 1
- #define **SEEK_END** 2
- #define **SEEK_SET** 0
- #define **SERIAL_8N1** 0x0000
- #define **SERIAL_DATABITS_8** 0x0000
- #define **SERIAL_DATABITS_9** 0x1000
- #define **SERIAL_PARITY_EVEN** 0x0400
- #define **SERIAL_PARITY_NONE** 0x0000
- #define **SERIAL_PARITY_ODD** 0x0600
- #define **SERIAL_STOPBITS_1** 0x0000
- #define **SERIAL_STOPBITS_2** 0x2000
- #define **stdin** ((**PROS_FILE** *))3)
- #define **stdout** ((**PROS_FILE** *)3)
- #define **TASK_DEAD** 0
- #define **TASK_DEFAULT_STACK_SIZE** 512
- #define **TASK_MAX** 16
- #define **TASK_MAX_PRIORITIES** 6
- #define **TASK_MINIMAL_STACK_SIZE** 64
- #define **TASK_PRIORITY_DEFAULT** 2
- #define **TASK_PRIORITY_HIGHEST** (**TASK_MAX_PRIORITIES** - 1)
- #define **TASK_PRIORITY_LOWEST** 0
- #define **TASK_RUNNABLE** 2
- #define **TASK_RUNNING** 1
- #define **TASK_SLEEPING** 3
- #define **TASK_SUSPENDED** 4
- #define **uart1** ((**PROS_FILE** *))1)
- #define **uart2** ((**PROS_FILE** *))2)
- #define **ULTRA_BAD_RESPONSE** -1

Typedefs

- typedef void * **Encoder**
- typedef void * **Gyro**
- typedef void(* **InterruptHandler**) (unsigned char pin)
- typedef void * **Mutex**
- typedef int **PROS_FILE**
- typedef void * **Semaphore**
- typedef void(* **TaskCode**) (void *)
- typedef void * **TaskHandle**
- typedef void * **Ultrasonic**

Functions

- void **__attribute__ ((format(printf, 3, 4)))** lcdPrint(**PROS_FILE** *lcdPort)
- int **analogCalibrate** (unsigned char channel)
- int **analogRead** (unsigned char channel)
- int **analogReadCalibrated** (unsigned char channel)
- int **analogReadCalibratedHR** (unsigned char channel)
- void **delay** (const unsigned long time)
- void **delayMicroseconds** (const unsigned long us)
- bool **digitalRead** (unsigned char pin)
- void **digitalWrite** (unsigned char pin, bool value)
- int **encoderGet** (**Encoder** enc)
- **Encoder encoderInit** (unsigned char portTop, unsigned char portBottom, bool **reverse**)
- void **encoderReset** (**Encoder** enc)
- void **encoderShutdown** (**Encoder** enc)
- void **fclose** (**PROS_FILE** *stream)
- int **fcount** (**PROS_FILE** *stream)
- int **fdelete** (const char *file)
- int **feof** (**PROS_FILE** *stream)
- int **fflush** (**PROS_FILE** *stream)
- int **fgetc** (**PROS_FILE** *stream)
- char * **fgets** (char *str, int num, **PROS_FILE** *stream)
- **PROS_FILE** * **fopen** (const char *file, const char * mode)
- void **fprint** (const char *string, **PROS_FILE** *stream)
- int **fprintf** (**PROS_FILE** *stream, const char * formatString,...)
- int **fputc** (int value, **PROS_FILE** *stream)
- int **fputs** (const char *string, **PROS_FILE** *stream)
- size_t **fread** (void *ptr, size_t size, size_t count, **PROS_FILE** *stream)
- int **fseek** (**PROS_FILE** *stream, long int offset, int origin)
- long int **ftell** (**PROS_FILE** *stream)
- size_t **fwrite** (const void *ptr, size_t size, size_t count, **PROS_FILE** *stream)
- int **getchar** ()
- int **gyroGet** (**Gyro** gyro)
- **Gyro gyroInit** (unsigned char port, unsigned short multiplier)
- void **gyroReset** (**Gyro** gyro)
- void **gyroShutdown** (**Gyro** gyro)
- bool **i2cRead** (uint8_t addr, uint8_t *data, uint16_t count)
- bool **i2cReadRegister** (uint8_t addr, uint8_t reg, uint8_t *value, uint16_t count)
- bool **i2cWrite** (uint8_t addr, uint8_t *data, uint16_t count)
- bool **i2cWriteRegister** (uint8_t addr, uint8_t reg, uint16_t value)
- bool **imeGet** (unsigned char address, int *value)
- bool **imeGetVelocity** (unsigned char address, int *value)
- unsigned int **imeInitializeAll** ()
- bool **imeReset** (unsigned char address)
- void **imeShutdown** ()
- void **ioClearInterrupt** (unsigned char pin)
- void **ioSetInterrupt** (unsigned char pin, unsigned char edges, **InterruptHandler** handler)
- bool **isAutonomous** ()
- bool **isEnabled** ()
- bool **isJoystickConnected** (unsigned char joystick)
- bool **isOnline** ()

- int **joystickGetAnalog** (unsigned char **joystick**, unsigned char axis)
- bool **joystickGetDigital** (unsigned char **joystick**, unsigned char buttonGroup, unsigned char button)
- void **LcdClear** (**PROS_FILE** *lcdPort)
- void **LcdInit** (**PROS_FILE** *lcdPort)
- void unsigned char const char unsigned int **LcdReadButtons** (**PROS_FILE** *lcdPort)
- void **LcdSetBacklight** (**PROS_FILE** *lcdPort, bool backlight)
- void **LcdSetText** (**PROS_FILE** *lcdPort, unsigned char **line**, const char *buffer)
- void **LcdShutdown** (**PROS_FILE** *lcdPort)
- unsigned long **micros** ()
- unsigned long **millis** ()
- int **motorGet** (unsigned char channel)
- void **motorSet** (unsigned char channel, int speed)
- void **motorStop** (unsigned char channel)
- void **motorStopAll** ()
- **Mutex** **mutexCreate** ()
- void **mutexDelete** (**Mutex** mutex)
- bool **mutexGive** (**Mutex** mutex)
- bool **mutexTake** (**Mutex** mutex, const unsigned long blockTime)
- void **pinMode** (unsigned char pin, unsigned char **mode**)
- unsigned int **powerLevelBackup** ()
- unsigned int **powerLevelMain** ()
- void **print** (const char *string)
- int **printf** (const char * **formatString**,...)
- int **putchar** (int value)
- int **puts** (const char *string)
- **Semaphore** **semaphoreCreate** ()
- void **semaphoreDelete** (**Semaphore** semaphore)
- bool **semaphoreGive** (**Semaphore** semaphore)
- bool **semaphoreTake** (**Semaphore** semaphore, const unsigned long blockTime)
- void **setTeamName** (const char *name)
- int **sprintf** (char *buffer, size_t limit, const char * **formatString**,...)
- void **speakerInit** ()
- void **speakerPlayArray** (const char **songs)
- void **speakerPlayRttl** (const char *song)
- void **speakerShutdown** ()
- int **sprintf** (char *buffer, const char * **formatString**,...)
- void **standaloneModeEnable** ()
- **TaskHandle** **taskCreate** (**TaskCode** taskCode, const unsigned int stackDepth, void *parameters, const unsigned int priority)
- void **taskDelay** (const unsigned long msToDelay)
- void **taskDelayUntil** (unsigned long *previousWakeTime, const unsigned long cycleTime)
- void **taskDelete** (**TaskHandle** taskToDelete)
- unsigned int **taskGetCount** ()
- unsigned int **taskGetState** (**TaskHandle** task)
- unsigned int **taskPriorityGet** (const **TaskHandle** task)
- void **taskPrioritySet** (**TaskHandle** task, const unsigned int newPriority)
- void **taskResume** (**TaskHandle** taskToResume)
- **TaskHandle** **taskRunLoop** (void(*fn)(void), const unsigned long increment)
- void **taskSuspend** (**TaskHandle** taskToSuspend)
- int **ultrasonicGet** (**Ultrasonic** ult)
- **Ultrasonic** **ultrasonicInit** (unsigned char portEcho, unsigned char portPing)

- void **ultrasonicShutdown** (Ultrasonic ult)
- void **uartInit** (PROS_FILE *uart, unsigned int baud, unsigned int flags)
- void **uartShutdown** (PROS_FILE *uart)
- void **wait** (const unsigned long time)
- void **waitFor** (unsigned long *previousWakeTime, const unsigned long time)
- void **watchdogInit** ()

Variables

- void unsigned char const char * **formatString**
- void unsigned char **line**

5.1.1 Detailed Description

Provides the high-level user functionality intended for use by typical VEX Cortex programmers.

This file should be included for you in the predefined stubs in each new VEX Cortex PROS project through the inclusion of "main.h". In any new C source file, it is advisable to include **main.h** (p. 170) instead of referencing **API.h** (p. 17) by name, to better handle any nomenclature changes to this file or its contents.

Copyright (c) 2011-2016, Purdue University ACM SIGBots. All rights reserved.

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>.

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

Definition in file **API.h**.

5.1.2 Macro Definition Documentation

5.1.2.1 ACCEL_X

```
#define ACCEL_X 5
```

Analog axis for the X acceleration from the VEX Joystick.

Definition at line **56** of file **API.h**.

5.1.2.2 ACCEL_Y

```
#define ACCEL_Y 6
```

Analog axis for the Y acceleration from the VEX Joystick.

Definition at line **60** of file **API.h**.

5.1.2.3 BOARD_NR_ADC_PINS

```
#define BOARD_NR_ADC_PINS 8
```

There are 8 available analog I/O on the Cortex.

Definition at line **141** of file **API.h**.

5.1.2.4 BOARD_NR_GPIO_PINS

```
#define BOARD_NR_GPIO_PINS 27
```

There are 27 available I/O on the Cortex that can be used for digital communication.

This excludes the crystal ports but includes the Communications, Speaker, and Analog ports.

The motor ports are not on the Cortex and are thus excluded from this count. Pin 0 is the Speaker port, pins 1-12 are the standard Digital I/O, 13-20 are the Analog I/O, 21+22 are UART1, 23+24 are UART2, and 25+26 are the I2C port.

Definition at line **151** of file **API.h**.

5.1.2.5 EOF

```
#define EOF ((int)-1)
```

EOF is a value evaluating to -1.

Definition at line **846** of file **API.h**.

5.1.2.6 FILE

```
#define FILE PROS_FILE
```

For convenience, FILE is defined as PROS_FILE if it wasn't already defined. This provides backwards compatibility with PROS, but also allows libraries such as newlib to be incorporated into PROS projects. If you're not using C++/newlib, you can disregard this and just use FILE.

Definition at line **759** of file **API.h**.

5.1.2.7 HIGH

```
#define HIGH 1
```

Used for **digitalWrite()** (p. 40) to specify a logic HIGH state to output.

In reality, using any non-zero expression or "true" will work to set a pin to HIGH.

Definition at line **157** of file **API.h**.

5.1.2.8 IME_ADDR_MAX

```
#define IME_ADDR_MAX 0x1F
```

IME addresses end at 0x1F. Actually using more than 10 (address 0x1A) encoders will cause unreliable communications.

Definition at line **458** of file **API.h**.

5.1.2.9 INPUT

```
#define INPUT 0x0A
```

pinMode() (p. 74) state for digital input, with pullup.

This is the default state for the 12 Digital pins. The pullup causes the input to read as "HIGH" when unplugged, but is fairly weak and can safely be driven by most sources. Many VEX digital sensors rely on this behavior and cannot be used with INPUT_FLOATING.

Definition at line **172** of file **API.h**.

5.1.2.10 INPUT_ANALOG

```
#define INPUT_ANALOG 0x00
```

pinMode() (p. 74) state for analog inputs.

This is the default state for the 8 Analog pins and the Speaker port. This only works on pins with analog input capabilities; use anywhere else results in undefined behavior.

Definition at line **179** of file **API.h**.

5.1.2.11 INPUT_FLOATING

```
#define INPUT_FLOATING 0x04
```

pinMode() (p. 74) state for digital input, without pullup.

Beware of power consumption, as digital inputs left "floating" may switch back and forth and cause spurious interrupts.

Definition at line **186** of file **API.h**.

5.1.2.12 INTERRUPT_EDGE_BOTH

```
#define INTERRUPT_EDGE_BOTH 3
```

When used in **ioSetInterrupt()** (p. 60), triggers an interrupt on both rising and falling edges (LOW to HIGH or HIGH to LOW).

Definition at line **327** of file **API.h**.

5.1.2.13 INTERRUPT_EDGE_FALLING

```
#define INTERRUPT_EDGE_FALLING 2
```

When used in **ioSetInterrupt()** (p. 60), triggers an interrupt on falling edges (HIGH to LOW).

Definition at line **322** of file **API.h**.

5.1.2.14 INTERRUPT_EDGE_RISING

```
#define INTERRUPT_EDGE_RISING 1
```

When used in **ioSetInterrupt()** (p. 60), triggers an interrupt on rising edges (LOW to HIGH).

Definition at line **318** of file **API.h**.

5.1.2.15 JOY_DOWN

```
#define JOY_DOWN 1
```

DOWN button (valid on channels 5, 6, 7, 8)

Definition at line **40** of file **API.h**.

Referenced by **buttonGetState()**, and **updateIntake()**.

5.1.2.16 JOY_LEFT

```
#define JOY_LEFT 2
```

LEFT button (valid on channels 7, 8)

Definition at line **44** of file **API.h**.

Referenced by **buttonGetState()**, and **update_control()**.

5.1.2.17 JOY_RIGHT

```
#define JOY_RIGHT 8
```

RIGHT button (valid on channels 7, 8)

Definition at line **52** of file **API.h**.

Referenced by **buttonGetState()**, and **update_control()**.

5.1.2.18 JOY_UP

```
#define JOY_UP 4
```

UP button (valid on channels 5, 6, 7, 8)

Definition at line **48** of file **API.h**.

Referenced by **buttonGetState()**, and **updateIntake()**.

5.1.2.19 LCD_BTN_CENTER

```
#define LCD_BTN_CENTER 2
```

CENTER button on LCD for use with **IcdReadButtons()** (p. 66)

Definition at line **1144** of file **API.h**.

Referenced by **buttonGetState()**.

5.1.2.20 LCD_BTN_LEFT

```
#define LCD_BTN_LEFT 1
```

LEFT button on LCD for use with **IcdReadButtons()** (p. 66)

Definition at line **1140** of file **API.h**.

Referenced by **buttonGetState()**.

5.1.2.21 LCD_BTN_RIGHT

```
#define LCD_BTN_RIGHT 4
```

RIGHT button on LCD for use with **IcdReadButtons()** (p. 66)

Definition at line **1148** of file **API.h**.

Referenced by **buttonGetState()**.

5.1.2.22 LOW

```
#define LOW 0
```

Used for **digitalWrite()** (p. 40) to specify a logic LOW state to output.

In reality, using a zero expression or "false" will work to set a pin to LOW.

Definition at line **163** of file **API.h**.

5.1.2.23 OUTPUT

```
#define OUTPUT 0x01
```

pinMode() (p. 74) state for digital output, push-pull.

This is the mode which should be used to output a digital HIGH or LOW value from the Cortex. This mode is useful for pneumatic solenoid valves and VEX LEDs.

Definition at line **193** of file **API.h**.

5.1.2.24 OUTPUT_OD

```
#define OUTPUT_OD 0x05
```

pinMode() (p. 74) state for open-drain outputs.

This is useful in a few cases for external electronics and should not be used for the VEX solenoid or LEDs.

Definition at line **200** of file **API.h**.

5.1.2.25 SEEK_CUR

```
#define SEEK_CUR 1
```

SEEK_CUR is used in **fseek()** (p. 50) to denote an relative position in bytes from the current file location.

Definition at line **861** of file **API.h**.

5.1.2.26 SEEK_END

```
#define SEEK_END 2
```

SEEK_END is used in **fseek()** (p. 50) to denote an absolute position in bytes from the end of the file. The offset will most likely be negative in this case.

Definition at line **868** of file **API.h**.

5.1.2.27 SEEK_SET

```
#define SEEK_SET 0
```

SEEK_SET is used in **fseek()** (p. 50) to denote an absolute position in bytes from the start of the file.

Definition at line **854** of file **API.h**.

5.1.2.28 SERIAL_8N1

```
#define SERIAL_8N1 0x0000
```

Specifies the default serial settings when used in **uartInit()** (p. 88)

Definition at line **793** of file **API.h**.

5.1.2.29 SERIAL_DATABITS_8

```
#define SERIAL_DATABITS_8 0x0000
```

Bit mask for **uartInit()** (p. 88) for 8 data bits (typical)

Definition at line **765** of file **API.h**.

5.1.2.30 SERIAL_DATABITS_9

```
#define SERIAL_DATABITS_9 0x1000
```

Bit mask for **uartInit()** (p. 88) for 9 data bits

Definition at line **769** of file **API.h**.

5.1.2.31 SERIAL_PARITY_EVEN

```
#define SERIAL_PARITY_EVEN 0x0400
```

Bit mask for **uartInit()** (p. 88) for Even parity

Definition at line **785** of file **API.h**.

5.1.2.32 SERIAL_PARITY_NONE

```
#define SERIAL_PARITY_NONE 0x0000
```

Bit mask for **uartInit()** (p. 88) for No parity (typical)

Definition at line **781** of file **API.h**.

5.1.2.33 SERIAL_PARITY_ODD

```
#define SERIAL_PARITY_ODD 0x0600
```

Bit mask for **uartInit()** (p. 88) for Odd parity

Definition at line **789** of file **API.h**.

5.1.2.34 SERIAL_STOPBITS_1

```
#define SERIAL_STOPBITS_1 0x0000
```

Bit mask for **uartInit()** (p. 88) for 1 stop bit (typical)

Definition at line **773** of file **API.h**.

5.1.2.35 SERIAL_STOPBITS_2

```
#define SERIAL_STOPBITS_2 0x2000
```

Bit mask for **uartInit()** (p. 88) for 2 stop bits

Definition at line **777** of file **API.h**.

5.1.2.36 stdin

```
#define stdin (( PROS_FILE *)3)
```

The standard input stream uses the PC debug terminal.

Definition at line **832** of file **API.h**.

5.1.2.37 stdout

```
#define stdout (( PROS_FILE *)3)
```

The standard output stream uses the PC debug terminal.

Definition at line **828** of file **API.h**.

5.1.2.38 TASK_DEAD

```
#define TASK_DEAD 0
```

Constant returned from **taskGetState()** (p. 84) when the task is dead or nonexistent.

Definition at line **1275** of file **API.h**.

5.1.2.39 TASK_DEFAULT_STACK_SIZE

```
#define TASK_DEFAULT_STACK_SIZE 512
```

The recommended stack size for a new task that does an average amount of work. This stack size is used for default tasks such as **autonomous()** (p. 172).

This is probably OK for 4-5 levels of function calls and the use of **printf()** (p. 75) with several arguments. Tasks requiring deep recursion or large local buffers will need a bigger stack.

Definition at line **1262** of file **API.h**.

5.1.2.40 TASK_MAX

```
#define TASK_MAX 16
```

Only this many tasks can exist at once. Attempts to create further tasks will not succeed until tasks end or are destroyed, AND the idle task cleans them up.

Changing this value will not change the limit without a kernel recompile. The idle task and VEX daemon task count against the limit. The user **autonomous()** (p. 172) or teleop() also counts against the limit, so 12 tasks usually remain for other uses.

Definition at line **1232** of file **API.h**.

5.1.2.41 TASK_MAX_PRIORITIES

```
#define TASK_MAX_PRIORITIES 6
```

The maximum number of available task priorities, which run from 0 to 5.

Changing this value will not change the priority count without a kernel recompile.

Definition at line **1238** of file **API.h**.

5.1.2.42 TASK_MINIMAL_STACK_SIZE

```
#define TASK_MINIMAL_STACK_SIZE 64
```

The minimum stack depth for a task. Scheduler state is stored on the stack, so even if the task never uses the stack, at least this much space must be allocated.

Function calls and other seemingly innocent constructs may place information on the stack. Err on the side of a larger stack when possible.

Definition at line **1270** of file **API.h**.

5.1.2.43 TASK_PRIORITY_DEFAULT

```
#define TASK_PRIORITY_DEFAULT 2
```

The default task priority, which should be used for most tasks.

Default tasks such as **autonomous()** (p. 172) inherit this priority.

Definition at line **1249** of file **API.h**.

5.1.2.44 TASK_PRIORITY_HIGHEST

```
#define TASK_PRIORITY_HIGHEST ( TASK_MAX_PRIORITIES - 1 )
```

The highest priority that can be assigned to a task. Unlike the lowest priority, this priority can be safely used without hampering interrupts. Beware of deadlock.

Definition at line **1254** of file **API.h**.

5.1.2.45 TASK_PRIORITY_LOWEST

```
#define TASK_PRIORITY_LOWEST 0
```

The lowest priority that can be assigned to a task, which puts it on a level with the idle task. This may cause severe performance problems and is generally not recommended.

Definition at line **1243** of file **API.h**.

5.1.2.46 TASK_RUNNABLE

```
#define TASK_RUNNABLE 2
```

Constant returned from **taskGetState()** (p. 84) when the task exists and is available to run, but not currently running.

Definition at line **1284** of file **API.h**.

5.1.2.47 TASK_RUNNING

```
#define TASK_RUNNING 1
```

Constant returned from **taskGetState()** (p. 84) when the task is actively executing.

Definition at line **1279** of file **API.h**.

5.1.2.48 TASK_SLEEPING

```
#define TASK_SLEEPING 3
```

Constant returned from **taskGetState()** (p. 84) when the task is delayed or blocked waiting for a semaphore, mutex, or I/O operation.

Definition at line **1289** of file **API.h**.

5.1.2.49 TASK_SUSPENDED

```
#define TASK_SUSPENDED 4
```

Constant returned from **taskGetState()** (p. 84) when the task is suspended using **taskSuspend()** (p. 86).

Definition at line **1293** of file **API.h**.

5.1.2.50 uart1

```
#define uart1 (( PROS_FILE *)1)
```

UART 1 on the Cortex; must be opened first using **uartInit()** (p. 88).

Definition at line **836** of file **API.h**.

Referenced by **buttonGetState()**.

5.1.2.51 uart2

```
#define uart2 (( PROS_FILE *)2)
```

UART 2 on the Cortex; must be opened first using **uartInit()** (p. 88).

Definition at line **840** of file **API.h**.

5.1.2.52 ULTRA_BAD_RESPONSE

```
#define ULTRA_BAD_RESPONSE -1
```

This value is returned if the sensor cannot find a reasonable value to return.

Definition at line **650** of file **API.h**.

5.1.3 Typedef Documentation

5.1.3.1 Encoder

```
typedef void* Encoder
```

Reference type for an initialized encoder.

Encoder information is stored as an opaque pointer to a structure in memory; as this is a pointer type, it can be safely passed or stored by value.

Definition at line **605** of file **API.h**.

5.1.3.2 Gyro

```
typedef void* Gyro
```

Reference type for an initialized gyro.

Gyro information is stored as an opaque pointer to a structure in memory; as this is a pointer type, it can be safely passed or stored by value.

Definition at line **548** of file **API.h**.

5.1.3.3 InterruptHandler

```
typedef void(* InterruptHandler) (unsigned char pin)
```

Type definition for interrupt handlers. Such functions must accept one argument indicating the pin which changed.

Definition at line **332** of file **API.h**.

5.1.3.4 Mutex

```
typedef void* Mutex
```

Type by which mutexes are referenced.

As this is a pointer type, it can be safely passed or stored by value.

Definition at line **1306** of file **API.h**.

5.1.3.5 PROS_FILE

```
typedef int PROS_FILE
```

PROS_FILE is an integer referring to a stream for the standard I/O functions.

PROS_FILE * is the standard library method of referring to a file pointer, even though there is actually nothing there.

Definition at line **750** of file **API.h**.

5.1.3.6 Semaphore

```
typedef void* Semaphore
```

Type by which semaphores are referenced.

As this is a pointer type, it can be safely passed or stored by value.

Definition at line **1312** of file **API.h**.

5.1.3.7 TaskCode

```
typedef void(* TaskCode) (void *)
```

Type for defining task functions. Task functions must accept one parameter of type "void *"; they need not use it.

For example:

```
void MyTask(void *ignore) { while (1); }
```

Definition at line **1323** of file **API.h**.

5.1.3.8 TaskHandle

```
typedef void* TaskHandle
```

Type by which tasks are referenced.

As this is a pointer type, it can be safely passed or stored by value.

Definition at line **1300** of file **API.h**.

5.1.3.9 Ultrasonic

```
typedef void* Ultrasonic
```

Reference type for an initialized ultrasonic sensor.

Ultrasonic information is stored as an opaque pointer to a structure in memory; as this is a pointer type, it can be safely passed or stored by value.

Definition at line **658** of file **API.h**.

5.1.4 Function Documentation

5.1.4.1 **__attribute__()**

```
void __attribute__ (
    format( printf, 3, 4))
```

Prints the formatted string to the attached LCD.

The output string will be truncated as necessary to fit on the LCD screen, 16 characters wide. It is probably better to generate the string in a local buffer and use **LcdSetText()** (p. 67) but this method is provided for convenience.

Parameters

<i>lcdPort</i>	the LCD to write, either <code>uart1</code> or <code>uart2</code>
<i>line</i>	the LCD line to write, either 1 or 2
<i>formatString</i>	the format string as specified in fprintf() (p. 47)

5.1.4.2 **analogCalibrate()**

```
int analogCalibrate (
    unsigned char channel)
```

Calibrates the analog sensor on the specified channel.

This method assumes that the true sensor value is not actively changing at this time and computes an average from approximately 500 samples, 1 ms apart, for a 0.5 s period of calibration. The average value thus calculated is returned and stored for later calls to the **analogReadCalibrated()** (p. 38) and **analogReadCalibratedHR()** (p. 38) functions. These functions will return the difference between this value and the current sensor value when called.

Do not use this function in **initializeIO()** (p. 175), or when the sensor value might be unstable (gyro rotation, accelerometer movement).

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

<i>channel</i>	the channel to calibrate from 1-8
----------------	-----------------------------------

Returns

the average sensor value computed by this function

5.1.4.3 analogRead()

```
int analogRead (
    unsigned char channel )
```

Reads an analog input channel and returns the 12-bit value.

The value returned is undefined if the analog pin has been switched to a different mode. This function is Wiring-compatible with the exception of the larger output range. The meaning of the returned value varies depending on the sensor attached.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

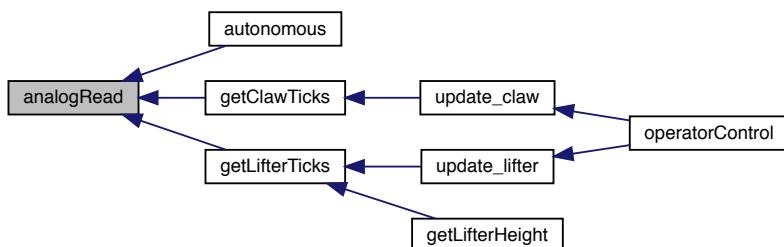
<i>channel</i>	the channel to read from 1-8
----------------	------------------------------

Returns

the analog sensor value, where a value of 0 reflects an input voltage of nearly 0 V and a value of 4095 reflects an input voltage of nearly 5 V

Referenced by **autonomous()**, **getClawTicks()**, and **getLifterTicks()**.

Here is the caller graph for this function:



5.1.4.4 analogReadCalibrated()

```
int analogReadCalibrated (
    unsigned char channel )
```

Reads the calibrated value of an analog input channel.

The **analogCalibrate()** (p. 36) function must be run first on that channel. This function is inappropriate for sensor values intended for integration, as round-off error can accumulate causing drift over time. Use **analogReadCalibratedHR()** (p. 38) instead.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

<i>channel</i>	the channel to read from 1-8
----------------	------------------------------

Returns

the difference of the sensor value from its calibrated default from -4095 to 4095

5.1.4.5 analogReadCalibratedHR()

```
int analogReadCalibratedHR (
    unsigned char channel )
```

Reads the calibrated value of an analog input channel 1-8 with enhanced precision.

The **analogCalibrate()** (p. 36) function must be run first. This is intended for integrated sensor values such as gyros and accelerometers to reduce drift due to round-off, and should not be used on a sensor such as a line tracker or potentiometer.

The value returned actually has 16 bits of "precision", even though the ADC only reads 12 bits, so that errors induced by the average value being between two values come out in the wash when integrated over time. Think of the value as the true value times 16.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

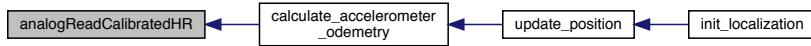
<i>channel</i>	the channel to read from 1-8
----------------	------------------------------

Returns

the difference of the sensor value from its calibrated default from -16384 to 16384

Referenced by **calculate_accelerometer_odometry()**.

Here is the caller graph for this function:



5.1.4.6 delay()

```
void delay (
    const unsigned long time )
```

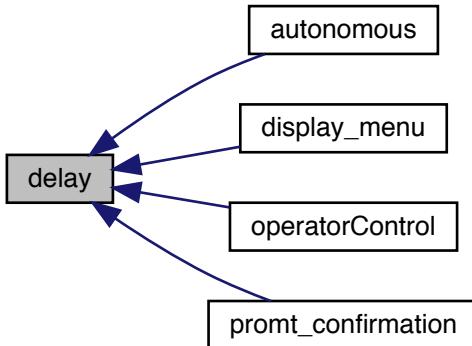
Wiring-compatible alias of **taskDelay()** (p. 82).

Parameters

<i>time</i>	the duration of the delay in milliseconds (1 000 milliseconds per second)
-------------	---

Referenced by **autonomous()**, **display_menu()**, **operatorControl()**, and **prompt_confirmation()**.

Here is the caller graph for this function:



5.1.4.7 delayMicroseconds()

```
void delayMicroseconds (
    const unsigned long us )
```

Wait for approximately the given number of microseconds.

The method used for delaying this length of time may vary depending on the argument. The current task will always be delayed by at least the specified period, but possibly much more depending on CPU load. In general, this function is less reliable than **delay()** (p. 39). Using this function in a loop may hog processing time from other tasks.

Parameters

<i>us</i>	the duration of the delay in microseconds (1 000 000 microseconds per second)
-----------	---

5.1.4.8 digitalWrite()

```
bool digitalWrite (
    unsigned char pin )
```

Gets the digital value (1 or 0) of a pin configured as a digital input.

If the pin is configured as some other mode, the digital value which reflects the current state of the pin is returned, which may or may not differ from the currently set value. The return value is undefined for pins configured as Analog inputs, or for ports in use by a Communications interface. This function is Wiring-compatible.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

<i>pin</i>	the pin to read from 1-26
------------	---------------------------

Returns

true if the pin is HIGH, or false if it is LOW

5.1.4.9 digitalWrite()

```
void digitalWrite (
    unsigned char pin,
    bool value )
```

Sets the digital value (1 or 0) of a pin configured as a digital output.

If the pin is configured as some other mode, behavior is undefined. This function is Wiring-compatible.

Parameters

<i>pin</i>	the pin to write from 1-26
<i>value</i>	an expression evaluating to "true" or "false" to set the output to HIGH or LOW respectively, or the constants HIGH or LOW themselves

5.1.4.10 encoderGet()

```
int encoderGet (
    Encoder enc )
```

Gets the number of ticks recorded by the encoder.

There are 360 ticks in one revolution.

Parameters

<i>enc</i>	the Encoder object from encoderInit() (p. 41) to read
------------	--

Returns

the signed and cumulative number of counts since the last start or reset

5.1.4.11 encoderInit()

```
Encoder encoderInit (
    unsigned char portTop,
    unsigned char portBottom,
    bool reverse )
```

Initializes and enables a quadrature encoder on two digital ports.

Neither the top port nor the bottom port can be digital port 10. NULL will be returned if either port is invalid or the encoder is already in use. Initializing an encoder implicitly resets its count.

Parameters

<i>portTop</i>	the "top" wire from the encoder sensor with the removable cover side UP
<i>portBottom</i>	the "bottom" wire from the encoder sensor
<i>reverse</i>	if "true", the sensor will count in the opposite direction

Returns

an Encoder object to be stored and used for later calls to encoder functions

5.1.4.12 encoderReset()

```
void encoderReset (
    Encoder enc )
```

Resets the encoder to zero.

It is safe to use this method while an encoder is enabled. It is not necessary to call this method before stopping or starting an encoder.

Parameters

<i>enc</i>	the Encoder object from encoderInit() (p. 41) to reset
------------	---

5.1.4.13 encoderShutdown()

```
void encoderShutdown (
    Encoder enc )
```

Stops and disables the encoder.

Encoders use processing power, so disabling unused encoders increases code performance. The encoder's count will be retained.

Parameters

<i>enc</i>	the Encoder object from encoderInit() (p. 41) to stop
------------	--

5.1.4.14 fclose()

```
void fclose (
    PROS_FILE * stream )
```

Closes the specified file descriptor. This function does not work on communication ports; use **usartShutdown()** (p. 88) instead.

Parameters

<i>stream</i>	the file descriptor to close from fopen() (p. 46)
---------------	--

5.1.4.15 fcount()

```
int fcount (
    PROS_FILE * stream )
```

Returns the number of characters that can be read without blocking (the number of characters available) from the specified stream. This only works for communication ports and files in Read mode; for files in Write mode, 0 is always returned.

This function may underestimate, but will not overestimate, the number of characters which meet this criterion.

Parameters

<i>stream</i>	the stream to read (stdin, uart1, uart2, or an open file in Read mode)
---------------	--

Returns

the number of characters which meet this criterion; if this number cannot be determined, returns 0

5.1.4.16 fdelete()

```
int fdelete (
    const char * file )
```

Delete the specified file if it exists and is not currently open.

The file will actually be erased from memory on the next re-boot. A physical power cycle is required to purge deleted files and free their allocated space for new files to be written. Deleted files are still considered inaccessible to **fopen()** (p. 46) in Read mode.

Parameters

<i>file</i>	the file name to erase
-------------	------------------------

Returns

0 if the file was deleted, or 1 if the file could not be found

5.1.4.17 feof()

```
int feof (
    PROS_FILE * stream )
```

Checks to see if the specified stream is at its end. This only works for communication ports and files in Read mode; for files in Write mode, 1 is always returned.

Parameters

<i>stream</i>	the channel to check (stdin, uart1, uart2, or an open file in Read mode)
---------------	--

Returns

0 if the stream is not at EOF, or 1 otherwise.

5.1.4.18 fflush()

```
int fflush (
    PROS_FILE * stream )
```

Flushes the data on the specified file channel open in Write mode. This function has no effect on a communication port or a file in Read mode, as these streams are always flushed as quickly as possible by the kernel.

Successful completion of an fflush function on a file in Write mode cannot guarantee that the file is valid until **fclose()** (p. 42) is used on that file descriptor.

Parameters

<i>stream</i>	the channel to flush (an open file in Write mode)
---------------	---

Returns

0 if the data was successfully flushed, EOF otherwise

5.1.4.19 fgetc()

```
int fgetc (
    PROS_FILE * stream )
```

Reads and returns one character from the specified stream, blocking until complete.

Do not use **fgetc()** (p. 45) on a VEX LCD port; deadlock may occur.

Parameters

<i>stream</i>	the stream to read (stdin, uart1, uart2, or an open file in Read mode)
---------------	--

Returns

the next character from 0 to 255, or -1 if no character can be read

5.1.4.20 fgets()

```
char* fgets (
    char * str,
    int num,
    PROS_FILE * stream )
```

Reads a string from the specified stream, storing the characters into the memory at str. Characters will be read until the specified limit is reached, a new line is found, or the end of file is reached.

If the stream is already at end of file (for files in Read mode), NULL will be returned; otherwise, at least one character will be read and stored into str.

Parameters

<i>str</i>	the location where the characters read will be stored
<i>num</i>	the maximum number of characters to store; at most (num - 1) characters will be read, with a null terminator ('\0') automatically appended
<i>stream</i>	the channel to read (stdin, uart1, uart2, or an open file in Read mode)

Returns

str, or NULL if zero characters could be read

5.1.4.21 fopen()

```
PROS_FILE* fopen (
    const char * file,
    const char * mode )
```

Opens the given file in the specified mode. The file name is truncated to eight characters. Only four files can be in use simultaneously in any given time, with at most one of those files in Write mode. This function does not work on communication ports; use **uartInit()** (p. 88) instead.

mode can be "r" or "w". Due to the nature of the VEX Cortex memory, the "r+", "w+", and "a" modes are not supported by the file system.

Opening a file that does not exist in Read mode will fail and return NULL, but opening a new file in Write mode will create it if there is space. Opening a file that already exists in Write mode will destroy the contents and create a new blank file if space is available.

There are important considerations when using of the file system on the VEX Cortex. Reading from files is safe, but writing to files should only be performed when robot actuators have been stopped. PROS will attempt to continue to handle events during file writes, but most user tasks cannot execute during file writing. Powering down the VEX Cortex mid-write may cause file system corruption.

Parameters

<i>file</i>	the file name
<i>mode</i>	the file mode

Returns

a file descriptor pointing to the new file, or NULL if the file could not be opened

5.1.4.22 fprintf()

```
void fprintf (
    const char * string,
    PROS_FILE * stream )
```

Prints the simple string to the specified stream.

This method is much, much faster than **fprintf()** (p. 47) and does not add a new line like **fputs()** (p. 49). Do not use **fprintf()** (p. 47) on a VEX LCD port. Use **LcdSetText()** (p. 67) instead.

Parameters

<i>string</i>	the string to write
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

5.1.4.23 fprintf()

```
int fprintf (
    PROS_FILE * stream,
    const char * formatString,
    ... )
```

Prints the formatted string to the specified output stream.

The specifiers supported by this minimalistic **printf()** (p. 75) function are:

- %d: Signed integer in base 10 (int)
- %u: Unsigned integer in base 10 (unsigned int)
- %x, %X: Integer in base 16 (unsigned int, int)
- %p: Pointer (void *, int *, ...)
- %c: Character (char)

- %s : Null-terminated string (char *)
- %%: Single literal percent sign
- %f : Floating-point number

Specifiers can be modified with:

- 0: Zero-pad, instead of space-pad
- a.b: Make the field at least "a" characters wide. If "b" is specified for "%f", changes the number of digits after the decimal point
- -: Left-align, instead of right-align
- +: Always display the sign character (displays a leading "+" for positive numbers)
- l: Ignored for compatibility

Invalid format specifiers, or mismatched parameters to specifiers, cause undefined behavior. Other characters are written out verbatim. Do not use **fprintf()** (p. 47) on a VEX LCD port. Use **lcdPrint()** instead.

Parameters

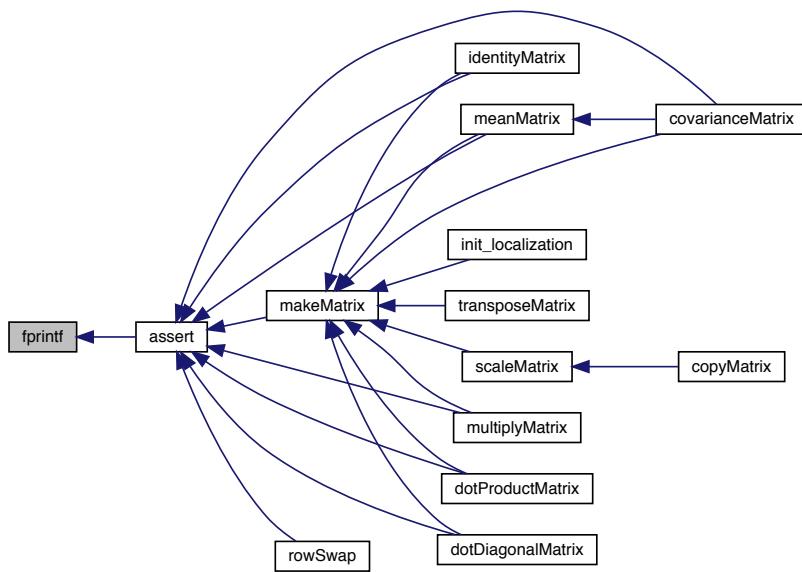
<i>stream</i>	the stream to write (stdout, uart1, or uart2)
<i>formatString</i>	the format string as specified above

Returns

the number of characters written

Referenced by **assert()**.

Here is the caller graph for this function:



5.1.4.24 fputc()

```
int fputc (
    int value,
    PROS_FILE * stream )
```

Writes one character to the specified stream.

Do not use **fputc()** (p. 49) on a VEX LCD port. Use **LcdSetText()** (p. 67) instead.

Parameters

<i>value</i>	the character to write (a value of type "char" can be used)
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

Returns

the character written

5.1.4.25 fputs()

```
int fputs (
    const char * string,
    PROS_FILE * stream )
```

Behaves the same as the "fprint" function, and appends a trailing newline ("\n").

Do not use **fputs()** (p. 49) on a VEX LCD port. Use **LcdSetText()** (p. 67) instead.

Parameters

<i>string</i>	the string to write
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

Returns

the number of characters written, excluding the new line

5.1.4.26 fread()

```
size_t fread (
    void * ptr,
    size_t size,
    size_t count,
    PROS_FILE * stream )
```

Reads data from a stream into memory. Returns the number of bytes thus read.

If the memory at *ptr* cannot store (*size* * *count*) bytes, undefined behavior occurs.

Parameters

<i>ptr</i>	a pointer to where the data will be stored
<i>size</i>	the size of each data element to read in bytes
<i>count</i>	the number of data elements to read
<i>stream</i>	the stream to read (stdout, uart1, uart2, or an open file in Read mode)

Returns

the number of bytes successfully read

5.1.4.27 fseek()

```
int fseek (
    PROS_FILE * stream,
    long int offset,
    int origin )
```

Seeks within a file open in Read mode. This function will fail when used on a file in Write mode or on any communications port.

Parameters

<i>stream</i>	the stream to seek within
<i>offset</i>	the location within the stream to seek
<i>origin</i>	the reference location for offset: SEEK_CUR, SEEK_SET, or SEEK_END

Returns

0 if the seek was successful, or 1 otherwise

5.1.4.28 ftell()

```
long int ftell (
    PROS_FILE * stream )
```

Returns the current position of the stream. This function works on files in either Read or Write mode, but will fail on communications ports.

Parameters

<i>stream</i>	the stream to check
---------------	---------------------

Returns

the offset of the stream, or -1 if the offset could not be determined

5.1.4.29 fwrite()

```
size_t fwrite (
    const void * ptr,
    size_t size,
    size_t count,
    PROS_FILE * stream )
```

Writes data from memory to a stream. Returns the number of bytes thus written.

If the memory at *ptr* is not as long as (*size* * *count*) bytes, undefined behavior occurs.

Parameters

<i>ptr</i>	a pointer to the data to write
<i>size</i>	the size of each data element to write in bytes
<i>count</i>	the number of data elements to write
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

Returns

the number of bytes successfully written

5.1.4.30 getchar()

```
int getchar ( )
```

Reads and returns one character from "stdin", which is the PC debug terminal.

Returns

the next character from 0 to 255, or -1 if no character can be read

5.1.4.31 gyroGet()

```
int gyroGet (
    Gyro gyro )
```

Gets the current gyro angle in degrees, rounded to the nearest degree.

There are 360 degrees in a circle.

Parameters

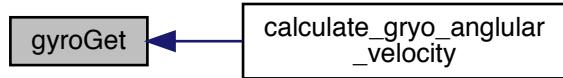
<i>gyro</i>	the Gyro object from gyroInit() (p. 53) to read
-------------	--

Returns

the signed and cumulative number of degrees rotated around the gyro's vertical axis since the last start or reset

Referenced by **calculate_gryo-angular_velocity()**.

Here is the caller graph for this function:



5.1.4.32 gyroInit()

```
Gyro gyroInit (
    unsigned char port,
    unsigned short multiplier )
```

Initializes and enables a gyro on an analog port.

NULL will be returned if the port is invalid or the gyro is already in use. Initializing a gyro implicitly calibrates it and resets its count. Do not move the robot while the gyro is being calibrated. It is suggested to call this function in **initialize()** (p. 174) and to place the robot in its final position before powering it on.

The multiplier parameter can tune the gyro to adapt to specific sensors. The default value at this time is 196; higher values will increase the number of degrees reported for a fixed actual rotation, while lower values will decrease the number of degrees reported. If your robot is consistently turning too far, increase the multiplier, and if it is not turning far enough, decrease the multiplier.

Parameters

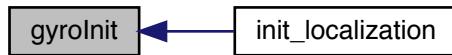
<i>port</i>	the analog port to use from 1-8
<i>multiplier</i>	an optional constant to tune the gyro readings; use 0 for the default value

Returns

a Gyro object to be stored and used for later calls to gyro functions

Referenced by **init_localization()**.

Here is the caller graph for this function:



5.1.4.33 gyroReset()

```
void gyroReset (
    Gyro gyro )
```

Resets the gyro to zero.

It is safe to use this method while a gyro is enabled. It is not necessary to call this method before stopping or starting a gyro.

Parameters

<i>gyro</i>	the Gyro object from gyroInit() (p. 53) to reset
-------------	---

5.1.4.34 gyroShutdown()

```
void gyroShutdown (
    Gyro gyro )
```

Stops and disables the gyro.

Gyros use processing power, so disabling unused gyros increases code performance. The gyro's position will be retained.

Parameters

<i>gyro</i>	the Gyro object from gyroInit() (p. 53) to stop
-------------	--

5.1.4.35 i2cRead()

```
bool i2cRead (
    uint8_t addr,
    uint8_t * data,
    uint16_t count )
```

i2cRead - Reads the specified number of data bytes from the specified 7-bit I2C address. The bytes will be stored at the specified location. Returns true if successful or false if failed. If only some bytes could be read, false is still returned.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Since most I2C devices use an 8-bit register architecture, this method has limited usefulness. Consider i2cReadRegister instead for the vast majority of applications.

5.1.4.36 i2cReadRegister()

```
bool i2cReadRegister (
    uint8_t addr,
    uint8_t reg,
    uint8_t * value,
    uint16_t count )
```

i2cReadRegister - Reads the specified amount of data from the given register address on the specified 7-bit I2C address. Returns true if successful or false if failed. If only some bytes could be read, false is still returned.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Most I2C devices support an auto-increment address feature, so using this method to read more than one byte will usually read a block of sequential registers. Try to merge reads to separate registers into a larger read using this function whenever possible to improve code reliability, even if a few intermediate values need to be thrown away.

5.1.4.37 i2cWrite()

```
bool i2cWrite (
    uint8_t addr,
    uint8_t * data,
    uint16_t count )
```

i2cWrite - Writes the specified number of data bytes to the specified 7-bit I2C address. Returns true if successful or false if failed. If only smoe bytes could be written, false is still returned.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Since most I2C devices use an 8-bit register architecture, this method is mostly useful for setting the register position (most devices remember the last-used address) or writing a sequence of bytes to one register address using an auto-increment feature. In these cases, the first byte written from the data buffer should have the register address to use.

5.1.4.38 i2cWriteRegister()

```
bool i2cWriteRegister (
    uint8_t addr,
    uint8_t reg,
    uint16_t value )
```

i2cWriteRegister - Writes the specified data byte to a register address on the specified 7-bit I2C address. Returns true if successful or false if failed.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Only one byte can be written to each register address using this method. While useful for the vast majority of I2C operations, writing multiple bytes requires the **i2cWrite** method.

5.1.4.39 imeGet()

```
bool imeGet (
    unsigned char address,
    int * value )
```

Gets the current 32-bit count of the specified IME.

Much like the count for a quadrature encoder, the tick count is signed and cumulative. The value reflects total counts since the last reset. Different VEX Motor Encoders have a different number of counts per revolution:

- 240,448 for the 269 IME
- 627,2 for the 393 IME in high torque mode (factory default)
- 392 for the 393 IME in high speed mode

If the IME address is invalid, or the IME has not been reset or initialized, the value stored in **value* is undefined.

Parameters

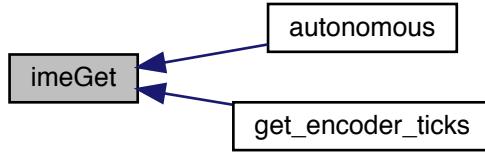
<i>address</i>	the IME address to fetch from 0 to IME_ADDR_MAX
<i>value</i>	a pointer to the location where the value will be stored (obtained using the "&" operator on the target variable name e.g. <code>imeGet(2, &counts)</code>)

Returns

true if the count was successfully read and the value stored in **value* is valid; false otherwise

Referenced by **autonomous()**, and **get_encoder_ticks()**.

Here is the caller graph for this function:



5.1.4.40 `imeGetVelocity()`

```
bool imeGetVelocity (
    unsigned char address,
    int * value )
```

Gets the current rotational velocity of the specified IME.

In this version of PROS, the velocity is positive if the IME count is increasing and negative if the IME count is decreasing. The velocity is in RPM of the internal encoder wheel. Since checking the IME for its type cannot reveal whether the motor gearing is high speed or high torque (in the 2-Wire Motor 393 case), the user must divide the return value by the number of output revolutions per encoder revolution:

- 30.056 for the 269 IME
- 39.2 for the 393 IME in high torque mode (factory default)
- 24.5 for the 393 IME in high speed mode

If the IME address is invalid, or the IME has not been reset or initialized, the value stored in `*value` is undefined.

Parameters

<code>address</code>	the IME address to fetch from 0 to <code>IME_ADDR_MAX</code>
<code>value</code>	a pointer to the location where the value will be stored (obtained using the "&" operator on the target variable name e.g. <code>imeGetVelocity(2, &counts)</code>)

Returns

true if the velocity was successfully read and the value stored in `*value` is valid; false otherwise

Referenced by `get_encoder_velocity()`.

Here is the caller graph for this function:



5.1.4.41 imelInitializeAll()

```
unsigned int imelInitializeAll ( )
```

Initializes all IMEs.

IMEs are assigned sequential incrementing addresses, beginning with the first IME on the chain (closest to the VEX Cortex I2C port). Therefore, a given configuration of IMEs will always have the same ID assigned to each encoder. The addresses range from 0 to IME_ADDR_MAX, so the first encoder gets 0, the second gets 1, ...

This function should most likely be used in **initialize()** (p. 174). Do not use it in **initializeIO()** (p. 175) or at any other time when the scheduler is paused (like an interrupt). Checking the return value of this function is important to ensure that all IMEs are plugged in and responding as expected.

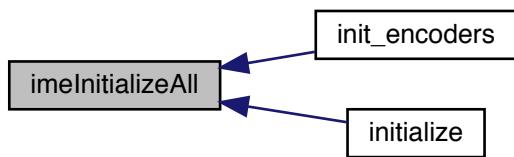
This function, unlike the other IME functions, is not thread safe. If using imelInitializeAll to re-initialize encoders, calls to other IME functions might behave unpredictably during this function's execution.

Returns

the number of IMEs successfully initialized.

Referenced by **init_encoders()**, and **initialize()**.

Here is the caller graph for this function:



5.1.4.42 imeReset()

```
bool imeReset (
    unsigned char address )
```

Resets the specified IME's counters to zero.

This method can be used while the IME is rotating.

Parameters

<code>address</code>	the IME address to reset from 0 to IME_ADDR_MAX
----------------------	---

Returns

true if the reset succeeded; false otherwise

Referenced by `autonomous()`.

Here is the caller graph for this function:



5.1.4.43 imeShutdown()

```
void imeShutdown ( )
```

Shuts down all IMEs on the chain; their addresses return to the default and the stored counts and velocities are lost. This function, unlike the other IME functions, is not thread safe.

To use the IME chain again, wait at least 0.25 seconds before using imelInitializeAll again.

5.1.4.44 ioClearInterrupt()

```
void ioClearInterrupt (
    unsigned char pin )
```

Disables interrupts on the specified pin.

Disabling interrupts on interrupt pins which are not in use conserves processing time.

Parameters

<i>pin</i>	the pin on which to reset interrupts from 1-9,11-12
------------	---

5.1.4.45 ioSetInterrupt()

```
void ioSetInterrupt (
    unsigned char pin,
    unsigned char edges,
    InterruptHandler handler )
```

Sets up an interrupt to occur on the specified pin, and resets any counters or timers associated with the pin.

Each time the specified change occurs, the function pointer passed in will be called with the pin that changed as an argument. Enabling pin-change interrupts consumes processing time, so it is best to only enable necessary interrupts and to keep the InterruptHandler function short. Pin change interrupts can only be enabled on pins 1-9 and 11-12.

Do not use API functions such as **delay()** (p. 39) inside the handler function, as the function will run in an ISR where the scheduler is paused and no other interrupts can execute. It is best to quickly update some state and allow a task to perform the work.

Do not use this function on pins that are also being used by the built-in ultrasonic or shaft encoder drivers, or on pins which have been switched to output mode.

Parameters

<i>pin</i>	the pin on which to enable interrupts from 1-9,11-12
<i>edges</i>	one of INTERRUPT_EDGE_RISING, INTERRUPT_EDGE_FALLING, or INTERRUPT_EDGE_BOTH
<i>handler</i>	the function to call when the condition is satisfied

5.1.4.46 isAutonomous()

```
bool isAutonomous ( )
```

Returns true if the robot is in autonomous mode, or false otherwise.

While in autonomous mode, joystick inputs will return a neutral value, but serial port communications (even over VexNET) will still work properly.

5.1.4.47 isEnabled()

```
bool isEnabled ( )
```

Returns true if the robot is enabled, or false otherwise.

While disabled via the VEX Competition Switch or VEX Field Controller, motors will not function. However, the digital I/O ports can still be changed, which may indirectly affect the robot state (e.g. solenoids). Avoid performing externally visible actions while disabled (the kernel should take care of this most of the time).

Referenced by `display_menu()`.

Here is the caller graph for this function:



5.1.4.48 `isJoystickConnected()`

```
bool isJoystickConnected (
    unsigned char joystick )
```

Returns true if a joystick is connected to the specified slot number (1 or 2), or false otherwise.

Useful for automatically merging joysticks for one operator, or splitting for two. This function does not work properly during `initialize()` (p. 174) or `initializeIO()` (p. 175) and can return false positives. It should be checked once and stored at the beginning of `operatorControl()` (p. 175).

Parameters

<code>joystick</code>	the joystick slot to check
-----------------------	----------------------------

5.1.4.49 `isOnline()`

```
bool isOnline ( )
```

Returns true if a VEX field controller or competition switch is connected, or false otherwise.

When in online mode, the switching between `autonomous()` (p. 172) and `operatorControl()` (p. 175) tasks is managed by the PROS kernel.

5.1.4.50 joystickGetAnalog()

```
int joystickGetAnalog (
    unsigned char joystick,
    unsigned char axis )
```

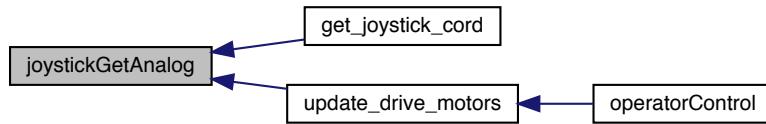
Gets the value of a control axis on the VEX joystick. Returns the value from -127 to 127, or 0 if no joystick is connected to the requested slot.

Parameters

<i>joystick</i>	the joystick slot to check
<i>axis</i>	one of 1, 2, 3, 4, ACCEL_X, or ACCEL_Y

Referenced by `get_joystick_cord()`, and `update_drive_motors()`.

Here is the caller graph for this function:

**5.1.4.51 joystickGetDigital()**

```
bool joystickGetDigital (
    unsigned char joystick,
    unsigned char buttonGroup,
    unsigned char button )
```

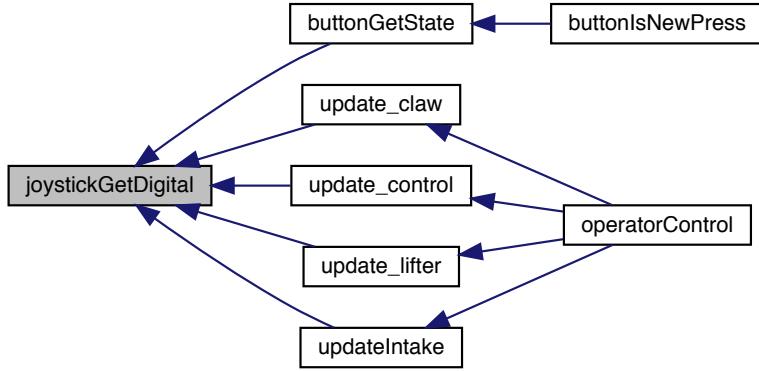
Gets the value of a button on the VEX joystick. Returns true if that button is pressed, or false otherwise. If no joystick is connected to the requested slot, returns false.

Parameters

<i>joystick</i>	the joystick slot to check
<i>buttonGroup</i>	one of 5, 6, 7, or 8 to request that button as labelled on the joystick
<i>button</i>	one of JOY_UP, JOY_DOWN, JOY_LEFT, or JOY_RIGHT; requesting JOY_LEFT or JOY_RIGHT for groups 5 or 6 will cause an undefined value to be returned

Referenced by `buttonGetState()`, `update_claw()`, `update_control()`, `update_lifter()`, and `updateIntake()`.

Here is the caller graph for this function:



5.1.4.52 lcdClear()

```
void lcdClear (
    PROS_FILE * lcdPort )
```

Clears the LCD screen on the specified port.

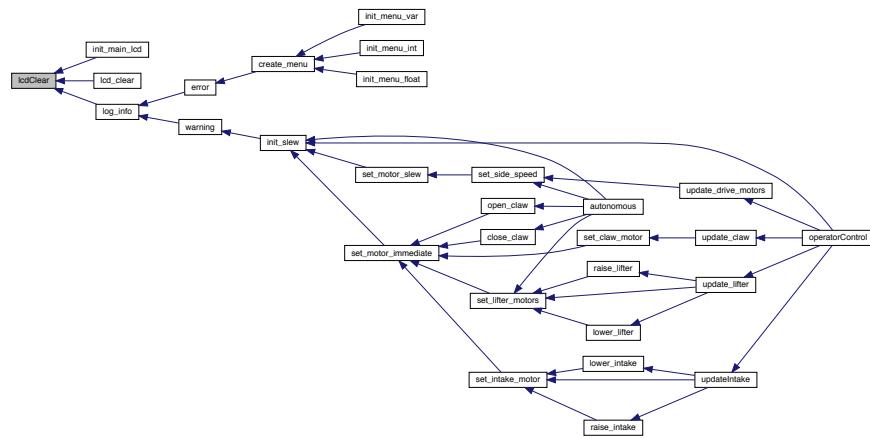
Printing to a line implicitly overwrites the contents, so clearing should only be required at startup.

Parameters

<i>lcdPort</i>	the LCD to clear, either uart1 or uart2
----------------	---

Referenced by `init_main_lcd()`, `lcd_clear()`, and `log_info()`.

Here is the caller graph for this function:



5.1.4.53 lcdInit()

```
void lcdInit (
    PROS_FILE * lcdPort )
```

Initializes the LCD port, but does not change the text or settings.

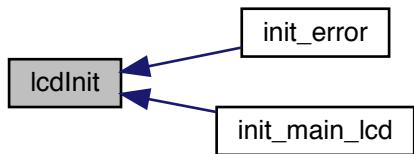
If the LCD was not initialized before, the text currently on the screen will be undefined. The port will not be usable with standard serial port functions until the LCD is stopped.

Parameters

<i>lcdPort</i>	the LCD to initialize, either uart1 or uart2
----------------	--

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

Here is the caller graph for this function:



5.1.4.54 lcdReadButtons()

```
void unsigned char const char unsigned int lcdReadButtons (
    PROS_FILE * lcdPort )
```

Reads the user button status from the LCD display.

For example, if the left and right buttons are pushed, $(1 | 4) = 5$ will be returned. 0 is returned if no buttons are pushed.

Parameters

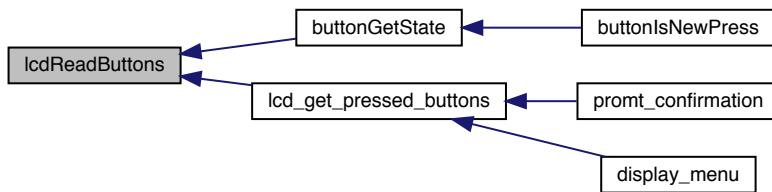
<i>lcdPort</i>	the LCD to poll, either uart1 or uart2
----------------	--

Returns

the buttons pressed as a bit mask

Referenced by **buttonGetState()**, and **lcd_get_pressed_buttons()**.

Here is the caller graph for this function:



5.1.4.55 lcdSetBacklight()

```
void lcdSetBacklight (
    PROS_FILE * lcdPort,
    bool backlight )
```

Sets the specified LCD backlight to be on or off.

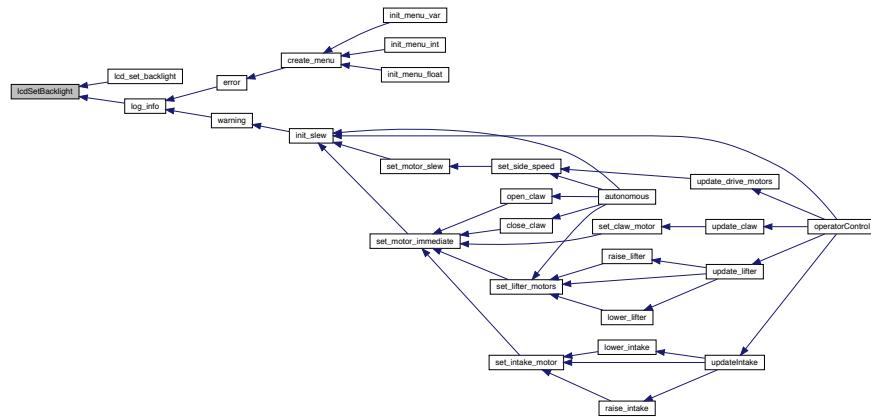
Turning it off will save power but may make it more difficult to read in dim conditions.

Parameters

<i>lcdPort</i>	the LCD to adjust, either uart1 or uart2
<i>backlight</i>	true to turn the backlight on, or false to turn it off

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

Here is the caller graph for this function:

**5.1.4.56 lcdSetText()**

```
void lcdSetText (
    PROS_FILE * lcdPort,
    unsigned char line,
    const char * buffer )
```

Prints the string buffer to the attached LCD.

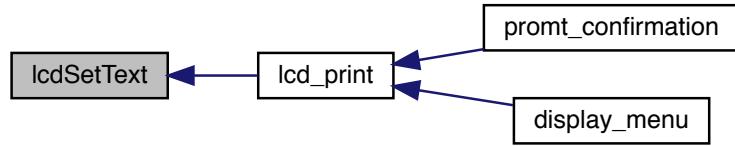
The output string will be truncated as necessary to fit on the LCD screen, 16 characters wide. This function, like `fprint()` (p. 47), is much, much faster than a formatted routine such as `lcdPrint()` and consumes less memory.

Parameters

<i>lcdPort</i>	the LCD to write, either uart1 or uart2
<i>line</i>	the LCD line to write, either 1 or 2
<i>buffer</i>	the string to write

Referenced by `Lcd_print()`.

Here is the caller graph for this function:



5.1.4.57 `lcdShutdown()`

```
void lcdShutdown (
    PROS_FILE * lcdPort )
```

Shut down the specified LCD port.

Parameters

<code>lcdPort</code>	the LCD to stop, either uart1 or uart2
----------------------	--

5.1.4.58 `micros()`

```
unsigned long micros ( )
```

Returns the number of microseconds since Cortex power-up. There are 10^6 microseconds in a second, so as a 32-bit integer, this will overflow and wrap back to zero every two hours or so.

This function is Wiring-compatible.

Returns

the number of microseconds since the Cortex was turned on or the last overflow

5.1.4.59 millis()

```
unsigned long millis ( )
```

Returns the number of milliseconds since Cortex power-up. There are 1000 milliseconds in a second, so as a 32-bit integer, this will not overflow for 50 days.

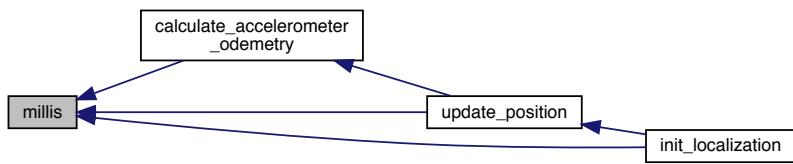
This function is Wiring-compatible.

Returns

the number of milliseconds since the Cortex was turned on

Referenced by [calculate_accelerometer_odometry\(\)](#), [init_localization\(\)](#), and [update_position\(\)](#).

Here is the caller graph for this function:



5.1.4.60 motorGet()

```
int motorGet (
    unsigned char channel )
```

Gets the last set speed of the specified motor channel.

This speed may have been set by any task or the PROS kernel itself. This is not guaranteed to be the speed that the motor is actually running at, or even the speed currently being sent to the motor, due to latency in the Motor Controller 29 protocol and physical loading. To measure actual motor shaft revolution speed, attach a VEX Integrated Motor Encoder or VEX Quadrature Encoder and use the velocity functions associated with each.

Parameters

<i>channel</i>	the motor channel to fetch from 1-10
----------------	--------------------------------------

Returns

the speed last sent to this channel; -127 is full reverse and 127 is full forward, with 0 being off

5.1.4.61 motorSet()

```
void motorSet (
    unsigned char channel,
    int speed )
```

Sets the speed of the specified motor channel.

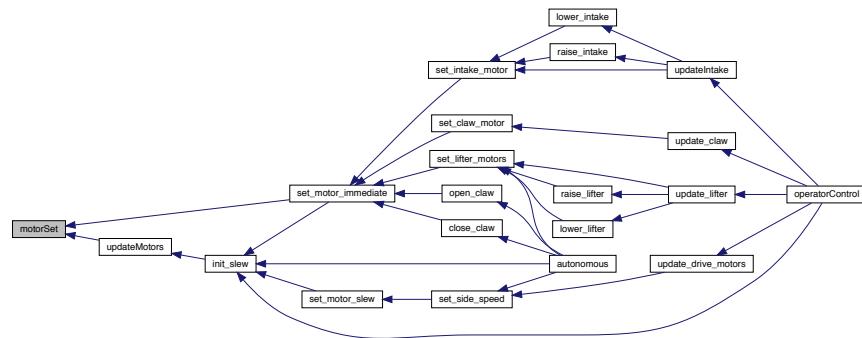
Do not use **motorSet()** (p. 70) with the same channel argument from two different tasks. It is safe to use **motorSet()** (p. 70) with different channel arguments from different tasks.

Parameters

<i>channel</i>	the motor channel to modify from 1-10
<i>speed</i>	the new signed speed; -127 is full reverse and 127 is full forward, with 0 being off

Referenced by **set_motor_immediate()**, and **updateMotors()**.

Here is the caller graph for this function:



5.1.4.62 motorStop()

```
void motorStop (
    unsigned char channel )
```

Stops the motor on the specified channel, equivalent to calling **motorSet()** (p. 70) with an argument of zero.

This performs a coasting stop, not an active brake. Since **motorStop** is similar to **motorSet(0)**, see the note for **motorSet()** (p. 70) about use from multiple tasks.

Parameters

<code>channel</code>	the motor channel to stop from 1-10
----------------------	-------------------------------------

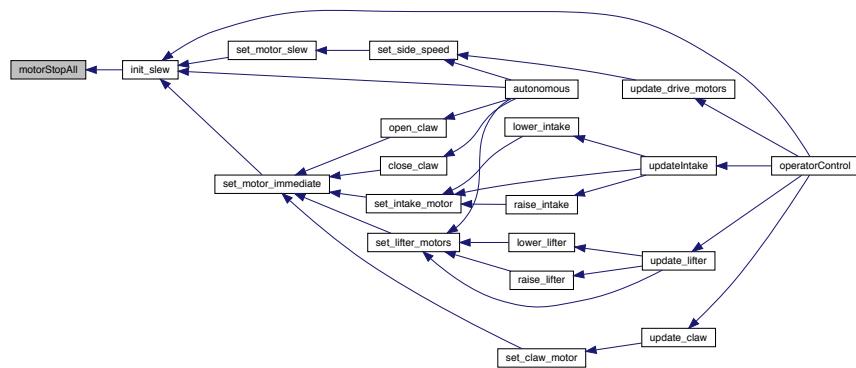
5.1.4.63 motorStopAll()

```
void motorStopAll( )
```

Stops all motors; significantly faster than looping through all motor ports and calling motorSet(channel, 0) on each one.

Referenced by `init_slew()`.

Here is the caller graph for this function:



5.1.4.64 mutexCreate()

```
Mutex mutexCreate( )
```

Creates a mutex intended to allow only one task to use a resource at a time. For signalling and synchronization, try using semaphores.

Mutexes created using this function can be accessed using the `mutexTake()` (p. 73) and `mutexGive()` (p. 72) functions. The semaphore functions must not be used on objects of this type.

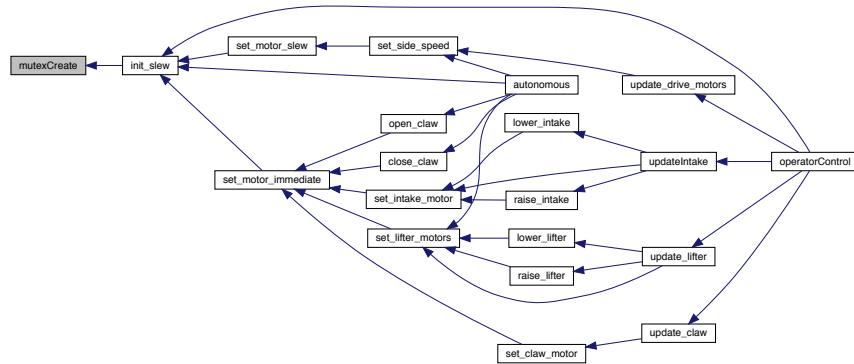
This type of object uses a priority inheritance mechanism so a task 'taking' a mutex MUST ALWAYS 'give' the mutex back once the mutex is no longer required.

Returns

a handle to the created mutex

Referenced by **init_slew()**.

Here is the caller graph for this function:

**5.1.4.65 mutexDelete()**

```
void mutexDelete (
    Mutex mutex )
```

Deletes the specified mutex. This function can be dangerous; deleting semaphores being waited on by a task may cause deadlock or a crash.

Parameters

<code>mutex</code>	the mutex to destroy
--------------------	----------------------

5.1.4.66 mutexGive()

```
bool mutexGive (
    Mutex mutex )
```

Relinquishes a mutex so that other tasks can use the resource it guards. The mutex must be held by the current task using a corresponding call to `mutexTake`.

Parameters

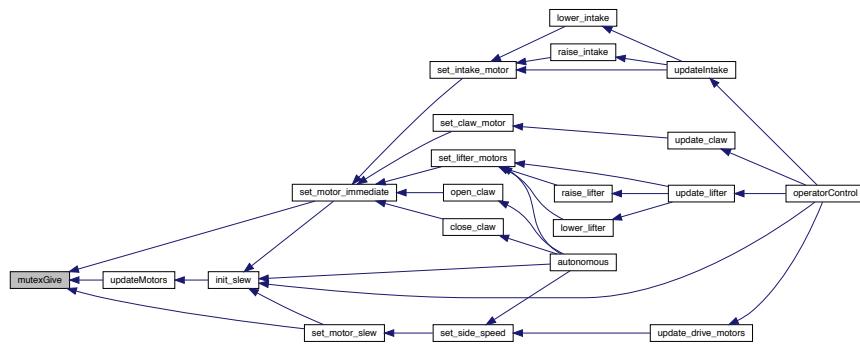
<code>mutex</code>	the mutex to release
--------------------	----------------------

Returns

true if the mutex was released, or false if the mutex was not already held

Referenced by `set_motor_immediate()`, `set_motor_slew()`, and `updateMotors()`.

Here is the caller graph for this function:



5.1.4.67 mutexTake()

```
bool mutexTake (
    Mutex mutex,
    const unsigned long blockTime )
```

Requests a mutex so that other tasks cannot simultaneously use the resource it guards. The mutex must not already be held by the current task. If another task already holds the mutex, the function will wait for the mutex to be released. Other tasks can run during this time.

Parameters

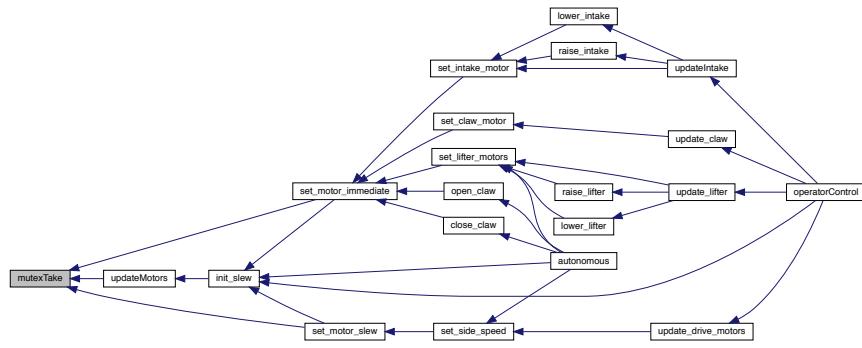
<code>mutex</code>	the mutex to request
<code>blockTime</code>	the maximum time to wait for the mutex to be available, where -1 specifies an infinite timeout

Returns

true if the mutex was successfully taken, or false if the timeout expired

Referenced by `set_motor_immediate()`, `set_motor_slew()`, and `updateMotors()`.

Here is the caller graph for this function:



5.1.4.68 pinMode()

```
void pinMode (
    unsigned char pin,
    unsigned char mode )
```

Configures the pin as an input or output with a variety of settings.

Do note that INPUT by default turns on the pull-up resistor, as most VEX sensors are open-drain active low. It should not be a big deal for most push-pull sources. This function is Wiring-compatible.

Parameters

<i>pin</i>	the pin to modify from 1-26
<i>mode</i>	one of INPUT, INPUT_ANALOG, INPUT_FLOATING, OUTPUT, or OUTPUT_OD

5.1.4.69 powerLevelBackup()

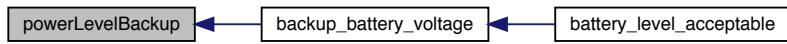
```
unsigned int powerLevelBackup ( )
```

Returns the backup battery voltage in millivolts.

If no backup battery is connected, returns 0.

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

Here is the caller graph for this function:



5.1.4.70 powerLevelMain()

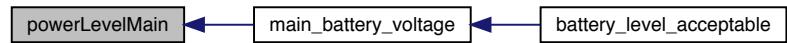
```
unsigned int powerLevelMain ( )
```

Returns the main battery voltage in millivolts.

In rare circumstances, this method might return 0. Check the output value for reasonability before blindly blasting the user.

Referenced by **main_battery_voltage()**.

Here is the caller graph for this function:



5.1.4.71 print()

```
void print (
    const char * string )
```

Prints the simple string to the debug terminal without formatting.

This method is much, much faster than **printf()** (p. 75).

Parameters

<i>string</i>	the string to write
---------------	---------------------

5.1.4.72 printf()

```
int printf (
    const char * formatString,
    ...
)
```

Prints the formatted string to the debug stream (the PC terminal).

Parameters

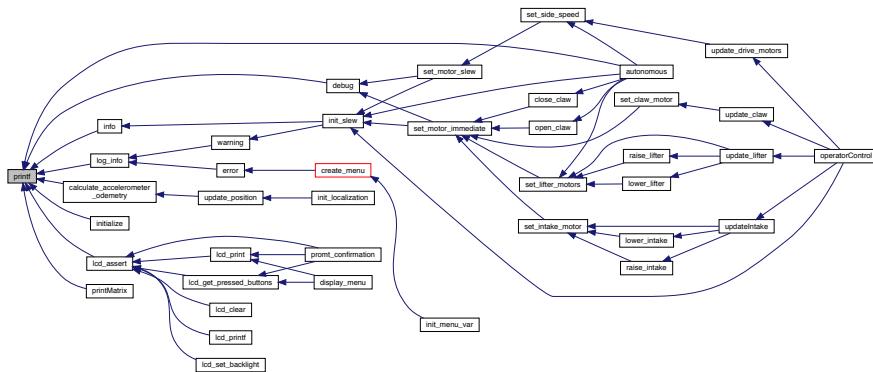
<i>formatString</i>	the format string as specified in fprintf() (p. 47)
---------------------	--

Returns

the number of characters written

Referenced by **autonomous()**, **calculate_accelerometer_odometry()**, **debug()**, **info()**, **initialize()**, **Lcd_assert()**, **log_info()**, and **printMatrix()**.

Here is the caller graph for this function:



5.1.4.73 putchar()

```
int putchar (
    int value )
```

Writes one character to "stdout", which is the PC debug terminal, and returns the input value.

When using a wireless connection, one may need to press the spacebar before the input is visible on the terminal.

Parameters

<i>value</i>	the character to write (a value of type "char" can be used)
--------------	---

Returns

the character written

5.1.4.74 puts()

```
int puts (
    const char * string )
```

Behaves the same as the "print" function, and appends a trailing newline ("\n").

Parameters

<i>string</i>	the string to write
---------------	---------------------

Returns

the number of characters written, excluding the new line

5.1.4.75 semaphoreCreate()

```
Semaphore semaphoreCreate ( )
```

Creates a semaphore intended for synchronizing tasks. To prevent some critical code from simultaneously modifying a shared resource, use mutexes instead.

Semaphores created using this function can be accessed using the **semaphoreTake()** (p. 78) and **semaphoreGive()** (p. 78) functions. The mutex functions must not be used on objects of this type.

This type of object does not need to have balanced take and give calls, so priority inheritance is not used. Semaphores can be signalled by an interrupt routine.

Returns

a handle to the created semaphore

5.1.4.76 semaphoreDelete()

```
void semaphoreDelete (
    Semaphore semaphore )
```

Deletes the specified semaphore. This function can be dangerous; deleting semaphores being waited on by a task may cause deadlock or a crash.

Parameters

<code>semaphore</code>	the semaphore to destroy
------------------------	--------------------------

5.1.4.77 `semaphoreGive()`

```
bool semaphoreGive (
    Semaphore semaphore )
```

Signals a semaphore. Tasks waiting for a signal using **semaphoreTake()** (p. 78) will be unblocked by this call and can continue execution.

Slow processes can give semaphores when ready, and fast processes waiting to take the semaphore will continue at that point.

Parameters

<code>semaphore</code>	the semaphore to signal
------------------------	-------------------------

Returns

true if the semaphore was successfully given, or false if the semaphore was not taken since the last give

5.1.4.78 `semaphoreTake()`

```
bool semaphoreTake (
    Semaphore semaphore,
    const unsigned long blockTime )
```

Waits on a semaphore. If the semaphore is already in the "taken" state, the current task will wait for the semaphore to be signaled. Other tasks can run during this time.

Parameters

<code>semaphore</code>	the semaphore to wait
<code>blockTime</code>	the maximum time to wait for the semaphore to be given, where -1 specifies an infinite timeout

Returns

true if the semaphore was successfully taken, or false if the timeout expired

5.1.4.79 setTeamName()

```
void setTeamName (
    const char * name )
```

Sets the team name displayed to the VEX field control and VEX Firmware Upgrade.

Parameters

<i>name</i>	a string containing the team name; only the first eight characters will be shown
-------------	--

Referenced by **initialize()**.

Here is the caller graph for this function:



5.1.4.80 snprintf()

```
int snprintf (
    char * buffer,
    size_t limit,
    const char * formatString,
    ... )
```

Prints the formatted string to the string buffer with the specified length limit.

The length limit, as per the C standard, includes the trailing null character, so an argument of 256 will cause a maximum of 255 non-null characters to be printed, and one null terminator in all cases.

Parameters

<i>buffer</i>	the string buffer where characters can be placed
<i>limit</i>	the maximum number of characters to write
<i>formatString</i>	the format string as specified in fprintf() (p. 47)

Returns

the number of characters stored

5.1.4.81 speakerInit()

```
void speakerInit ( )
```

Initializes VEX speaker support.

The VEX speaker is not thread safe; it can only be used from one task at a time. Using the VEX speaker may impact robot performance. Teams may benefit from an if statement that only enables sound if **isOnline()** (p. 61) returns false.

5.1.4.82 speakerPlayArray()

```
void speakerPlayArray (
    const char ** songs )
```

Plays up to three RTTTL (Ring Tone Text Transfer Language) songs simultaneously over the VEX speaker. The audio is mixed to allow polyphonic sound to be played. Many simple songs are available in RTTTL format online, or compose your own.

The song must not be NULL, but unused tracks within the song can be set to NULL. If any of the three song tracks is invalid, the result of this function is undefined.

The VEX speaker is not thread safe; it can only be used from one task at a time. Using the VEX speaker may impact robot performance. Teams may benefit from an if statement that only enables sound if **isOnline()** (p. 61) returns false.

Parameters

<i>songs</i>	an array of up to three (3) RTTTL songs as string values to play
--------------	--

5.1.4.83 speakerPlayRtttl()

```
void speakerPlayRtttl (
    const char * song )
```

Plays an RTTTL (Ring Tone Text Transfer Language) song over the VEX speaker. Many simple songs are available in RTTTL format online, or compose your own.

The song must not be NULL. If an invalid song is specified, the result of this function is undefined.

The VEX speaker is not thread safe; it can only be used from one task at a time. Using the VEX speaker may impact robot performance. Teams may benefit from an if statement that only enables sound if **isOnline()** (p. 61) returns false.

Parameters

<i>song</i>	the RTTTL song as a string value to play
-------------	--

5.1.4.84 speakerShutdown()

```
void speakerShutdown ( )
```

Powers down and disables the VEX speaker.

If a song is currently being played in another task, the behavior of this function is undefined, since the VEX speaker is not thread safe.

5.1.4.85 sprintf()

```
int sprintf (
    char * buffer,
    const char * formatString,
    ... )
```

Prints the formatted string to the string buffer.

If the buffer is not big enough to contain the complete formatted output, undefined behavior occurs. See **sprintf()** (p. 79) for a safer version of this function.

Parameters

<i>buffer</i>	the string buffer where characters can be placed
<i>formatString</i>	the format string as specified in fprintf() (p. 47)

Returns

the number of characters stored

5.1.4.86 standaloneModeEnable()

```
void standaloneModeEnable ( )
```

Enables the Cortex to run the op control task in a standalone mode- no VEXnet connection required.

This function should only be called once in **initializeIO()** (p. 175)

5.1.4.87 taskCreate()

```
TaskHandle taskCreate (
    TaskCode taskCode,
    const unsigned int stackDepth,
    void * parameters,
    const unsigned int priority )
```

Creates a new task and add it to the list of tasks that are ready to run.

Parameters

<i>taskCode</i>	the function to execute in its own task
<i>stackDepth</i>	the number of variables available on the stack (4 * stackDepth bytes will be allocated on the Cortex)
<i>parameters</i>	an argument passed to the taskCode function
<i>priority</i>	a value from TASK_PRIORITY_LOWEST to TASK_PRIORITY_HIGHEST determining the initial priority of the task

Returns

a handle to the created task, or NULL if an error occurred

5.1.4.88 taskDelay()

```
void taskDelay (
    const unsigned long msToDelay )
```

Delays the current task for a given number of milliseconds.

Delaying for a period of zero will force a reschedule, where tasks of equal priority may be scheduled if available. The calling task will still be available for immediate rescheduling once the other tasks have had their turn or if nothing of equal or higher priority is available to be scheduled.

This is not the best method to have a task execute code at predefined intervals, as the delay time is measured from when the delay is requested. To delay cyclically, use **taskDelayUntil()** (p. 82).

Parameters

<i>msToDelay</i>	the number of milliseconds to wait, with 1000 milliseconds per second
------------------	---

5.1.4.89 taskDelayUntil()

```
void taskDelayUntil (
    unsigned long * previousWakeTime,
    const unsigned long cycleTime )
```

Delays the current task until a specified time. The task will be unblocked at the time `*previousWakeTime + cycleTime`, and `*previousWakeTime` will be changed to reflect the time at which the task will unblock.

If the target time is in the past, no delay occurs, but a reschedule is forced, as if `taskDelay()` (p. 82) was called with an argument of zero. If the sum of `cycleTime` and `*previousWakeTime` overflows or underflows, undefined behavior occurs.

This function should be used by cyclical tasks to ensure a constant execution frequency. While `taskDelay()` (p. 82) specifies a wake time relative to the time at which the function is called, `taskDelayUntil()` (p. 82) specifies the absolute future time at which it wishes to unblock. Calling `taskDelayUntil` with the same `cycleTime` parameter value in a loop, with `previousWakeTime` referring to a local variable initialized to `millis()` (p. 68), will cause the loop to execute with a fixed period.

Parameters

<code>previousWakeTime</code>	a pointer to the location storing the last unblock time, obtained by using the "&" operator on a variable (e.g. "taskDelayUntil(&now, 50);")
<code>cycleTime</code>	the number of milliseconds to wait, with 1000 milliseconds per second

5.1.4.90 taskDelete()

```
void taskDelete (
    TaskHandle taskToDelete )
```

Kills and removes the specified task from the kernel task list.

Deleting the last task will end the program, possibly leading to undesirable states as some outputs may remain in their last set configuration.

NOTE: The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of processing time. Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

Parameters

<code>taskToDelete</code>	the task to kill; passing NULL kills the current task
---------------------------	---

Referenced by `deinitSlew()`.

Here is the caller graph for this function:



5.1.4.91 taskGetCount()

```
unsigned int taskGetCount ( )
```

Determines the number of tasks that are currently being managed.

This includes all ready, blocked and suspended tasks. A task that has been deleted but not yet freed by the idle task will also be included in the count. Tasks recently created may take one context switch to be counted.

Returns

the number of tasks that are currently running, waiting, or suspended

5.1.4.92 taskGetState()

```
unsigned int taskGetState (
    TaskHandle task )
```

Retrieves the state of the specified task. Note that the state of tasks which have died may be re-used for future tasks, causing the value returned by this function to reflect a different task than possibly intended in this case.

Parameters

<i>task</i>	Handle to the task to query. Passing NULL will query the current task status (which will, by definition, be TASK_RUNNING if this call returns)
-------------	--

Returns

A value reflecting the task's status, one of the constants TASK_DEAD, TASK_RUNNING, TASK_RUNNABLE, TASK_SLEEPING, or TASK_SUSPENDED

5.1.4.93 taskPriorityGet()

```
unsigned int taskPriorityGet (
    const TaskHandle task )
```

Obtains the priority of the specified task.

Parameters

<i>task</i>	the task to check; passing NULL checks the current task
-------------	---

Returns

the priority of that task from 0 to TASK_MAX_PRIORITIES

5.1.4.94 taskPrioritySet()

```
void taskPrioritySet (
    TaskHandle task,
    const unsigned int newPriority )
```

Sets the priority of the specified task.

A context switch may occur before the function returns if the priority being set is higher than the currently executing task and the task being mutated is available to be scheduled.

Parameters

<code>task</code>	the task to change; passing NULL changes the current task
<code>newPriority</code>	a value between TASK_PRIORITY_LOWEST and TASK_PRIORITY_HIGHEST inclusive indicating the new task priority

5.1.4.95 taskResume()

```
void taskResume (
    TaskHandle taskToResume )
```

Resumes the specified task.

A task that has been suspended by one or more calls to **taskSuspend()** (p. 86) will be made available for scheduling again by a call to **taskResume()** (p. 85). If the task was not suspended at the time of the call to **taskResume()** (p. 85), undefined behavior occurs.

Parameters

<code>taskToResume</code>	the task to change; passing NULL is not allowed as the current task cannot be suspended (it is obviously running if this function is called)
---------------------------	--

5.1.4.96 taskRunLoop()

```
TaskHandle taskRunLoop (
    void(*)(void) fn,
    const unsigned long increment )
```

Starts a task which will periodically call the specified function.

Intended for use as a quick-start skeleton for cyclic tasks with higher priority than the "main" tasks. The created task will have priority `TASK_PRIORITY_DEFAULT + 1` with the default stack size. To customize behavior, create a task manually with the specified function.

This task will automatically terminate after one further function invocation when the robot is disabled or when the robot mode is switched.

Parameters

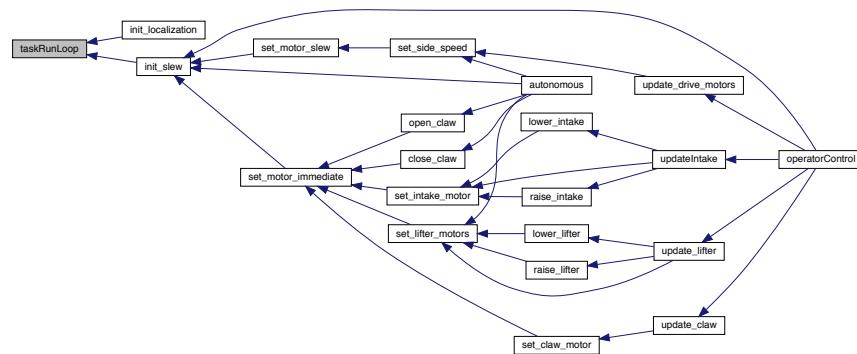
<i>fn</i>	the function to call in this loop
<i>increment</i>	the delay between successive calls in milliseconds; the <code>taskDelayUntil()</code> (p. 82) function is used for accurate cycle timing

Returns

a handle to the task, or `NULL` if an error occurred

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

Here is the caller graph for this function:



5.1.4.97 taskSuspend()

```
void taskSuspend (
    TaskHandle taskToSuspend )
```

Suspends the specified task.

When suspended a task will not be scheduled, regardless of whether it might be otherwise available to run.

Parameters

<i>taskToSuspend</i>	the task to suspend; passing NULL suspends the current task
----------------------	---

5.1.4.98 ultrasonicGet()

```
int ultrasonicGet (
    Ultrasonic ult )
```

Gets the current ultrasonic sensor value in centimeters.

If no object was found or if the ultrasonic sensor is polled while it is pinging and waiting for a response, -1 (ULTRA_B↔AD_RESPONSE) is returned. If the ultrasonic sensor was never started, the return value is undefined. Round and fluffy objects can cause inaccurate values to be returned.

Parameters

<i>ult</i>	the Ultrasonic object from ultrasonicInit() (p. 87) to read
------------	--

Returns

the distance to the nearest object in centimeters

5.1.4.99 ultrasonicInit()

```
Ultrasonic ultrasonicInit (
    unsigned char portEcho,
    unsigned char portPing )
```

Initializes an ultrasonic sensor on the specified digital ports.

The ultrasonic sensor will be polled in the background in concert with the other sensors registered using this method. NULL will be returned if either port is invalid or the ultrasonic sensor port is already in use.

Parameters

<i>portEcho</i>	the port connected to the orange cable from 1-9,11-12
<i>portPing</i>	the port connected to the yellow cable from 1-12

Returns

an Ultrasonic object to be stored and used for later calls to ultrasonic functions

5.1.4.100 ultrasonicShutdown()

```
void ultrasonicShutdown (
    Ultrasonic ult )
```

Stops and disables the ultrasonic sensor.

The last distance it had before stopping will be retained. One more ping operation may occur before the sensor is fully disabled.

Parameters

<i>ult</i>	the Ultrasonic object from ultrasonicInit() (p. 87) to stop
------------	--

5.1.4.101 usartInit()

```
void usartInit (
    PROS_FILE * usart,
    unsigned int baud,
    unsigned int flags )
```

Initialize the specified serial interface with the given connection parameters.

I/O to the port is accomplished using the "standard" I/O functions such as **fputs()** (p. 49), **fprintf()** (p. 47), and **fputc()** (p. 49).

Re-initializing an open port may cause loss of data in the buffers. This routine may be safely called from **initializeIO()** (p. 175) or when the scheduler is paused. If I/O is attempted on a serial port which has never been opened, the behavior will be the same as if the port had been disabled.

Parameters

<i>usart</i>	the port to open, either "uart1" or "uart2"
<i>baud</i>	the baud rate to use from 2400 to 1000000 baud
<i>flags</i>	a bit mask combination of the SERIAL_* flags specifying parity, stop, and data bits

5.1.4.102 usartShutdown()

```
void usartShutdown (
    PROS_FILE * usart )
```

Disables the specified USART interface.

Any data in the transmit and receive buffers will be lost. Attempts to read from the port when it is disabled will deadlock, and attempts to write to it may deadlock depending on the state of the buffer.

Parameters

<i>usart</i>	the port to close, either "uart1" or "uart2"
--------------	--

5.1.4.103 wait()

```
void wait (
    const unsigned long time )
```

Alias of **taskDelay()** (p. 82) intended to help EasyC users.

Parameters

<i>time</i>	the duration of the delay in milliseconds (1 000 milliseconds per second)
-------------	---

5.1.4.104 waitUntil()

```
void waitUntil (
    unsigned long * previousWakeTime,
    const unsigned long time )
```

Alias of **taskDelayUntil()** (p. 82) intended to help EasyC users.

Parameters

<i>previousWakeTime</i>	a pointer to the last wakeup time
<i>time</i>	the duration of the delay in milliseconds (1 000 milliseconds per second)

5.1.4.105 watchdogInit()

```
void watchdogInit ( )
```

Enables IWDG watchdog timer which will reset the cortex if it locks up due to static shock or a misbehaving task preventing the timer to be reset. Not recovering from static shock will cause the robot to continue moving its motors indefinitely until turned off manually.

This function should only be called once in **initializeIO()** (p. 175)

Referenced by **initializeIO()**.

Here is the caller graph for this function:



5.1.5 Variable Documentation

5.1.5.1 formatString

```
void unsigned char const char* formatString
```

Definition at line **1182** of file **API.h**.

5.1.5.2 line

```
void unsigned char line
```

Definition at line **1182** of file **API.h**.

5.2 API.h

```
00001
00021 #ifndef API_H_
00022 #define API_H_
00023
00024 // System includes
00025 #include <stdlib.h>
00026 #include <stdbool.h>
00027 #include <stdarg.h>
00028 #include <stdint.h>
00029
00030 // Begin C++ extern to C
00031 #ifdef __cplusplus
00032 extern "C" {
00033 #endif
00034
00035 // ----- VEX competition functions -----
00036
00040 #define JOY_DOWN 1
00041
00044 #define JOY_LEFT 2
00045
00048 #define JOY_UP 4
00049
00052 #define JOY_RIGHT 8
00053
00056 #define ACCEL_X 5
```

```
00057
00060 #define ACCEL_Y 6
00061
00068 bool isAutonomous();
00077 bool isEnabled();
00088 bool isJoystickConnected(unsigned char joystick);
00096 bool isOnline();
00104 int joystickGetAnalog(unsigned char joystick, unsigned char axis);
00114 bool joystickGetDigital(unsigned char joystick, unsigned char buttonGroup,
00115     unsigned char button);
00121 unsigned int powerLevelBackup();
00128 unsigned int powerLevelMain();
00134 void setTeamName(const char *name);
00135
00136 // ----- Pin control functions -----
00137
00141 #define BOARD_NR_ADC_PINS 8
00142
00151 #define BOARD_NR_GPIO_PINS 27
00152
00157 #define HIGH 1
00158
00163 #define LOW 0
00164
00172 #define INPUT 0x0A
00173
00179 #define INPUT_ANALOG 0x00
00180
00186 #define INPUT_FLOATING 0x04
00187
00193 #define OUTPUT 0x01
00194
00200 #define OUTPUT_OD 0x05
00201
00221 int analogCalibrate(unsigned char channel);
00237 int analogRead(unsigned char channel);
00252 int analogReadCalibrated(unsigned char channel);
00271 int analogReadCalibratedHR(unsigned char channel);
00287 bool digitalRead(unsigned char pin);
00298 void digitalWrite(unsigned char pin, bool value);
00309 void pinMode(unsigned char pin, unsigned char mode);
00310
00311 /*
00312 * Digital port 10 cannot be used as an interrupt port, or for an encoder. Plan accordingly.
00313 */
00314
00318 #define INTERRUPT_EDGE_RISING 1
00319
00322 #define INTERRUPT_EDGE_FALLING 2
00323
00327 #define INTERRUPT_EDGE_BOTH 3
00328
00332 typedef void (*InterruptHandler)(unsigned char pin);
00333
00341 void ioClearInterrupt(unsigned char pin);
00362 void ioSetInterrupt(unsigned char pin, unsigned char edges, InterruptHandler handler);
00363
00364 // ----- Physical output control functions -----
00365
00379 int motorGet(unsigned char channel);
00390 void motorSet(unsigned char channel, int speed);
00400 void motorStop(unsigned char channel);
00405 void motorStopAll();
00406
00414 void speakerInit();
00429 void speakerPlayArray(const char ** songs);
00443 void speakerPlayRttl(const char *song);
00450 void speakerShutdown();
00451
00452 // ----- VEX sensor control functions -----
00453
00458 #define IME_ADDR_MAX 0x1F
00459
00479 unsigned int imeInitializeAll();
00500 bool imeGet(unsigned char address, int *value);
00523 bool imeGetVelocity(unsigned char address, int *value);
00532 bool imeReset(unsigned char address);
00540 void imeShutdown();
00541
00548 typedef void * Gyro;
00549
```

```
00559 int gyroGet(Gyro gyro);
00579 Gyro gyroInit(unsigned char port, unsigned short multiplier);
00588 void gyroReset(Gyro gyro);
00597 void gyroShutdown(Gyro gyro);
00598
00605 typedef void * Encoder;
00614 int encoderGet(Encoder enc);
00627 Encoder encoderInit(unsigned char portTop, unsigned char portBottom, bool
    reverse);
00636 void encoderReset(Encoder enc);
00645 void encoderShutdown(Encoder enc);
00646
00650 #define ULTRA_BAD_RESPONSE -1
00651
00658 typedef void * Ultrasonic;
00670 int ultrasonicGet(Ultrasonic ult);
00682 Ultrasonic ultrasonicInit(unsigned char portEcho, unsigned char portPing);
00691 void ultrasonicShutdown(Ultrasonic ult);
00692
00693 // ----- Custom sensor control functions -----
00694
00695 // ---- I2C port control ----
00706 bool i2cRead(uint8_t addr, uint8_t *data, uint16_t count);
00719 bool i2cReadRegister(uint8_t addr, uint8_t reg, uint8_t *value, uint16_t count);
00732 bool i2cWrite(uint8_t addr, uint8_t *data, uint16_t count);
00742 bool i2cWriteRegister(uint8_t addr, uint8_t reg, uint16_t value);
00743
00750 typedef int PROS_FILE;
00751
00752
00753 #ifndef FILE
00754
00759 #define FILE PROS_FILE
00760 #endif
00761
00765 #define SERIAL_DATABITS_8 0x0000
00766
00769 #define SERIAL_DATABITS_9 0x1000
00770
00773 #define SERIAL_STOPBITS_1 0x0000
00774
00777 #define SERIAL_STOPBITS_2 0x2000
00778
00781 #define SERIAL_PARITY_NONE 0x0000
00782
00785 #define SERIAL_PARITY_EVEN 0x0400
00786
00789 #define SERIAL_PARITY_ODD 0x0600
00790
00793 #define SERIAL_8N1 0x0000
00794
00811 void usartInit(PROS_FILE *usart, unsigned int baud, unsigned int flags);
00821 void usartShutdown(PROS_FILE *usart);
00822
00823 // ----- Character input and output -----
00824
00828 #define stdout ((PROS_FILE *)3)
00829
00832 #define stdin ((PROS_FILE *)3)
00833
00836 #define uart1 ((PROS_FILE *)1)
00837
00840 #define uart2 ((PROS_FILE *)2)
00841
00842 #ifndef EOF
00843
00846 #define EOF ((int)-1)
00847 #endif
00848
00849 #ifndef SEEK_SET
00850
00854 #define SEEK_SET 0
00855 #endif
00856 #ifndef SEEK_CUR
00857
00861 #define SEEK_CUR 1
00862 #endif
00863 #ifndef SEEK_END
00864
00868 #define SEEK_END 2
00869 #endif
```

```
00870
00877 void fclose(PROS_FILE *stream);
00890 int fcount(PROS_FILE *stream);
00901 int fdelete(const char *file);
00909 int feof(PROS_FILE *stream);
00921 int fflush(PROS_FILE *stream);
00930 int fgetc(PROS_FILE *stream);
00945 char* fgets(char *str, int num, PROS_FILE *stream);
00969 PROS_FILE * fopen(const char *file, const char *mode);
00979 void fprintf(const char *string, PROS_FILE *stream);
00989 int fputc(int value, PROS_FILE *stream);
00999 int fputs(const char *string, PROS_FILE *stream);
01011 size_t fread(void *ptr, size_t size, size_t count, PROS_FILE *stream);
01021 int fseek(PROS_FILE *stream, long int offset, int origin);
01029 long int ftell(PROS_FILE *stream);
01041 size_t fwrite(const void *ptr, size_t size, size_t count, PROS_FILE *stream);
01047 int getchar();
01055 void print(const char *string);
01066 int putchar(int value);
01073 int puts(const char *string);
01074
01104 int fprintf(PROS_FILE *stream, const char *formatString, ...);
01111 int printf(const char *formatString, ...);
01124 int sprintf(char *buffer, size_t limit, const char *formatString, ...);
01135 int sprint(char *buffer, const char *formatString, ...);
01136
01140 #define LCD_BTN_LEFT 1
01141
01144 #define LCD_BTN_CENTER 2
01145
01148 #define LCD_BTN_RIGHT 4
01149
01158 void lcdClear(PROS_FILE *lcdPort);
01167 void lcdInit(PROS_FILE *lcdPort);
01179 #ifdef DOXYGEN
01180 void lcdPrint(PROS_FILE *lcdPort, unsigned char line, const char *formatString, ...);
01181 #else
01182 void __attribute__ ((format (printf, 3, 4))) lcdPrint(PROS_FILE *lcdPort, unsigned char
line,
01183     const char *formatString, ...);
01184 #endif
01185
01194 unsigned int lcdReadButtons(PROS_FILE *lcdPort);
01203 void lcdSetBacklight(PROS_FILE *lcdPort, bool backlight);
01215 void lcdSetText(PROS_FILE *lcdPort, unsigned char line, const char *buffer);
01221 void lcdShutdown(PROS_FILE *lcdPort);
01222
01223 // ----- Real-time scheduler functions -----
01232 #define TASK_MAX 16
01233
01238 #define TASK_MAX_PRIORITIES 6
01239
01243 #define TASK_PRIORITY_LOWEST 0
01244
01249 #define TASK_PRIORITY_DEFAULT 2
01250
01254 #define TASK_PRIORITY_HIGHEST (TASK_MAX_PRIORITIES - 1)
01255
01262 #define TASK_DEFAULT_STACK_SIZE 512
01263
01270 #define TASK_MINIMAL_STACK_SIZE 64
01271
01275 #define TASK_DEAD 0
01276
01279 #define TASK_RUNNING 1
01280
01284 #define TASK_RUNNABLE 2
01285
01289 #define TASK_SLEEPING 3
01290
01293 #define TASK_SUSPENDED 4
01294
01300 typedef void * TaskHandle;
01306 typedef void * Mutex;
01312 typedef void * Semaphore;
01323 typedef void (*TaskCode)(void *);
01324
01336 TaskHandle taskCreate(TaskCode taskCode, const unsigned int stackDepth, void *parameters,
01337     const unsigned int priority);
01352 void taskDelay(const unsigned long msToDelay);
01373 void taskDelayUntil(unsigned long *previousWakeTime, const unsigned long cycleTime);
```

```

01387 void taskDelete(TaskHandle taskToDelete);
01397 unsigned int taskGetCount();
01409 unsigned int taskGetState(TaskHandle task);
01416 unsigned int taskPriorityGet(const TaskHandle task);
01427 void taskPrioritySet(TaskHandle task, const unsigned int newPriority);
01438 void taskResume(TaskHandle taskToResume);
01454 TaskHandle taskRunLoop(void (*fn)(void), const unsigned long increment);
01463 void taskSuspend(TaskHandle taskToSuspend);
01464
01477 Semaphore semaphoreCreate();
01489 bool semaphoreGive(Semaphore semaphore);
01499 bool semaphoreTake(Semaphore semaphore, const unsigned long blockTime);
01506 void semaphoreDelete(Semaphore semaphore);
01507
01520 Mutex mutexCreate();
01528 bool mutexGive(Mutex mutex);
01540 bool mutexTake(Mutex mutex, const unsigned long blockTime);
01547 void mutexDelete(Mutex mutex);
01548
01554 void delay(const unsigned long time);
01565 void delayMicroseconds(const unsigned long us);
01575 unsigned long micros();
01584 unsigned long millis();
01590 void wait(const unsigned long time);
01597 void waitUntil(unsigned long *previousWakeTime, const unsigned long time);
01605 void watchdogInit();
01611 void standaloneModeEnable();
01612
01613 // End C++ extern to C
01614 #ifdef __cplusplus
01615 }
01616 #endif
01617
01618 #endif

```

5.3 include/auto.h File Reference

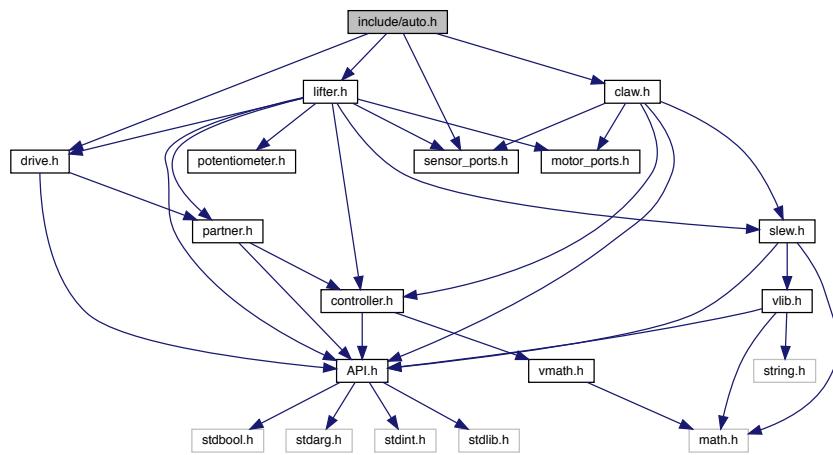
Autonomous declarations and macros.

```

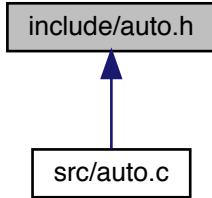
#include "drive.h"
#include "sensor_ports.h"
#include "lifter.h"
#include "claw.h"

```

Include dependency graph for auto.h:



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



Macros

- `#define FRONT_LEFT_IME 0`
Front left motor integrated motor encoder.
- `#define GOAL_HEIGHT 1325`
The height of the goal using potentiometer readings.
- `#define MID_LEFT_DRIVE 1`
Middle left motor integrated motor encoder.
- `#define MID_RIGHT_DRIVE 4`
Middle right motor integrated motor encoder.
- `#define STOP_ONE 500`
First Stop position for stationary autonomous.

5.3.1 Detailed Description

Autonomous declarations and macros.

Author

Chris Jerrett

Date

9/18/2017

Definition in file **auto.h**.

5.3.2 Macro Definition Documentation

5.3.2.1 FRONT_LEFT_IME

```
#define FRONT_LEFT_IME 0
```

Front left motor integrated motor encoder.

Definition at line **18** of file **auto.h**.

5.3.2.2 GOAL_HEIGHT

```
#define GOAL_HEIGHT 1325
```

The height of the goal using potentiometer readings.

Definition at line **38** of file **auto.h**.

Referenced by **autonomous()**.

5.3.2.3 MID_LEFT_DRIVE

```
#define MID_LEFT_DRIVE 1
```

Middle left motor integrated motor encoder.

Definition at line **23** of file **auto.h**.

Referenced by **autonomous()**.

5.3.2.4 MID_RIGHT_DRIVE

```
#define MID_RIGHT_DRIVE 4
```

Middle right motor integrated motor encoder.

Definition at line **28** of file **auto.h**.

Referenced by **autonomous()**.

5.3.2.5 STOP_ONE

```
#define STOP_ONE 500
```

First Stop position for stationary autonomous.

Definition at line **33** of file **auto.h**.

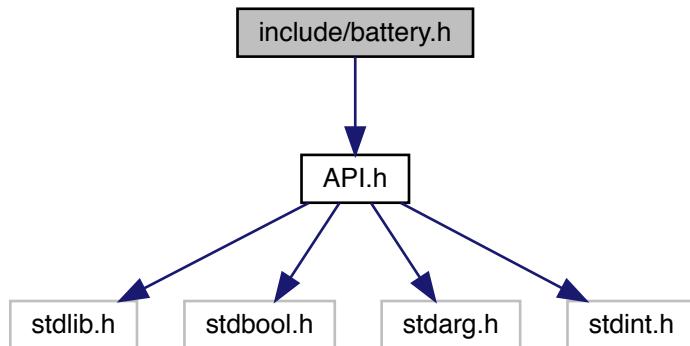
5.4 auto.h

```
00001
00007 #ifndef _AUTO_H_
00008 #define _AUTO_H_
00009
00010 #include "drive.h"
00011 #include "sensor_ports.h"
00012 #include "lifter.h"
00013 #include "claw.h"
00014
00018 #define FRONT_LEFT_IME 0
00019
00023 #define MID_LEFT_DRIVE 1
00024
00028 #define MID_RIGHT_DRIVE 4
00029
00033 #define STOP_ONE 500
00034
00038 #define GOAL_HEIGHT 1325
00039
00040
00041 #endif
```

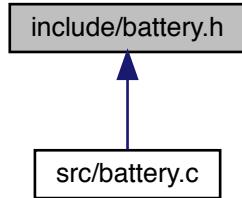
5.5 include/battery.h File Reference

Battery management related functions.

```
#include <API.h>
Include dependency graph for battery.h:
```



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



Macros

- `#define MIN_BACKUP_VOLTAGE 7.8`
The minimum acceptable backup battery voltage before a match.
- `#define MIN_MAIN_VOLTAGE 7.8`
The minimum acceptable main battery voltage before a match.

Functions

- `double backup_battery_voltage ()`
gets the backup battery voltage
- `bool battery_level_acceptable ()`
returns if the batteries are acceptable
- `double main_battery_voltage ()`
gets the main battery voltage

5.5.1 Detailed Description

Battery management related functions.

Author

Chris Jerrett

Date

9/18/2017

Definition in file **battery.h**.

5.5.2 Macro Definition Documentation

5.5.2.1 MIN_BACKUP_VOLTAGE

```
#define MIN_BACKUP_VOLTAGE 7.8
```

The minimum acceptable backup battery voltage before a match.

Definition at line **20** of file **battery.h**.

Referenced by **battery_level_acceptable()**.

5.5.2.2 MIN_MAIN_VOLTAGE

```
#define MIN_MAIN_VOLTAGE 7.8
```

The minimum acceptable main battery voltage before a match.

Definition at line **15** of file **battery.h**.

Referenced by **battery_level_acceptable()**.

5.5.3 Function Documentation

5.5.3.1 backup_battery_voltage()

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

Author

Chris Jerrett

Definition at line **17** of file **battery.c**.

References **powerLevelBackup()**.

Referenced by **battery_level_acceptable()**.

```
00017 {  
00018   return powerLevelBackup() / 1000.0;  
00019 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.2 battery_level_acceptable()

bool battery_level_acceptable ()

returns if the batteries are acceptable

See also

MIN_MAIN_VOLTAGE (p. 99)
MIN_BACKUP_VOLTAGE (p. 99)

Author

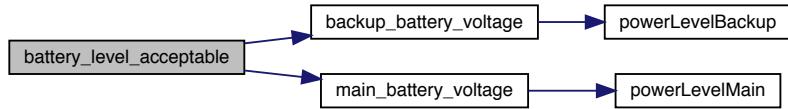
Chris Jerrett

Definition at line **28** of file **battery.c**.

References **backup_battery_voltage()**, **main_battery_voltage()**, **MIN_BACKUP_VOLTAGE**, and **MIN_MAIN_VOLTAGE**.

```
00028
00029     if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
00030     if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
00031     return true;
00032 }
```

Here is the call graph for this function:



5.5.3.3 main_battery_voltage()

double main_battery_voltage ()

gets the main battery voltage

Author

Chris Jerrett

Definition at line 9 of file **battery.c**.

References **powerLevelMain()**.

Referenced by **battery_level_acceptable()**.

```
00009 {  
00010     return powerLevelMain() / 1000.0;  
00011 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.6 battery.h

```

00001
00007 #ifndef _BATTERY_H_
00008 #define _BATTERY_H_
00009
00010 #include <API.h>
00011
00015 #define MIN_MAIN_VOLTAGE 7.8
00016
00020 #define MIN_BACKUP_VOLTAGE 7.8
00021
00026 double main_battery_voltage();
00027
00032 double backup_battery_voltage();
00033
00041 bool battery_level_acceptable();
00042
00043 #endif

```

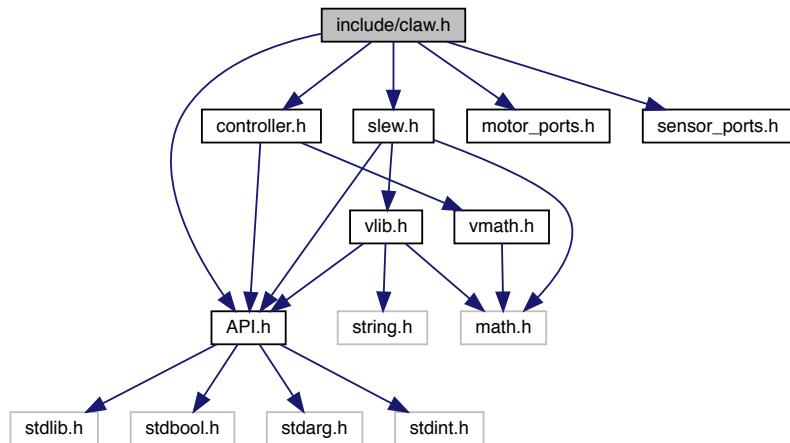
5.7 include/claw.h File Reference

Code for controlling the claw that grabs the cones.

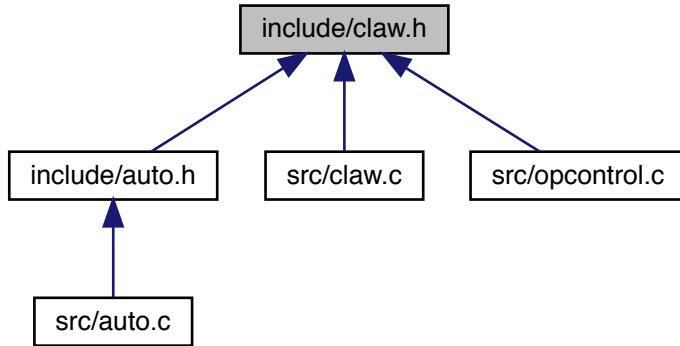
```

#include "slew.h"
#include <API.h>
#include "controller.h"
#include "motor_ports.h"
#include "sensor_ports.h"
Include dependency graph for claw.h:

```



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



Macros

- #define **CLAW_CLOSE_MASTER**, 6, JOY_UP
The joystick parameters for closing the claw.
- #define **CLAW_CLOSE_VAL** 3000
The potentiometer value for a closed claw.
- #define **CLAW_D** .1
The derivative constant for the claw PID controller.
- #define **CLAW_I** 0
The integral constant for the claw PID controller.
- #define **CLAW_OPEN_MASTER**, 6, JOY_DOWN
The joystick parameters for opening the claw.
- #define **CLAW_OPEN_VAL** 1500
The potentiometer value for an open claw.
- #define **CLAW_P** .1
The proportional constant for the claw PID controller.
- #define **MAX_CLAW_SPEED** 50
The max motor value of the claw.
- #define **MIN_CLAW_SPEED** -50
The min motor value of the claw.

Enumerations

- enum **claw_state** { **CLAW_OPEN_STATE**, **CLAW_CLOSE_STATE** }
The different states of the claw.

Functions

- void **close_claw ()**
Drives the motors to close the claw.
- unsigned int **getClawTicks ()**
Gets the claw position in potentiometer ticks.
- void **open_claw ()**
Drives the motors to open the claw.
- void **set_claw_motor (const int v)**
sets the claw motor speed
- void **update_claw ()**
Updates the claw motor values.

5.7.1 Detailed Description

Code for controlling the claw that grabs the cones.

Author

Chris Jerrett, Christian Desimone

Date

8/30/2017

Definition in file **claw.h**.

5.7.2 Macro Definition Documentation

5.7.2.1 CLAW_CLOSE

```
#define CLAW_CLOSE  MASTER, 6, JOY_UP
```

The joystick parameters for closing the claw.

Author

Chris Jerrett

Definition at line **47** of file **claw.h**.

Referenced by **update_claw()**.

5.7.2.2 CLAW_CLOSE_VAL

```
#define CLAW_CLOSE_VAL 3000
```

The potentiometer value for a closed claw.

Author

Chris Jerrett

Definition at line **59** of file **claw.h**.

Referenced by **update_claw()**.

5.7.2.3 CLAW_D

```
#define CLAW_D .1
```

The derivative constant for the claw PID controller.

Author

Chris Jerrett

Definition at line **25** of file **claw.h**.

Referenced by **update_claw()**.

5.7.2.4 CLAW_I

```
#define CLAW_I 0
```

The integral constant for the claw PID controller.

Author

Chris Jerrett

Definition at line **30** of file **claw.h**.

5.7.2.5 CLAW_OPEN

```
#define CLAW_OPEN  MASTER, 6, JOY_DOWN
```

The joystick parameters for opening the claw.

Author

Chris Jerrett

Definition at line **53** of file **claw.h**.

Referenced by **update_claw()**.

5.7.2.6 CLAW_OPEN_VAL

```
#define CLAW_OPEN_VAL 1500
```

The potentiometer value for a open claw.

Author

Chris Jerrett

Definition at line **65** of file **claw.h**.

Referenced by **update_claw()**.

5.7.2.7 CLAW_P

```
#define CLAW_P .1
```

The proportional constant for the claw PID controller.

Author

Chris Jerrett

Definition at line **20** of file **claw.h**.

Referenced by **update_claw()**.

5.7.2.8 MAX_CLAW_SPEED

```
#define MAX_CLAW_SPEED 50
```

The max motor value of the claw.

Author

Chris Jerrett

Definition at line **36** of file **claw.h**.

Referenced by **open_claw()**.

5.7.2.9 MIN_CLAW_SPEED

```
#define MIN_CLAW_SPEED -50
```

The min motor value of the claw.

Author

Chris Jerrett

Definition at line **41** of file **claw.h**.

Referenced by **close_claw()**.

5.7.3 Enumeration Type Documentation

5.7.3.1 claw_state

```
enum claw_state
```

The different states of the claw.

Author

Chris Jerrett

Enumerator

CLAW_OPEN_STATE	
CLAW_CLOSE_STATE	

Definition at line **101** of file **claw.h**.

```
00101      {
00102      CLAW_OPEN_STATE,
00103      CLAW_CLOSE_STATE
00104  };
```

5.7.4 Function Documentation

5.7.4.1 close_claw()

```
void close_claw ( )
```

Drives the motors to close the claw.

Author

Chris Jerrett

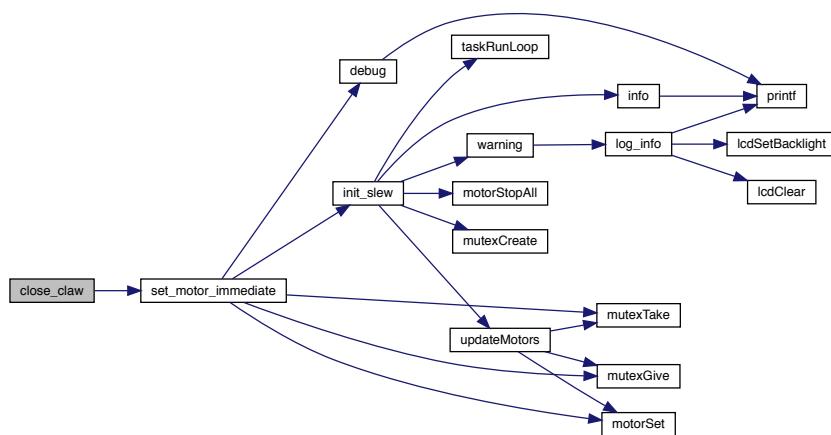
Definition at line **67** of file **claw.c**.

References **CLAW_MOTOR**, **MIN_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00067      {
00068      set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
00069  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.4.2 getClawTicks()

```
unsigned int getClawTicks ( )
```

Gets the claw position in potentiometer ticks.

Author

Chris Jerrett

Definition at line 51 of file **claw.c**.

References **analogRead()**, and **CLAW_POT**.

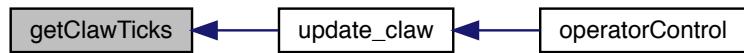
Referenced by **update_claw()**.

```
00051     {  
00052     return analogRead(CLAW_POT);  
00053 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.4.3 open_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

Author

Chris Jerrett

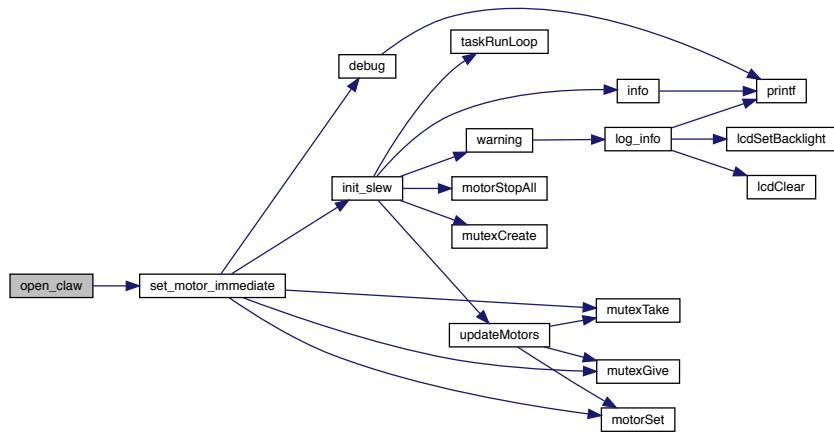
Definition at line **59** of file **claw.c**.

References **CLAW_MOTOR**, **MAX_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00059           {  
00060     set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);  
00061 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.4.4 set_claw_motor()

```
void set_claw_motor (
    const int v )
```

sets the claw motor speed

Author

Chris Jerrett

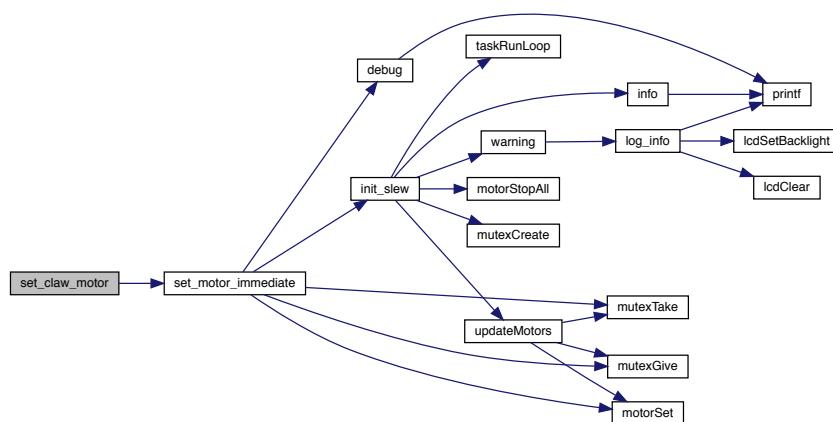
Definition at line 43 of file **claw.c**.

References **CLAW_MOTOR**, and **set_motor_immediate()**.

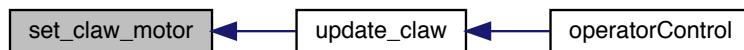
Referenced by **update_claw()**.

```
00043     {
00044     set_motor_immediate(CLAW_MOTOR, v);
00045 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.4.5 update_claw()

```
void update_claw ( )
```

Updates the claw motor values.

Author

Chris Jerrett

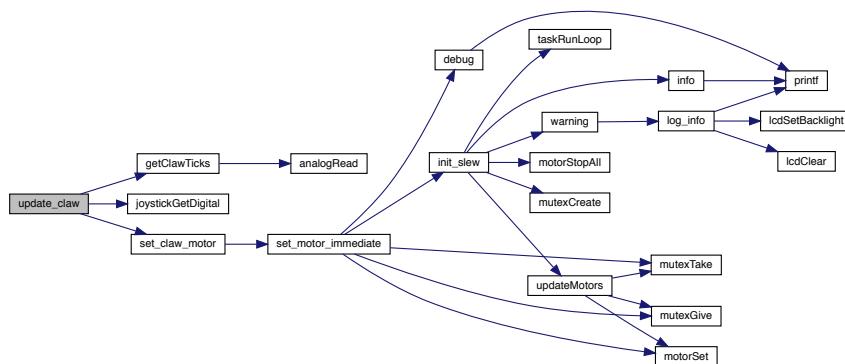
Definition at line 7 of file `claw.c`.

References `CLAW_CLOSE`, `CLAW_CLOSE_STATE`, `CLAW_CLOSE_VAL`, `CLAW_D`, `CLAW_OPEN`, `CLAW_OPEN_STATE`, `CLAW_OPEN_VAL`, `CLAW_P`, `getClawTicks()`, `joystickGetDigital()`, and `set_claw_motor()`.

Referenced by `operatorControl()`.

```
00007      {
00008      //Set the Error used in calculating d
00009      static int last_error = 0;
00010      //Set the initial claw state to open
00011      static enum claw_state state = CLAW_OPEN_STATE;
00012      //Listen for input and either close or open the claw
00013      if(joystickGetDigital(CLAW_CLOSE)){
00014          state = CLAW_CLOSE_STATE;
00015      }
00016      else if(joystickGetDigital(CLAW_OPEN) ){
00017          state = CLAW_OPEN_STATE;
00018      } else {
00019          //set the default motor speed
00020          int p = 0;
00021          //Change the base speed to the difference between the target
00022          // and the current value
00023          if(state == CLAW_OPEN_STATE) {
00024              p = getClawTicks() - CLAW_OPEN_VAL;
00025          } else {
00026              p = getClawTicks() - CLAW_CLOSE_VAL;
00027          }
00028          //Set the d value to the difference between the current p and the last p
00029          int d = (p - last_error);
00030          //Set last error for use the next time the function is run
00031          last_error = p;
00032          //Construct the final motor speed value
00033          int motor = CLAW_P * p + CLAW_D * d;
00034          //Set the motor speed
00035          set_claw_motor(motor);
00036      }
00037 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



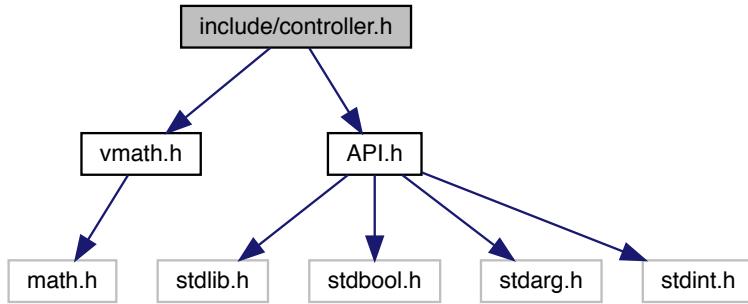
5.8 claw.h

```
00001
00007 #ifndef _CLAW_H_
00008 #define _CLAW_H_
00009
00010 #include "slew.h"
00011 #include <API.h>
00012 #include "controller.h"
00013 #include "motor_ports.h"
00014 #include "sensor_ports.h"
00015
00020 #define CLAW_P .1
00021
00025 #define CLAW_D .1
00026
00030 #define CLAW_I 0
00031
00036 #define MAX_CLAW_SPEED 50
00037
00041 #define MIN_CLAW_SPEED -50
00042
00047 #define CLAW_CLOSE MASTER, 6, JOY_UP
00048
00053 #define CLAW_OPEN MASTER, 6, JOY_DOWN
00054
00059 #define CLAW_CLOSE_VAL 3000
00060
00065 #define CLAW_OPEN_VAL 1500
00066
00071 void update_claw();
00072
00077 void set_claw_motor(const int v);
00078
00083 unsigned int getClawTicks();
00084
00089 void open_claw();
00090
00095 void close_claw();
00096
00101 enum claw_state {
00102     CLAW_OPEN_STATE,
00103     CLAW_CLOSE_STATE
00104 };
00105
00106 #endif
```

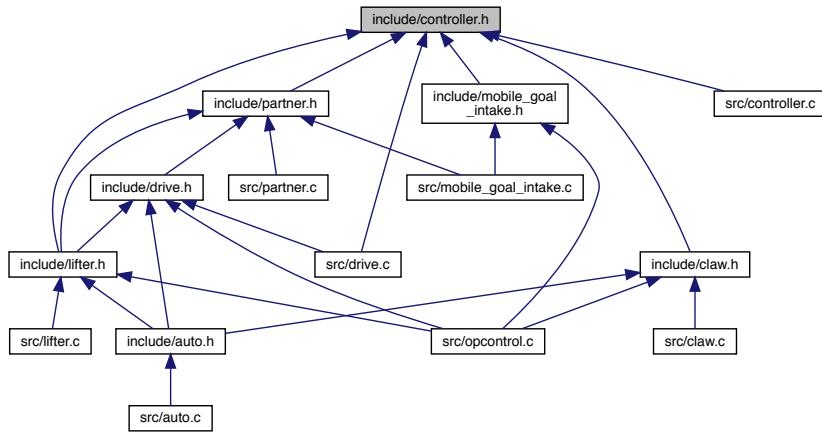
5.9 include/controller.h File Reference

controller definitions, macros and functions to assist with usig the vex controllers.

```
#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_BUMPERS** 6
- #define **LEFT_BUTTONS** 7
- #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_BUMPERS 5`
 - `#define RIGHT_BUTTONS 8`
 - `#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)`
Gets the location of a joystick on the controller.

5.9.1 Detailed Description

controller definitions, macros and functions to assist with usig the vex controllers.

Author

Chris Jerrett, Christian Desimone

Date

9/9/2017

Definition in file **controller.h**.

5.9.2 Macro Definition Documentation

5.9.2.1 LEFT_BUMPERS

```
#define LEFT_BUMPERS 6
```

Definition at line **18** of file **controller.h**.

5.9.2.2 LEFT_BUTTONS

```
#define LEFT_BUTTONS 7
```

Definition at line **16** of file **controller.h**.

5.9.2.3 LEFT_JOY_X

```
#define LEFT_JOY_X 4
```

the left x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **53** of file **controller.h**.

Referenced by **get_joystick_cord()**.

5.9.2.4 LEFT_JOY_Y

```
#define LEFT_JOY_Y 3
```

the left y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **60** of file **controller.h**.

Referenced by **get_joystick_cord()**.

5.9.2.5 MASTER

```
#define MASTER 1
```

the master controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **25** of file **controller.h**.

Referenced by **update_drive_motors()**, and **updateIntake()**.

5.9.2.6 PARTNER

```
#define PARTNER 2
```

the slave/partner controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **32** of file **controller.h**.

Referenced by **update_control()**, **update_drive_motors()**, and **updateIntake()**.

5.9.2.7 RIGHT_BUMPERS

```
#define RIGHT_BUMPERS 5
```

Definition at line **17** of file **controller.h**.

5.9.2.8 RIGHT_BUTTONS

```
#define RIGHT_BUTTONS 8
```

Definition at line **15** of file **controller.h**.

5.9.2.9 RIGHT_JOY_X

```
#define RIGHT_JOY_X 1
```

the right x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **39** of file **controller.h**.

Referenced by **get_joystick_cord()**.

5.9.2.10 RIGHT_JOY_Y

```
#define RIGHT_JOY_Y 2
```

the right y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **46** of file **controller.h**.

Referenced by **get_joystick_cord()**.

5.9.3 Enumeration Type Documentation

5.9.3.1 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 **67** of file **controller.h**.

```
00067      {
00069      RIGHT_JOY,
00071      LEFT_JOY,
00072  };
```

5.9.4 Function Documentation**5.9.4.1 get_joystick_cord()**

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

Gets the location of a joystick on the controller.

Author

Chris Jerrett

Definition at line **7** of file **controller.c**.

References **joystickGetAnalog()**, **LEFT_JOY_X**, **LEFT_JOY_Y**, **RIGHT_JOY**, **RIGHT_JOY_X**, **RIGHT_JOY_Y**, **cord::x**, and **cord::y**.

```
00007
00008     int x;
00009     int y;
00010 //Get the joystick value for either the right or left,
00011 //depending on the mode
00012 if(side == RIGHT_JOY) {
00013     y = joystickGetAnalog(controller, RIGHT_JOY_X);
00014     x = joystickGetAnalog(controller, RIGHT_JOY_Y);
00015 } else {
00016     y = joystickGetAnalog(controller, LEFT_JOY_X);
00017     x = joystickGetAnalog(controller, LEFT_JOY_Y);
00018 }
00019 //Define a coordinate for the joystick value
00020 struct cord c;
00021 c.x = x;
00022 c.y = y;
00023 return c;
00024 }
```

Here is the call graph for this function:



5.10 controller.h

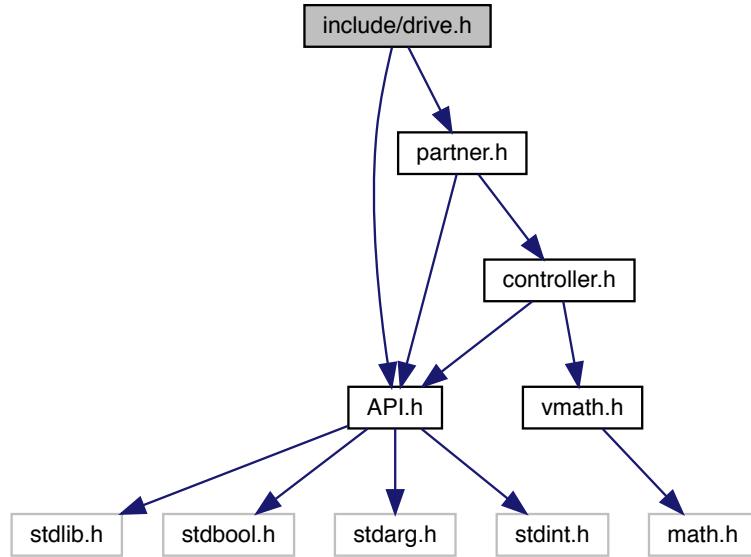
```
00001
00009 #ifndef _CONTROLLER_H_
00010 #define _CONTROLLER_H_
00011
00012 #include "vmath.h"
00013 #include <API.h>
00014
00015 #define RIGHT_BUTTONS 8
00016 #define LEFT_BUTTONS 7
00017 #define RIGHT_BUMPERS 5
00018 #define LEFT_BUMPERS 6
00019
00025 #define MASTER 1
00026
00032 #define PARTNER 2
00033
00039 #define RIGHT_JOY_X 1
00040
00046 #define RIGHT_JOY_Y 2
00047
00053 #define LEFT_JOY_X 4
00054
00060 #define LEFT_JOY_Y 3
00061
00067 enum joystick {
00069     RIGHT_JOY,
00071     LEFT_JOY,
00072 };
00073
00078 struct cord get_joystick_cord(enum joystick side, int controller);
00079
00080 #endif
```

5.11 include/drive.h File Reference

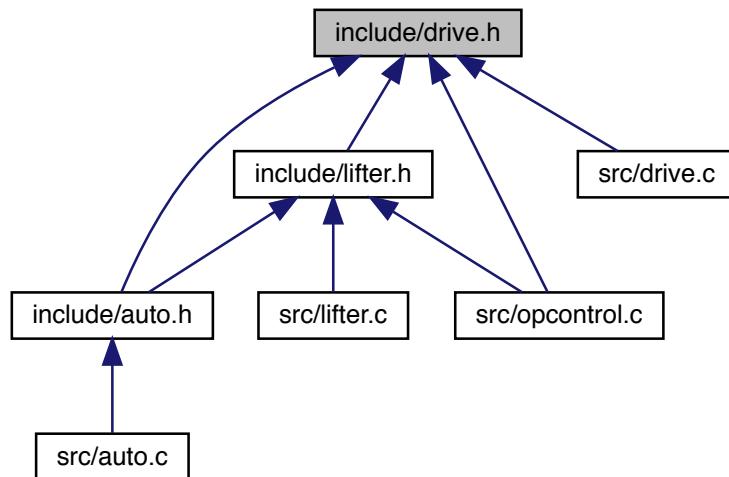
Drive base definitions and enumerations.

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

Include dependency graph for drive.h:



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



Macros

- `#define THRESHOLD 10`

The dead spot on the controller to avoid running motors at low speeds.

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 setThresh (int t)`

Sets the deadzone threshold on the drive.

- `void update_drive_motors ()`

Updates the drive motors during teleop.

5.11.1 Detailed Description

Drive base definitions and enumerations.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **drive.h**.

5.11.2 Macro Definition Documentation

5.11.2.1 THRESHOLD

```
#define THRESHOLD 10
```

The dead spot on the controller to avoid running motors at low speeds.

Definition at line **18** of file **drive.h**.

Referenced by **joystickExp()**, and **update_lifter()**.

5.11.3 Typedef Documentation

5.11.3.1 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.

5.11.4 Enumeration Type Documentation

5.11.4.1 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 **26** of file **drive.h**.

```
00026
00027     LEFT,
00028     BOTH,
00029     RIGHT
00030 } side_t;
```

5.11.5 Function Documentation

5.11.5.1 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

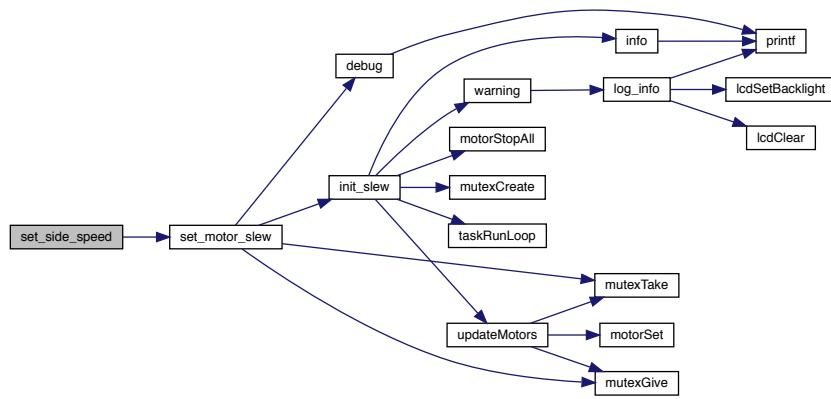
Definition at line **68** of file **drive.c**.

References **BOTH**, **LEFT**, **MOTOR_BACK_LEFT**, **MOTOR_BACK_RIGHT**, **MOTOR_FRONT_LEFT**, **MOTOR_FRONT_RIGHT**, **MOTOR_MIDDLE_LEFT**, **MOTOR_MIDDLE_RIGHT**, **RIGHT**, and **set_motor_slew()**.

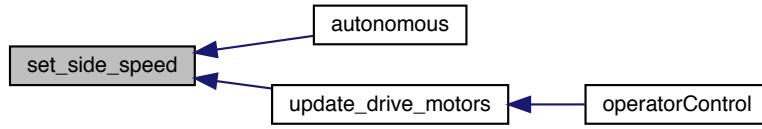
Referenced by **autonomous()**, and **update_drive_motors()**.

```
00068
00069     if(side == RIGHT || side == BOTH){
00070         set_motor_slew(MOTOR_BACK_RIGHT , -speed);
00071         set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
00072         set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
00073     }
00074     if(side == LEFT || side == BOTH){
00075         set_motor_slew(MOTOR_BACK_LEFT, speed);
00076         set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
00077         set_motor_slew(MOTOR_FRONT_LEFT, speed);
00078     }
00079 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.5.2 setThresh()

```
void setThresh (
    int t )
```

Sets the deadzone threshhold on the drive.

Author

Chris Jerrett
Christian Desimone

Definition at line **25** of file **drive.c**.

References **thresh**.

```
00025
00026     thresh = t;
00027 }
```

5.11.5.3 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

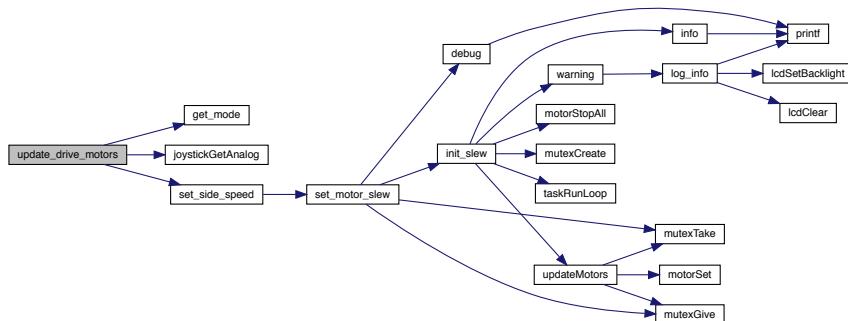
Definition at line 34 of file **drive.c**.

References **get_mode()**, **joystickGetAnalog()**, **LEFT**, **MASTER**, **PARTNER**, **PARTNER_CONTROLLER_MODE**, **RIGHT**, **set_side_speed()**, **thresh**, **cord::x**, and **cord::y**.

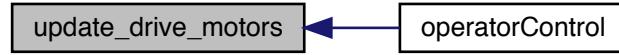
Referenced by **operatorControl()**.

```
00034             {
00035     //Get the joystick values from the controller
00036     int x = 0;
00037     int y = 0;
00038     if(get_mode() == PARTNER_CONTROLLER_MODE) {
00039         x = (joystickGetAnalog(PARTNER, 3));
00040         y = (joystickGetAnalog(PARTNER, 1));
00041     } else {
00042         x = -(joystickGetAnalog(MASTER, 3));
00043         y = (joystickGetAnalog(MASTER, 1));
00044     }
00045     //Make sure the joystick values are significant enough to change the motors
00046     if(x < thresh && x > -thresh){
00047         x = 0;
00048     }
00049     if(y < thresh && y > -thresh){
00050         y = 0;
00051     }
00052     //Create motor values for the left and right from the x and y of the joystick
00053     int r = (x + y);
00054     int l = -(x - y);
00055
00056     //Set the drive motors
00057     set_side_speed(LEFT, l);
00058     set_side_speed(RIGHT, -r);
00059 }
00060 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



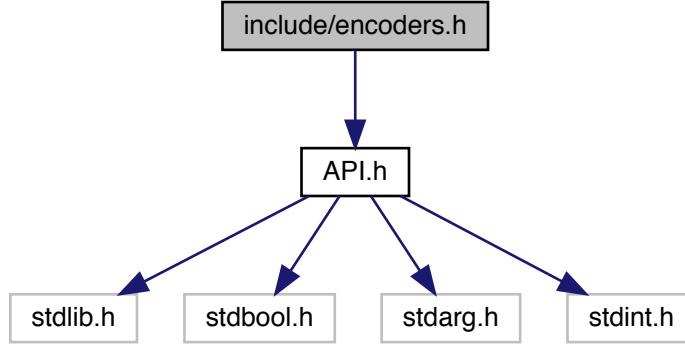
5.12 drive.h

```
00001
00008 #ifndef _DRIVE_H_
00009 #define _DRIVE_H_
00010
00011 #include <API.h>
00012 #include "partner.h"
00013
00018 #define THRESHOLD 10
00019
00026 typedef enum side{
00027     LEFT,
00028     BOTH,
00029     RIGHT
00030 } side_t;
00031
00038 void set_side_speed(side_t side, int speed);
00039
00044 void setThresh(int t);
00045
00051 void update_drive_motors();
00052
00053 #endif
```

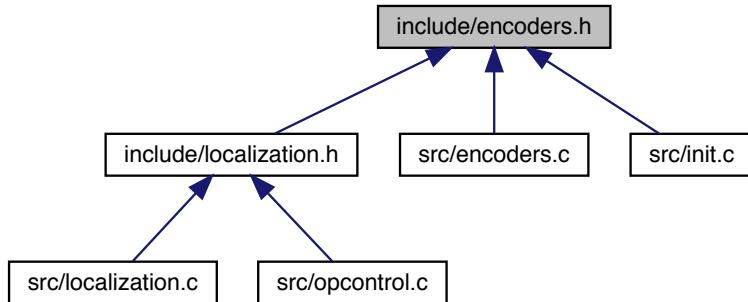
5.13 include/encoders.h File Reference

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 6`

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.

5.13.1 Detailed Description

wrapper around encoder functions

Author

Chris Jerrett, Christian Desimone

Date

9/9/2017

Definition in file **encoders.h**.

5.13.2 Macro Definition Documentation

5.13.2.1 IME_NUMBER

```
#define IME_NUMBER 6
```

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

See also

[init_encoders\(\)](#) (p. 131)

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#) (p. 129)

Definition at line **20** of file **encoders.h**.

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

5.13.3 Function Documentation

5.13.3.1 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 **23** of file **encoders.c**.

References **imeGet()**.

```
00023
00024     int i = 0;
00025     imeGet(address, &i);
00026     return i;
00027 }
```

Here is the call graph for this function:



5.13.3.2 get_encoder_velocity()

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

Gets the encoder reads.

Author

Chris Jerrett

Date

9/15/2017

Definition at line **34** of file **encoders.c**.

References **imeGetVelocity()**.

```
00034     {
00035     int i = 0;
00036     imeGetVelocity(address, &i);
00037     return i;
00038 }
```

Here is the call graph for this function:



5.13.3.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also**IME_NUMBER** (p. 129)Definition at line **10** of file **encoders.c**.References **IME_NUMBER**, and **imeInitializeAll()**.

```
00010             {
00011     #ifdef IME_NUMBER
00012     return imeInitializeAll() == IME_NUMBER;
00013     #else
00014     return imeInitializeAll();
00015     #endif
00016 }
```

Here is the call graph for this function:

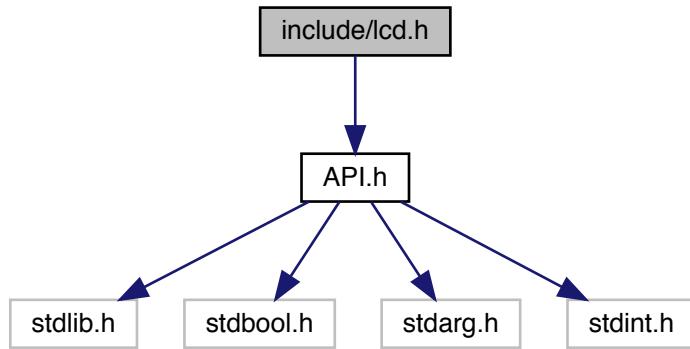
**5.14 encoders.h**

```
00001
00007 #ifndef _ENCODERS_H_
00008 #define _ENCODERS_H_
00009
00010 #include <API.h>
00011
00020 #define IME_NUMBER 6
00021
00028 bool init_encoders();
00029
00035 int get_encoder_ticks(unsigned char address);
00036
00042 int get_encoder_velocity(unsigned char address);
00043
00044 #endif
```

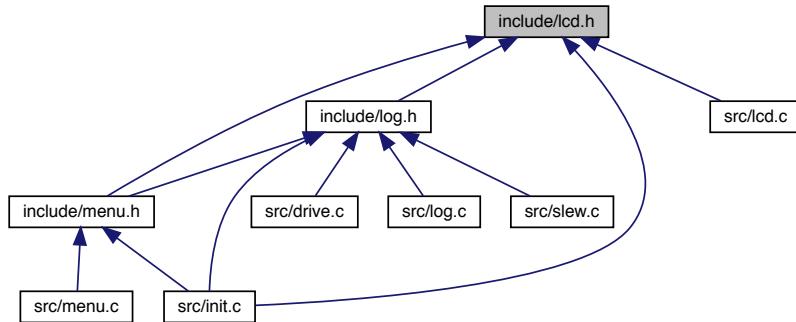
5.15 include/lcd.h File Reference

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 formated string to a line on the lcd. Smilar 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.

5.15.1 Detailed Description

LCD wrapper functions and macros.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **lcd.h**.

5.15.2 Macro Definition Documentation

5.15.2.1 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()**.

5.15.2.2 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()**.

5.15.3 Enumeration Type Documentation

5.15.3.1 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**.

```
00036      {
00038     RELEASED = false,
00040     PRESSED = true,
00041 } button_state;
```

5.15.4 Function Documentation**5.15.4.1 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 urart port of the lcd screen
------------	----------------------------------

See also

uart1 (p. 33)
uart2 (p. 33)

Author

Chris Jerrett

Date

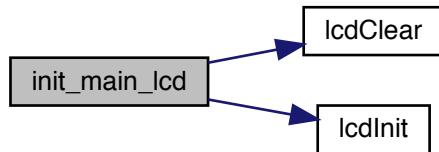
9/9/2017

Definition at line **60** of file **lcd.c**.

References **lcd_port**, **lcdClear()**, and **lcdInit()**.

```
00060      {
00061     lcdInit(lcd);
00062     lcdClear(lcd);
00063     lcd_port = lcd;
00064 }
```

Here is the call graph for this function:



5.15.4.2 lcd_clear()

```
void lcd_clear ( )
```

Clears the lcd.

Author

Chris Jerrett

Date

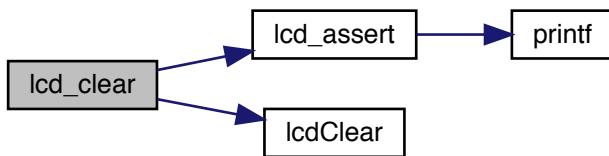
9/9/2017

Definition at line **46** of file **lcd.c**.

References **lcd_assert()**, **lcd_port**, and **lcdClear()**.

```
00046      {
00047      lcd_assert();
00048      lcdClear(lcd_port);
00049 }
```

Here is the call graph for this function:



5.15.4.3 `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 \(p. 8\)](#)

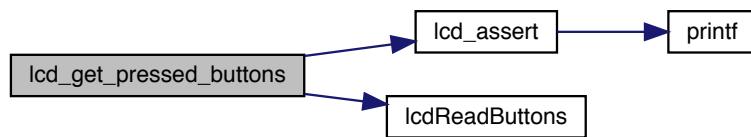
Definition at line **27** of file [lcd.c](#).

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

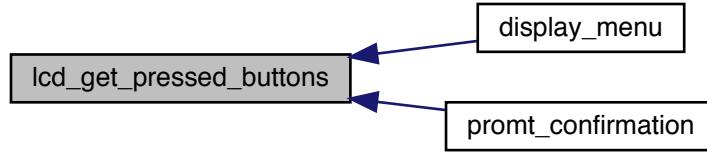
Referenced by `display_menu()`, and `prompt_confirmation()`.

```
00027             {
00028     lcd_assert();
00029     unsigned int btn_binary = lcdReadButtons(lcd_port);
00030     bool left = btn_binary & 0x1; //0001
00031     bool middle = btn_binary & 0x2; //0010
00032     bool right = btn_binary & 0x4; //0100
00033     lcd_buttons btns;
00034     btns.left = left ? PRESSED : RELEASED;
00035     btns.middle = middle ? PRESSED : RELEASED;
00036     btns.right = right ? PRESSED : RELEASED;
00037
00038     return btns;
00039 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.4.4 `lcd_print()`

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

prints a string to a line on the lcd

Parameters

<code>line</code>	the line to print on
<code>str</code>	string to print

Author

Chris Jerrett

Date

9/9/2017

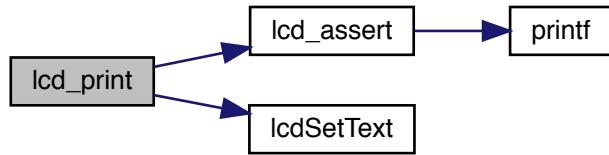
Definition at line **73** of file **lcd.c**.

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

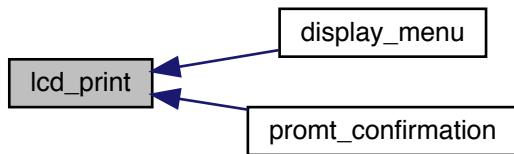
Referenced by `display_menu()`, and `prompt_confirmation()`.

```
00073
00074     lcd_assert();
00075     lcdSetText(lcd_port, line, str);
00076 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.4.5 lcd_printf()

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

prints a formated 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

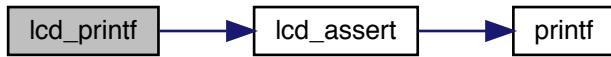
Date

9/9/2017

Definition at line **85** of file **lcd.c**.References **lcd_assert()**, and **lcd_port**.

```
00085 {  
00086     lcd_assert();  
00087     lcdPrint(lcd_port, line, format_str);  
00088 }
```

Here is the call graph for this function:

**5.15.4.6 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

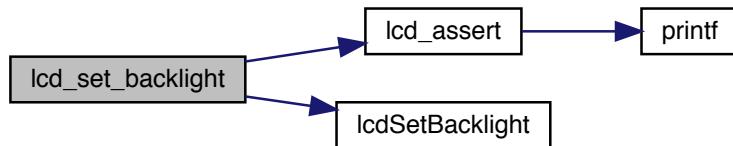
9/9/2017

Definition at line **96** of file **lcd.c**.References **lcd_assert()**, **lcd_port**, and **LcdSetBacklight()**.

```

00096     {
00097     lcd_assert();
00098     lcdSetBacklight(lcd_port, state);
00099 }
```

Here is the call graph for this function:



5.15.4.7 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

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

Author

Chris Jerrett

Date

9/9/2017

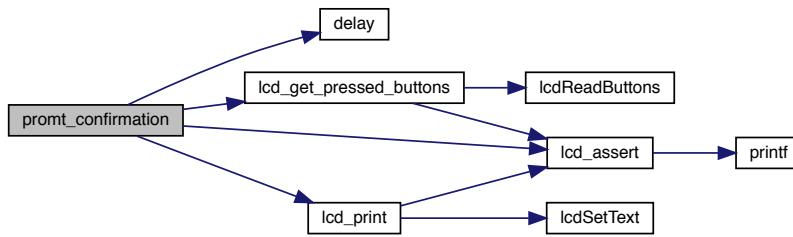
Definition at line **110** of file **Lcd.c**.

References **delay()**, **Lcd_assert()**, **Lcd_get_pressed_buttons()**, **Lcd_print()**, and **PRESSED**.

```

00110     {
00111     lcd_assert();
00112     lcd_print(1, confirm_text);
00113     while(lcd_get_pressed_buttons().middle != PRESSED) {
00114         delay(200);
00115     }
00116 }
```

Here is the call graph for this function:



5.16 lcd.h

```

00001
00008 #ifndef _LCD_H_
00009 #define _LCD_H_
0010
0011 #include <API.h>
0012
0018 #define TOP_ROW 1
0019
0025 #define BOTTOM_ROW 2
0026
0036 typedef enum {
0038     RELEASED = false,
0040     PRESSED = true,
0041 } button_state;
0042
0048 typedef struct {
0049     button_state left;
0050     button_state middle;
0051     button_state right;
0052 } lcd_buttons;
0053
0054
0062 lcd_buttons lcd_get_pressed_buttons();
0063
0069 void lcd_clear();
0070
0080 void init_main_lcd(FILE *lcd);
0081
0089 void lcd_print(unsigned int line, const char *str);
0090
0098 void lcd_printf(unsigned int line, const char *format_str, ...);
0099
0106 void lcd_set_backlight(bool state);
0107
0117 void prompt_confirmation(const char *confirm_text);
00118
00119 #endif
  
```

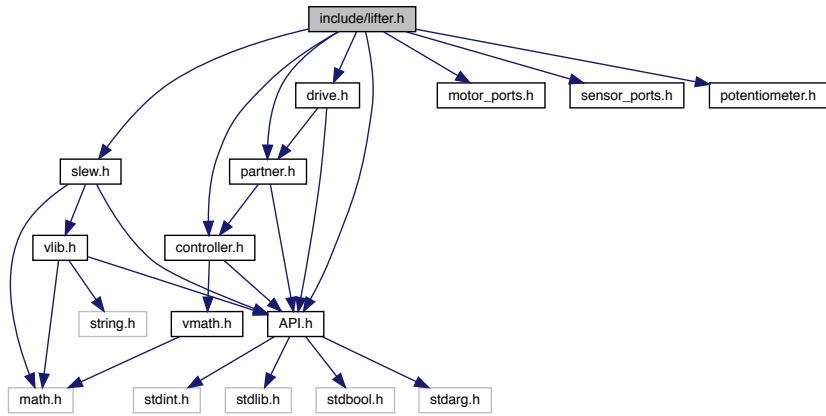
5.17 include/lifter.h File Reference

Declarations and macros for controlling and manipulating the lifter.

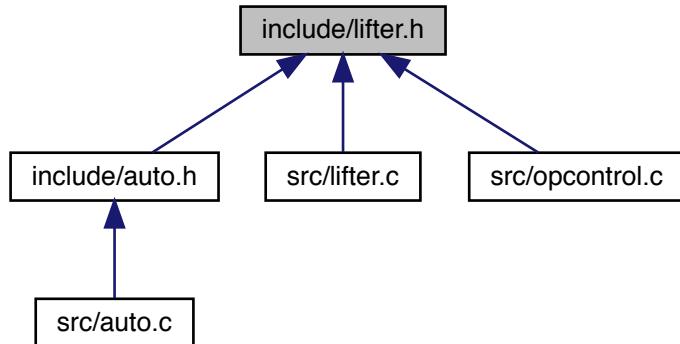
```

#include <API.h>
#include "motor_ports.h"
#include "sensor_ports.h"
  
```

```
#include "slew.h"
#include "controller.h"
#include "potentiometer.h"
#include "partner.h"
#include "drive.h"
Include dependency graph for lifter.h:
```



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



Macros

- `#define HEIGHT 19.1 - 3.8`
The integral constant for the lifter PID.
- `#define INIT_ROTATION 680`

- **#define LIFTER_D 0**
The initial rotation of the lifter potentiometer at height zero.
- **#define LIFTER_DOWN_MASTER, 5, JOY_DOWN**
The lifter down controller params.
- **#define LIFTER_DOWN_PARTNER_PARTNER, 5, JOY_DOWN**
The lifter down controller params for the partner.
- **#define LIFTER_DRIVER_LOAD_MASTER, RIGHT_BUTTONS, JOY_RIGHT**
Height to raise lifter to driver preload height.
- **#define LIFTER_I 0**
The integral constant for the lifter PID.
- **#define LIFTER_P .15**
The proportional constant for the lifter PID.
- **#define LIFTER_UP_MASTER, 5, JOY_UP**
The lifter up controller params.
- **#define LIFTER_UP_PARTNER_PARTNER, 5, JOY_UP**
The lifter up controller params for the partner.
- **#define THRESHOLD 10**
The threshold of a significant speed for the lifter.

Functions

- **double getLifterHeight ()**
Gets the height of the lifter in inches.
- **int getLifterTicks ()**
Gets the value of the lifter pot.
- **float lifterPotentiometerToDegree (int x)**
height of the lifter in degrees from 0 height
- **void set_lifter_motors (const int)**
Sets the lifter motors to the given value.
- **void set_lifter_pos (int pos)**
Sets the lifter positions to the given value.
- **void update_lifter ()**
Updates the lifter in teleop.

5.17.1 Detailed Description

Declarations and macros for controlling and manipulating the lifter.

Author

Chris Jerrett, Christian Desimone

Date

8/27/2017

Definition in file **lifter.h**.

5.17.2 Macro Definition Documentation

5.17.2.1 HEIGHT

```
#define HEIGHT 19.1 - 3.8
```

The integral constant for the lifter PID.

Definition at line **48** of file **lifter.h**.

5.17.2.2 INIT_ROTATION

```
#define INIT_ROTATION 680
```

The initial rotation of the lifter potentiometer at height zero.

Definition at line **22** of file **lifter.h**.

Referenced by **lifterPotentiometerToDegree()**.

5.17.2.3 LIFTER_D

```
#define LIFTER_D 0
```

The derivative constant for the lifter PID.

Definition at line **32** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.4 LIFTER_DOWN

```
#define LIFTER_DOWN MASTER, 5, JOY_DOWN
```

The lifter down controller params.

Definition at line **58** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.5 LIFTER_DOWN_PARTNER

```
#define LIFTER_DOWN_PARTNER PARTNER, 5, JOY_DOWN
```

The lifter down controller params for the partner.

Definition at line **73** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.6 LIFTER_DRIVER_LOAD

```
#define LIFTER_DRIVER_LOAD MASTER, RIGHT_BUTTONS, JOY_RIGHT
```

Height to raise lifter to driver preload height.

Definition at line **63** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.7 LIFTER_I

```
#define LIFTER_I 0
```

The integral constant for the lifter PID.

Definition at line **42** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.8 LIFTER_P

```
#define LIFTER_P .15
```

The proportional constant for the lifter PID.

Definition at line **27** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.9 LIFTER_UP

```
#define LIFTER_UP  MASTER, 5, JOY_UP
```

The lifter up controller params.

Definition at line **53** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.10 LIFTER_UP_PARTNER

```
#define LIFTER_UP_PARTNER PARTNER, 5, JOY_UP
```

The lifter up controller params for the partner.

Definition at line **68** of file **lifter.h**.

Referenced by **update_lifter()**.

5.17.2.11 THRESHOLD

```
#define THRESHOLD 10
```

The threshold of a significant speed for the lifter.

Definition at line **37** of file **lifter.h**.

5.17.3 Function Documentation

5.17.3.1 getLifterHeight()

```
double getLifterHeight ( )
```

Gets the height of the lifter in inches.

Returns

the height of the lifter.

Author

Chris Jerrett

Date

9/17/2017

Definition at line **137** of file **lifter.c**.

References **getLifterTicks()**.

```
00137             {
00138     unsigned int ticks = getLifterTicks();
00139     return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) + 0.0198 * ticks + 2.3033);
00140 }
```

Here is the call graph for this function:



5.17.3.2 getLifterTicks()

```
int getLifterTicks ( )
```

Gets the value of the lifter pot.

Returns

the value of the pot.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **126** of file **lifter.c**.

References **analogRead()**, and **LIFTER**.

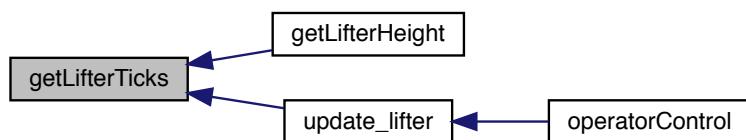
Referenced by **getLifterHeight()**, and **update_lifter()**.

```
00126      {  
00127     return analogRead(LIFTER);  
00128 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.17.3.3 lifterPotentiometerToDegree()

```
float lifterPotentiometerToDegree (
    int x )
```

height of the lifter in degrees from 0 height

Parameters

x	the pot value
---	---------------

Returns

the positions in degrees

Author

Chris Jerrett

Date

10/13/2017

Definition at line **115** of file **lifter.c**.

References **DEG_MAX**, **INIT_ROTATION**, and **TICK_MAX**.

```
00115             {
00116     return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
00117 }
```

5.17.3.4 set_lifter_motors()

```
void set_lifter_motors (
    const int v )
```

Sets the lifter motors to the given value.

Parameters

v	value for the lifter motor. Between -128 - 127, any values outside are clamped.
---	---

Author

Chris Jerrett

Date

9/9/2017

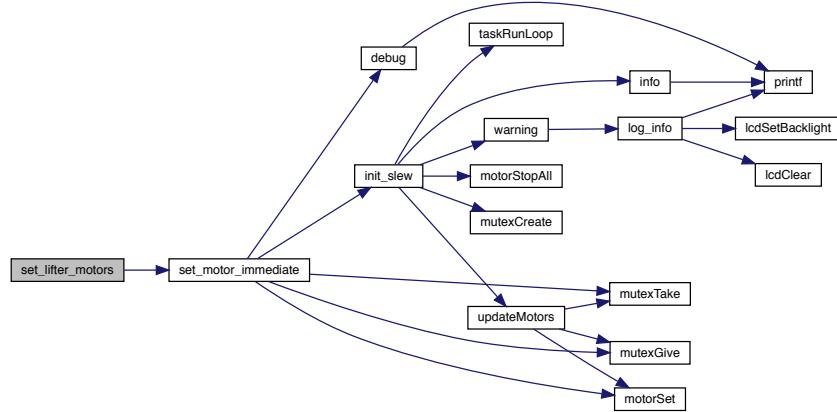
Definition at line **10** of file **lifter.c**.

References **MOTOR_LIFT_TOP_LEFT**, **MOTOR_LIFT_TOP_RIGHT**, and **set_motor_immediate()**.

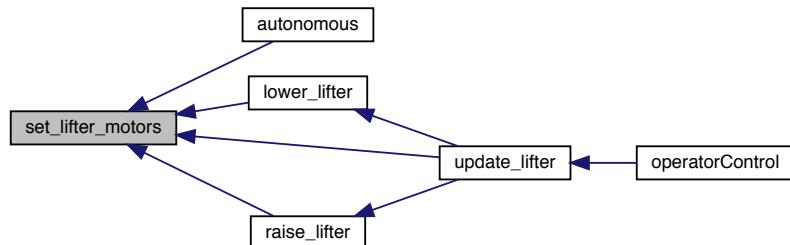
Referenced by **autonomous()**, **lower_lifter()**, **raise_lifter()**, and **update_lifter()**.

```
00010      {
00011      set_motor_immediate(MOTOR_LIFT_TOP_RIGHT, -v);
00012      set_motor_immediate(MOTOR_LIFT_TOP_LEFT, -v);
00013 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.17.3.5 set_lifter_pos()

```
void set_lifter_pos (
    int pos )
```

Sets the lifter positions to the given value.

Parameters

<i>pos</i>	The height in inches
------------	----------------------

Author

Chris Jerrett

Date

9/12/2017

Definition at line **22** of file **lifter.c**.

```
00022             {
00023
00024 }
```

5.17.3.6 update_lifter()

```
void update_lifter ( )
```

Updates the lifter in teleop.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **40** of file **lifter.c**.

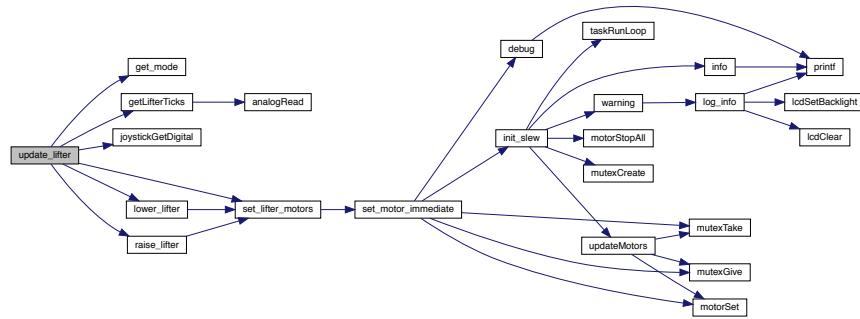
References **get_mode()**, **getLifterTicks()**, **joystickGetDigital()**, **LIFTER_D**, **LIFTER_DOWN**, **LIFTER_DOWN_PARTNER**, **LIFTER_DRIVER_LOAD**, **LIFTER_I**, **LIFTER_P**, **LIFTER_UP**, **LIFTER_UP_PARTNER**, **lower_lifter()**, **MAIN_CONTROLLER_MODE**, **PARTNER_CONTROLLER_MODE**, **raise_lifter()**, **set_lifter_motors()**, and **THRESHOLD**.

Referenced by **operatorControl()**.

```

00040
00041     {
00042         //Establish variables to be used repeatedly
00043         static bool changed = true;
00044         static unsigned int target = 0;
00045         static bool first_run = true;
00046         //Set the target to the current height for the first run
00047         if(first_run) {
00048             target = getLifterTicks();
00049             first_run = false;
00050         }
00051         //Establish the error as 0
00052         static int last_error = 0;
00053         static long long i = 0;
00054         //Check the buttons on the controller indicated by the controller mode
00055         if((joystickGetDigital(LIFTER_UP) && get_mode() == MAIN_CONTROLLER_MODE)
00056             || (joystickGetDigital(LIFTER_UP_PARTNER) && get_mode() ==
00057                 PARTNER_CONTROLLER_MODE)){
00058             changed = true;
00059             i = 0;
00060             //Change the target and start the motion
00061             target = getLifterTicks() + 200;
00062             lower_lifter();
00063         }
00064         else if((joystickGetDigital(LIFTER_DOWN) && get_mode() == MAIN_CONTROLLER_MODE)
00065             || (joystickGetDigital(LIFTER_DOWN_PARTNER) && get_mode() ==
00066                 PARTNER_CONTROLLER_MODE)) {
00067             changed = true;
00068             i = 0;
00069             //Change the target and start the motion
00070             target = getLifterTicks();
00071             raise_lifter();
00072         }
00073         //Raise the lifter to the driver load height
00074         else if(joystickGetDigital(LIFTER_DRIVER_LOAD) && get_mode() ==
00075             MAIN_CONTROLLER_MODE){
00076             changed = true;
00077             i = 0;
00078             int k = 0;
00079             if(getLifterTicks() < 1270){
00080                 lower_lifter();
00081             }
00082             target = 1250;
00083         }
00084         //Change lifter motor values based upon the target
00085         else {
00086             //Don't if we are using the partner controller
00087             if(get_mode() == PARTNER_CONTROLLER_MODE){
00088                 set_lifter_motors(0);
00089                 return;
00090             }
00091             //Define the proportion, derivative, and integral to be used in the motor speed
00092             int p = getLifterTicks() - target;
00093             int d = p - last_error;
00094             last_error = p;
00095             i += p;
00096             int motor = LIFTER_P * p + LIFTER_D * d + LIFTER_I * i;
00097             //Avoid wasting battery if value is insignificant
00098             if (motor < THRESHOLD) {
00099                 set_lifter_motors(0);
00100             } else {
00101                 set_lifter_motors(motor);
00102             }
00103         }
00104     }
00105 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.18 lifter.h

```

00001
00007 #ifndef _LIFTER_H_
00008 #define _LIFTER_H_
00009
00010 #include <API.h>
00011 #include "motor_ports.h"
00012 #include "sensor_ports.h"
00013 #include "slew.h"
00014 #include "controller.h"
00015 #include "potentiometer.h"
00016 #include "partner.h"
00017 #include "drive.h"
00018
00022 #define INIT_ROTATION 680
00023
00027 #define LIFTER_P .15
00028
00032 #define LIFTER_D 0
00033
00037 #define THRESHOLD 10
00038
00042 #define LIFTER_I 0
00043
00044
00048 #define HEIGHT 19.1 - 3.8
00049
00053 #define LIFTER_UP MASTER, 5, JOY_UP
00054
00058 #define LIFTER_DOWN MASTER, 5, JOY_DOWN
00059
00063 #define LIFTER_DRIVER_LOAD MASTER, RIGHT_BUTTONS, JOY_RIGHT
  
```

```

00064
00068 #define LIFTER_UP_PARTNER PARTNER, 5, JOY_UP
00069
00073 #define LIFTER_DOWN_PARTNER PARTNER, 5, JOY_DOWN
00074
00082 void set_lifter_motors(const int);
00083
00091 void set_lifter_pos(int pos);
00092
00099 void update_lifter();
00100
00109 float lifterPotentiometerToDegree(int x);
00110
00118 double getLifterHeight();
00119
00127 int getLifterTicks();
00128
00129 #endif

```

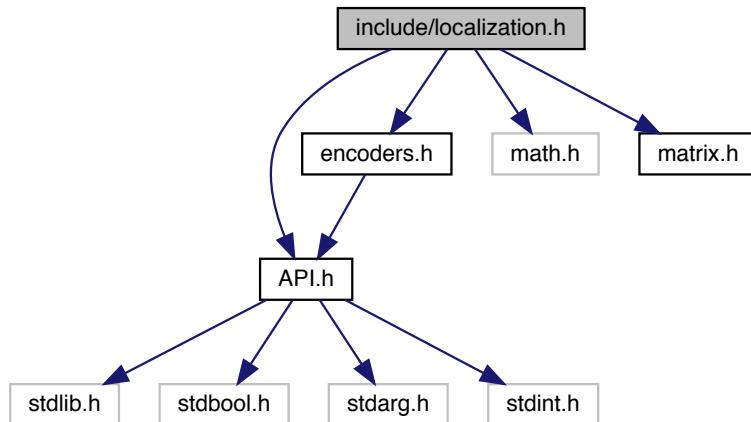
5.19 include/localization.h File Reference

Declarations and macros for determining the location of the robot. [WIP].

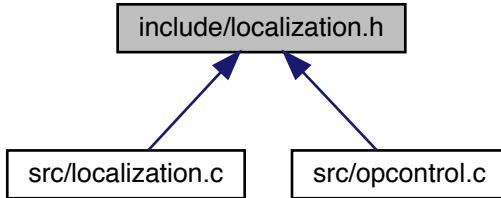
```

#include <API.h>
#include "encoders.h"
#include <math.h>
#include "matrix.h"
Include dependency graph for localization.h:

```



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



Data Structures

- struct **location**

Macros

- #define **LOCALIZATION_UPDATE_FREQUENCY** 0.500

Functions

- struct **location** **get_position** ()
Gets the current position of the robot.
- bool **init_localization** (const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int start_theta)
Starts the localization process.

5.19.1 Detailed Description

Declarations and macros for determining the location of the robot. [WIP].

Author

Chris Jerrett, Christian Desimone

Date

9/27/2017

Definition in file **localization.h**.

5.19.2 Macro Definition Documentation

5.19.2.1 LOCALIZATION_UPDATE_FREQUENCY

```
#define LOCALIZATION_UPDATE_FREQUENCY 0.500
```

How often the localization code updates the position.

Definition at line **18** of file **localization.h**.

Referenced by **calculate_gryo_angular_velocity()**, **init_localization()**, and **integrate_gyro_w()**.

5.19.3 Function Documentation

5.19.3.1 get_position()

```
struct location get_position ( )
```

Gets the current positiuon of the robot.

Parameters

<i>gyro1</i>	The first gyro
--------------	----------------

Returns

the loacation of the robot as a struct.

Definition at line **25** of file **localization.c**.

```
00025
00026
00027 }
```

5.19.3.2 init_localization()

```
bool init_localization (
    const unsigned char gyrol,
    unsigned short multiplier,
    int start_x,
    int start_y,
    int start_theta )
```

Starts the localization process.

Author

Chris Jerrett

Parameters

<code>gyro1</code>	The first gyro The multiplier parameter can tune the gyro to adapt to specific sensors. The default value at this time is 196; higher values will increase the number of degrees reported for a fixed actual rotation, while lower values will decrease the number of degrees reported. If your robot is consistently turning too far, increase the multiplier, and if it is not turning far enough, decrease the multiplier.
--------------------	---

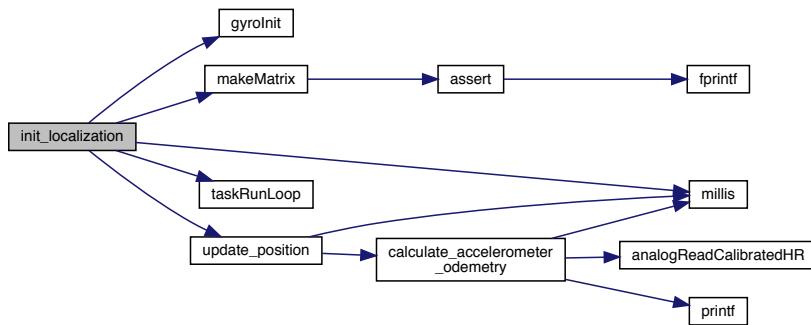
Definition at line 89 of file `localization.c`.

References `g1`, `gyroInit()`, `last_call`, `localization_task`, `LOCALIZATION_UPDATE_FREQUENCY`, `makeMatrix()`, `millis()`, `taskRunLoop()`, and `update_position()`.

```

00089
{
00090     g1 = gyroInit(gyro1, multiplier);
00091     //init state matrix
00092
00093     //one dimensional vector with x, y, theta, acceleration in x and y
00094     state_matrix = makeMatrix(1, 5);
00095     localization_task = taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
00096     last_call = millis();
00097     return true;
00098 }
```

Here is the call graph for this function:



5.20 localization.h

```

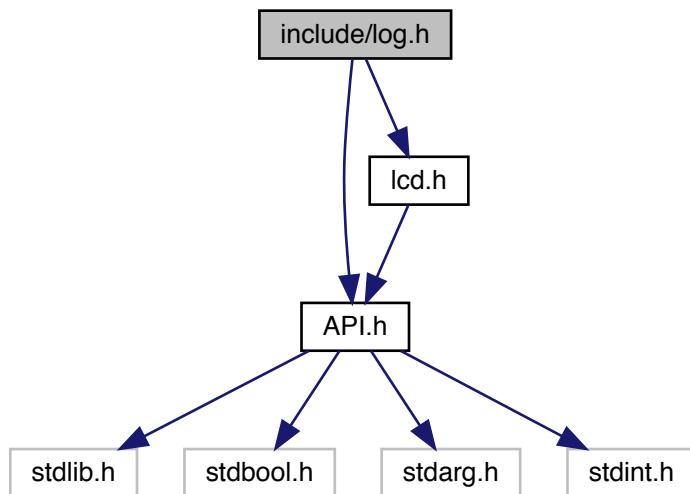
00001
00007 #ifndef _LOCALIZATION_H_
00008 #define _LOCALIZATION_H_
00009
00010 #include <API.h>
00011 #include "encoders.h"
00012 #include <math.h>
00013 #include "matrix.h"
```

```
00014
00018 #define LOCALIZATION_UPDATE_FREQUENCY 0.500
00019
00023 struct location {
00024     int x;
00025     int y;
00026     int theta;
00027 };
00028
00040 bool init_localization(const unsigned char gyrol, unsigned short multiplier, int start_x, int start_y, int
    start_theta);
00041
00048 struct location get_position();
00049
00050 #endif
```

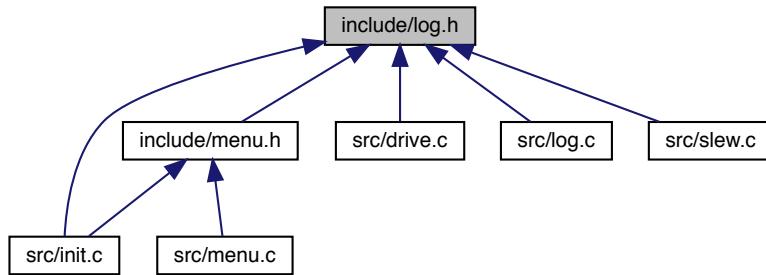
5.21 include/log.h File Reference

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

5.21.1 Detailed Description

Contains logging functions.

Author

Chris Jerrett

Date

9/16/2017

Definition in file **log.h**.

5.21.2 Macro Definition Documentation

5.21.2.1 DEBUG

```
#define DEBUG 4
```

logging only info debug. most verbose level

Author

Chris Jerrett

Date

9/10/17

Definition at line **50** of file **log.h**.

5.21.2.2 ERROR

```
#define ERROR 1
```

logging only errors. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line **27** of file **log.h**.

Referenced by **debug()**, and **info()**.

5.21.2.3 INFO

```
#define INFO 3  
logging only info messages and higher.
```

Author

Chris Jerrett

Date

9/10/17

Definition at line **42** of file **log.h**.

5.21.2.4 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 **19** of file **log.h**.

Referenced by **error()**.

5.21.2.5 WARNING

```
#define WARNING 2
```

logs errors and warnings. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line **35** of file **log.h**.

Referenced by **warning()**.

5.21.3 Function Documentation

5.21.3.1 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 (p. 321)

Parameters

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

Definition at line 37 of file **log.c**.

References **ERROR**, **log_level**, and **printf()**.

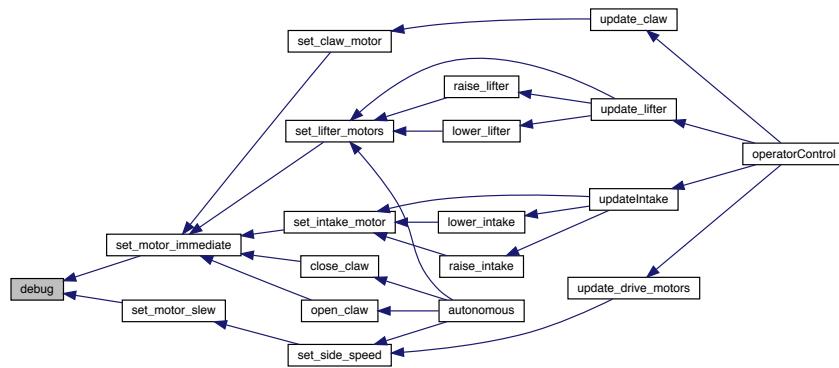
Referenced by **set_motor_immediate()**, and **set_motor_slew()**.

```
00037
00038     if(log_level>ERROR) {
00039         printf("[INFO]: %s\n", debug_message);
00040     }
00041 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.21.3.2 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 (p. 321)

Author

Chris Jerrett

Date

9/10/17

Parameters

<code>error_message</code>	the message
----------------------------	-------------

Definition at line **21** of file **log.c**.

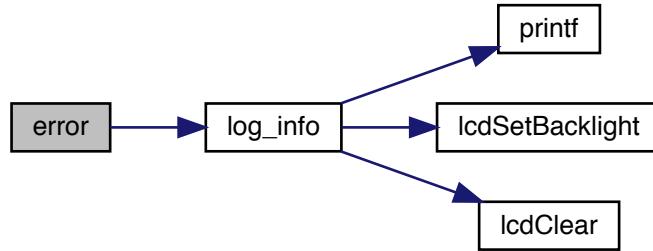
References **log_info()**, **log_level**, and **NONE**.

Referenced by **create_menu()**.

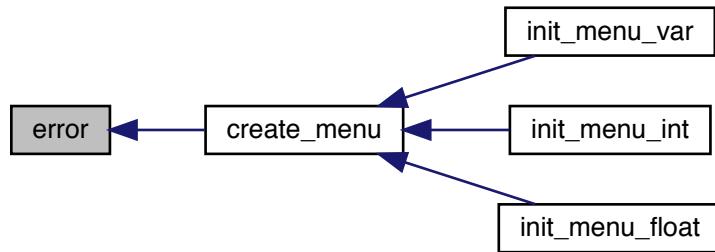
```

00021
00022     if(log_level>NONE)
00023         log_info("ERROR", error_message);
00024 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.21.3.3 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 (p. 321)

Parameters

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

Definition at line **31** of file **log.c**.

References **ERROR**, **log_level**, and **printf()**.

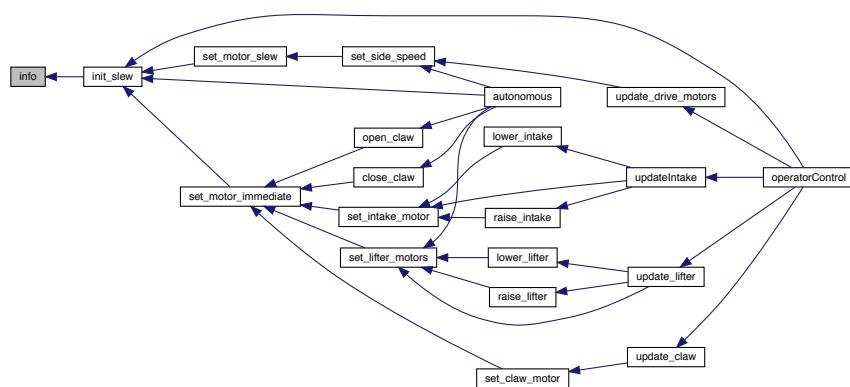
Referenced by **init_slew()**.

```
00031      {
00032      if(log_level>ERROR) {
00033          printf("[INFO]: %s\n", info_message);
00034      }
00035 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.21.3.4 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

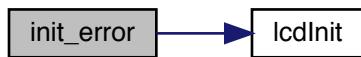
<i>use_lcd</i>	whether to use the lcd
<i>lcd</i>	the lcd

Definition at line **6** of file **log.c**.

References **lcdInit()**, and **log_lcd**.

```
00006
00007     if(use_lcd) {
00008         lcdInit(lcd);
00009         log_lcd = lcd;
00010     }
00011 }
```

Here is the call graph for this function:



5.21.3.5 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](#) (p. 321)

Author

Chris Jerrett

Date

9/10/17

Parameters

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

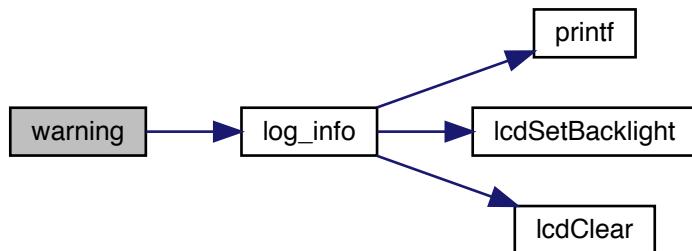
Definition at line **26** of file **log.c**.

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

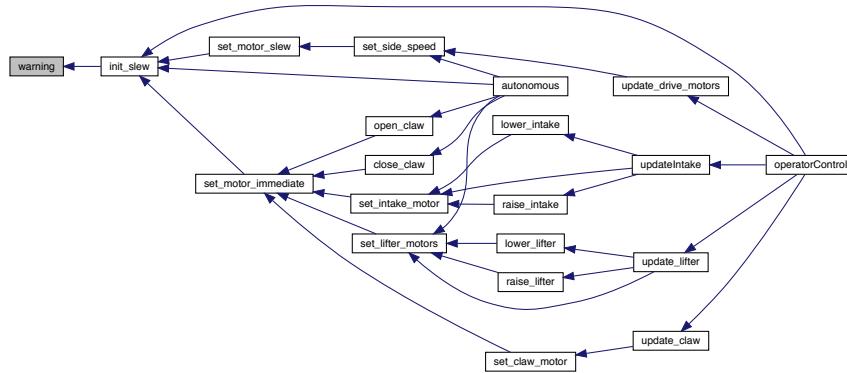
Referenced by [init_slew\(\)](#).

```
00026
00027     if(log_level>WARNING)
00028         log_info("WARNING", warning_message);
00029 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.22 log.h

```

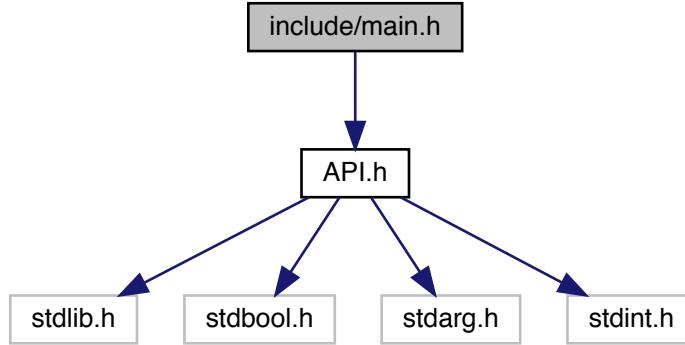
00001
00007 #ifndef _LOG_H_
00008 #define _LOG_H_
00009
00010 #include <API.h>
00011 #include "lcd.h"
00012
00019 #define NONE 0
00020
00027 #define ERROR 1
00028
00035 #define WARNING 2
00036
00042 #define INFO 3
00043
00050 #define DEBUG 4
00051
00060 void init_error(bool use_lcd, FILE *lcd);
00061
00070 void error(const char *error_message);
00071
00080 void warning(const char *warning_message);
00081
00089 void info(const char *info_message);
00090
00098 void debug(const char *debug_message);
00099
00100
00101 #endif

```

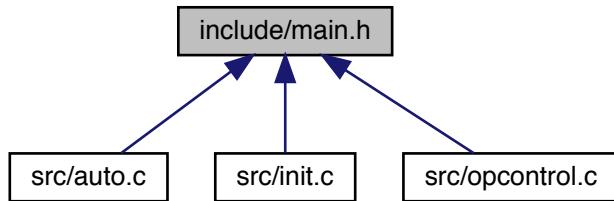
5.23 include/main.h File Reference

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**()

5.23.1 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.

Definition in file **main.h**.

5.23.2 Function Documentation

5.23.2.1 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.

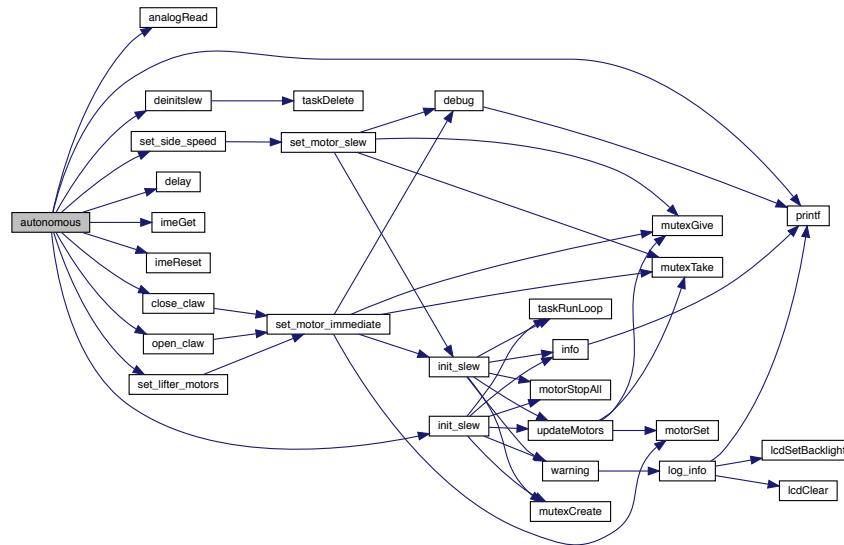
The autonomous task may exit, unlike **operatorControl()** (p. 175) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

Definition at line 30 of file **auto.c**.

References **analogRead()**, **BOTH**, **close_claw()**, **deinitSlew()**, **delay()**, **GOAL_HEIGHT**, **imeGet()**, **imeReset()**, **init_slew()**, **LIFTER**, **MID_LEFT_DRIVE**, **MID_RIGHT_DRIVE**, **open_claw()**, **printf()**, **set_lifter_motors()**, and **set_side_speed()**.

```
00030          {
00031     init_slew();
00032
00033     delay(10);
00034     printf("auto\n");
00035     //How far the left wheels have gone
00036     int counts_drive_left;
00037     //How far the right wheels have gone
00038     int counts_drive_right;
00039     //The average distance traveled forward
00040     int counts_drive;
00041
00042     //Reset the integrated motor controllers
00043     imeReset(MID_LEFT_DRIVE);
00044     imeReset(MID_RIGHT_DRIVE);
00045     //Set initial values for how far the wheels have gone
00046     imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00047     imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00048     counts_drive = counts_drive_left + counts_drive_right;
00049     counts_drive /= 2;
00050
00051     //Grab pre-load cone
00052     close_claw();
00053     delay(300);
00054
00055     //Raise the lifter
00056     while(analogRead(LIFTER) < GOAL_HEIGHT){
00057         set_lifter_motors(-127);
00058     }
00059     set_lifter_motors(0);
00060     //Drive towards the goal
00061     while(counts_drive < 530){
00062         set_side_speed(BOTH, 127);
00063         //Restablish the distance traveled
00064         imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00065         imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00066         counts_drive = counts_drive_left + counts_drive_right;
00067         counts_drive /= 2;
00068     }
00069     //Stop moving
00070     set_side_speed(BOTH, 0);
00071     delay(1000);
00072
00073     //Drop the cone on the goal
00074     open_claw();
00075     delay(1000);
00076     deinitSlew();
00077 }
```

Here is the call graph for this function:



5.23.2.2 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()** (p. 175) and **autonomous()** (p. 172) tasks will not start. An autonomous mode selection menu like the pre_auton() in other environments can be implemented in this task if desired.

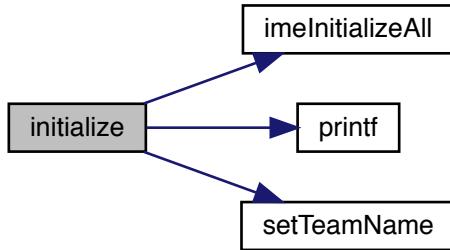
Definition at line 47 of file **init.c**.

References **imeInitializeAll()**, **printf()**, and **setTeamName()**.

```

00047      {
00048      int c = imeInitializeAll();
00049      setTeamName("9228A");
00050      printf("Counts : %d\n", c);
00051  }
```

Here is the call graph for this function:



5.23.2.3 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()** (p. 74)) and port states (**digitalWrite()** (p. 40)) of limit switches, push buttons, and solenoids. It can also safely configure a UART port (**uartOpen()**) but cannot set up an LCD (**lcdInit()** (p. 65)).

Definition at line **30** of file **init.c**.

References **watchdogInit()**.

```
00030     {
00031     watchdogInit () ;
00032 }
```

Here is the call graph for this function:



5.23.2.4 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()** (p. 39) or **taskDelayUntil()** (p. 82) is highly recommended to give other tasks (including system tasks such as updating LCDs) time to run.

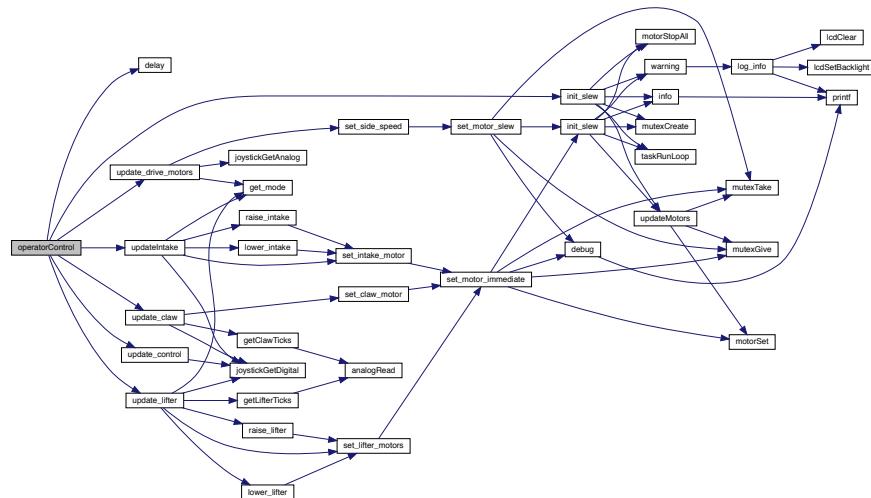
This task should never exit; it should end with some kind of infinite loop, even if empty.

Definition at line 40 of file **opcontrol.c**.

References **delay()**, **init_slew()**, **update_claw()**, **update_control()**, **update_drive_motors()**, **update_lifter()**, and **updateIntake()**.

```
00040          {
00041      init_slew();
00042      delay(10);
00043      while (1) {
00044          update_drive_motors();
00045          update_lifter();
00046          update_claw();
00047          updateIntake();
00048          update_control();
00049          delay(25);
00050      }
00051 }
```

Here is the call graph for this function:



5.24 main.h

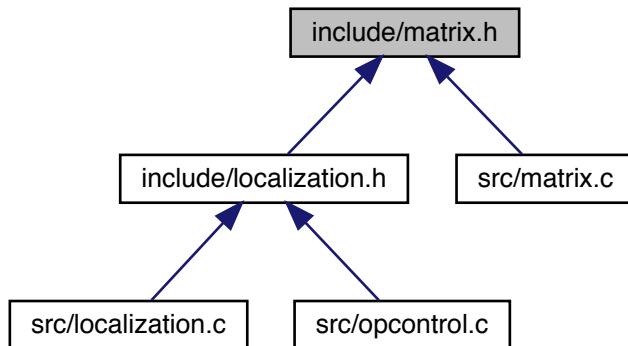
```
00001
00041 #ifndef MAIN_H_
00042
00043 // This prevents multiple inclusion, which isn't bad for this file but is good practice
00044 #define MAIN_H_
00045
00046 #include <API.h>
00047
00048 // Allow usage of this file in C++ programs
00049 #ifdef __cplusplus
00050 extern "C" {
00051 #endif
00052
00053 // #define AUTO_DEBUG
00054
00055 // A function prototype looks exactly like its declaration, but with a semicolon instead of
00056 // actual code. If a function does not match a prototype, compile errors will occur.
00057
00058 // Prototypes for initialization, operator control and autonomous
00059
00074 void autonomous();
00083 void initializeIO();
00097 void initialize();
00115 void operatorControl();
00116
00117 // End C++ export structure
00118 #ifdef __cplusplus
00119 }
00120 #endif
00121
00122 #endif
```

5.25 include/matrix.h File Reference

Various Matrix operations.

None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevalent, be sure to use the function to reclaim some of that memory.

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



Data Structures

- struct `_matrix`

TypeDefs

- `typedef struct _matrix matrix`

Functions

- `void assert (int assertion, const char *message)`
Asserts a condition is true.
- `matrix * copyMatrix (matrix *m)`
Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.
- `matrix * covarianceMatrix (matrix *m)`
returns the covariance of the matrix
- `matrix * dotDiagonalMatrix (matrix *a, matrix *b)`
performs a diagonal matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a $a \times height$ matrix of the cross product of each matrix along the diagonal.
- `matrix * dotProductMatrix (matrix *a, matrix *b)`
returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a $a \times height$ by $b \times height$ matrix of the cross product of each matrix.
- `void freeMatrix (matrix *m)`
Frees the resources of a matrix.
- `matrix * identityMatrix (int n)`
Returns an identity matrix of size n by n .
- `matrix * makeMatrix (int width, int height)`
Makes a matrix with a width and height parameters.
- `matrix * meanMatrix (matrix *m)`
Given an " m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.
- `matrix * multiplyMatrix (matrix *a, matrix *b)`
Given a two matrices, returns the multiplication of the two.
- `void printMatrix (matrix *m)`
Prints a matrix.
- `void rowSwap (matrix *a, int p, int q)`
swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q , provided that p and q are less than or equal to the height of matrix A and p and q are different values.
- `matrix * scaleMatrix (matrix *m, double value)`
scales a matrix.
- `double traceMatrix (matrix *m)`
Given an " m rows by n columns" matrix.
- `matrix * transposeMatrix (matrix *m)`
returns the transpose matrix.

5.25.1 Detailed Description

Various Matrix operations.

None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevalent, be sure to use the function to reclaim some of that memory.

Definition in file **matrix.h**.

5.25.2 Typedef Documentation

5.25.2.1 matrix

```
typedef struct _matrix matrix
```

A struct representing a matrix

5.25.3 Function Documentation

5.25.3.1 assert()

```
void assert (
    int assertion,
    const char * message )
```

Asserts a condition is true.

If the assertion is non-zero (i.e. true), then it returns. If the assertion is zero (i.e. false), then it displays the string and aborts the program. This is meant to act like Python's assert keyword.

Definition at line 14 of file **matrix.c**.

References **fprintf()**.

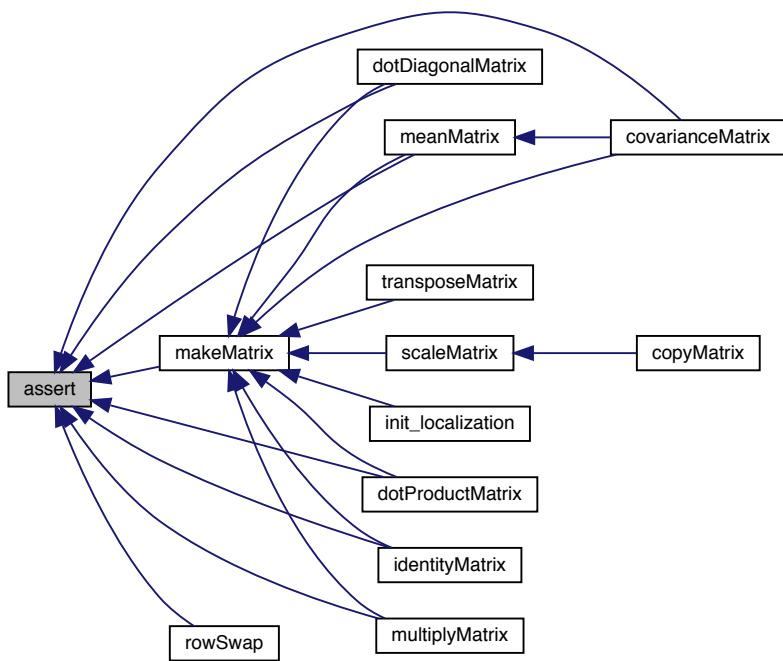
Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, and **rowSwap()**.

```
00014
00015     if (assertion == 0) {
00016         fprintf(stderr, "%s\n", message);
00017         exit(1);
00018     }
00019 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.3.2 copyMatrix()

```
matrix* copyMatrix (
    matrix * m )
```

Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.

Parameters

<i>m</i>	a pointer to the matrix
----------	-------------------------

Returns

a copied matrix

Definition at line **52** of file **matrix.c**.

References **scaleMatrix()**.

```
00052
00053     return scaleMatrix(m, 1);
00054 }
```

Here is the call graph for this function:

**5.25.3.3 covarianceMatrix()**

```
matrix* covarianceMatrix (
    matrix * m )
```

returns the covariance of the matrix

Parameters

<i>the</i>	matrix
------------	--------

Returns

a matrix with n row and n columns, where each element represents covariance of 2 columns.

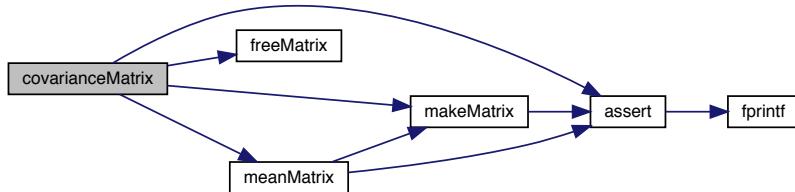
Definition at line **168** of file **matrix.c**.

References **assert()**, **_matrix::data**, **freeMatrix()**, **_matrix::height**, **makeMatrix()**, **meanMatrix()**, and **_matrix::width**.

```

00168     int i, j, k = 0;
00169     matrix* out;
00170     matrix* mean;
00171     double* ptrA;
00172     double* ptrB;
00173     double* ptrOut;
00174
00175
00176     assert(m->height > 1, "Height of matrix cannot be zero or one.");
00177
00178     mean = meanMatrix(m);
00179     out = makeMatrix(m->width, m->width);
00180     ptrOut = out->data;
00181
00182     for (i = 0; i < m->width; i++) {
00183         for (j = 0; j < m->width; j++) {
00184             ptrA = &m->data[i];
00185             ptrB = &m->data[j];
00186             *ptrOut = 0.0;
00187             for (k = 0; k < m->height; k++) {
00188                 *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
00189                 ptrA += m->width;
00190                 ptrB += m->width;
00191             }
00192             *ptrOut /= m->height - 1;
00193             ptrOut++;
00194         }
00195     }
00196
00197     freeMatrix(mean);
00198     return out;
00199 }
```

Here is the call graph for this function:



5.25.3.4 dotDiagonalMatrix()

```

matrix* dotDiagonalMatrix (
    matrix * a,
    matrix * b )
```

performs a diagonal matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a 1 by a->height matrix of the cross product of each matrix along the diagonal.

Dot product is essentially the sum-of-squares of two vectors.

If the second parameter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

<i>a</i>	the first matrix
<i>b</i>	the second matrix

Returns

the matrix result

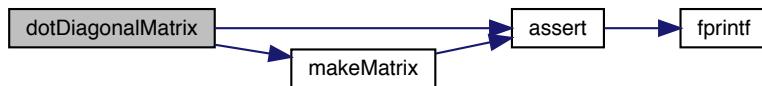
Definition at line 385 of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```

00385
00386     matrix* out;
00387     double* ptrOut;
00388     double* ptrA;
00389     double* ptrB;
00390     int i, j;
00391
00392     if (b != NULL) {
00393         assert(a->width == b->width && a->height == b->height, "Matrices must be of the same
00394         dimensionality.");
00395     }
00396
00397     // Are we computing the sum of squares of the same matrix?
00398     if (a == b || b == NULL) {
00399         b = a; // May not appear safe, but we can do this without risk of losing b.
00400     }
00401
00402     out = makeMatrix(1, a->height);
00403     ptrOut = out->data;
00404     ptrA = a->data;
00405     ptrB = b->data;
00406
00407     for (i = 0; i < a->height; i++) {
00408         *ptrOut = 0;
00409         for (j = 0; j < a->width; j++) {
00410             *ptrOut += *ptrA * *ptrB;
00411             ptrA++;
00412             ptrB++;
00413         }
00414         ptrOut++;
00415     }
00416
00417     return out;
00418 }
```

Here is the call graph for this function:



5.25.3.5 dotProductMatrix()

```
matrix* dotProductMatrix (
    matrix * a,
    matrix * b )
```

returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a $a \times b$ matrix of the cross product of each matrix.

Dot product is essentially the sum-of-squares of two vectors.

Also, if the second parameter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

<i>a</i>	the first matrix
<i>the</i>	second matrix

Returns

the result of the dot product

Definition at line 333 of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
00333
00334     matrix* out;
00335     double* ptrOut;
00336     double* ptrA;
00337     double* ptrB;
00338     int i, j, k;
00339
00340     if (b != NULL) {
00341         assert(a->width == b->width, "Matrices must be of the same dimensionality.");
00342     }
00343
00344     // Are we computing the sum of squares of the same matrix?
00345     if (a == b || b == NULL) {
00346         b = a; // May not appear safe, but we can do this without risk of losing b.
00347     }
00348
00349     out = makeMatrix(b->height, a->height);
00350     ptrOut = out->data;
00351
00352     for (i = 0; i < a->height; i++) {
00353         ptrB = b->data;
00354
00355         for (j = 0; j < b->height; j++) {
00356             ptrA = &a->data[ i * a->width ];
00357
00358             *ptrOut = 0;
00359             for (k = 0; k < a->width; k++) {
00360                 *ptrOut += *ptrA * *ptrB;
00361                 ptrA++;
00362                 ptrB++;
00363             }
00364             ptrOut++;
00365         }
00366     }
00367
00368     return out;
00369 }
```

Here is the call graph for this function:



5.25.3.6 freeMatrix()

```
void freeMatrix (
    matrix * m )
```

Frees the resources of a matrix.

Parameters

<i>the</i>	matrix to free
------------	----------------

Definition at line **60** of file **matrix.c**.

References `_matrix::data`.

Referenced by `covarianceMatrix()`.

```
00060
00061     if (m != NULL) {
00062         if (m->data != NULL) {
00063             free(m->data);
00064             m->data = NULL;
00065         }
00066         free(m);
00067     }
00068     return;
00069 }
```

Here is the caller graph for this function:



5.25.3.7 identityMatrix()

```
matrix* identityMatrix (
    int n )
```

Returns an identity matrix of size n by n.

Parameters

<i>n</i>	the input matrix. parameter.
<i>n</i>	the input matrix.

Returns

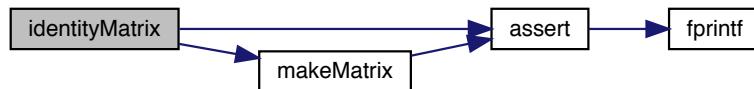
the identity matrix parameter.

Definition at line **94** of file **matrix.c**.

References **assert()**, **_matrix::data**, and **makeMatrix()**.

```
00094 {
00095     int i;
00096     matrix *out;
00097     double* ptr;
00098
00099     assert(n > 0, "Identity matrix must have value greater than zero.");
00100
00101     out = makeMatrix(n, n);
00102     ptr = out->data;
00103     for (i = 0; i < n; i++) {
00104         *ptr = 1.0;
00105         ptr += n + 1;
00106     }
00107
00108     return out;
00109 }
```

Here is the call graph for this function:



5.25.3.8 makeMatrix()

```
matrix* makeMatrix (
    int width,
    int height )
```

Makes a matrix with a width and height parameters.

Parameters

<i>width</i>	The width of the matrix
<i>height</i>	the height of the matrix

Returns

the new matrix

Definition at line 27 of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, and **_matrix::width**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **init_localization()**, **meanMatrix()**, **multiplyMatrix()**, **scaleMatrix()**, and **transposeMatrix()**.

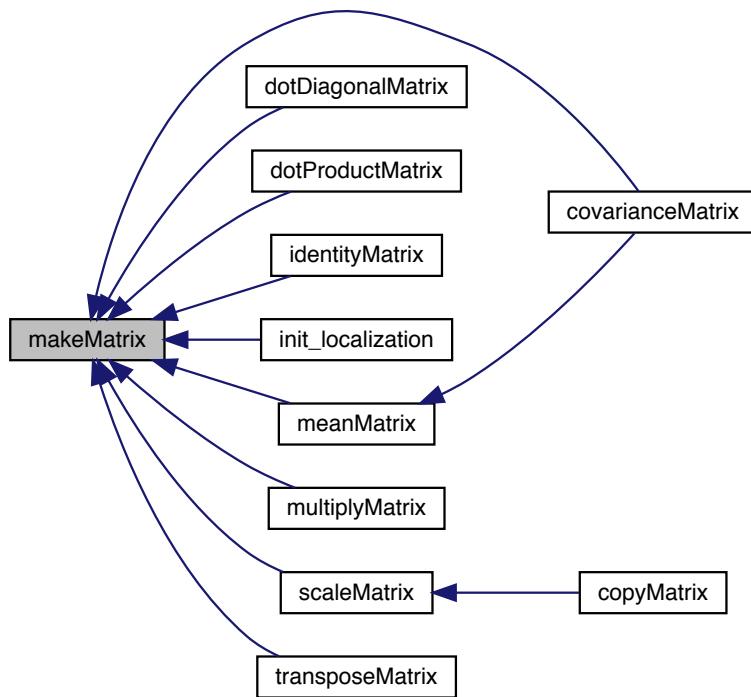
```

00027         {
00028     matrix* out;
00029     assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
00030     out = (matrix*) malloc(sizeof(matrix));
00031     assert(out != NULL, "Out of memory.");
00032     out->width = width;
00033     out->height = height;
00034     out->data = (double*) malloc(sizeof(double) * width * height);
00035     assert(out->data != NULL, "Out of memory.");
00036     memset(out->data, 0.0, width * height * sizeof(double));
00037     return out;
00038 }
00039 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.3.9 meanMatrix()

```
matrix* meanMatrix (
    matrix * m )
```

Given an "m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.

Returns

matrix with 1 row and n columns each element represents the mean of that full column.

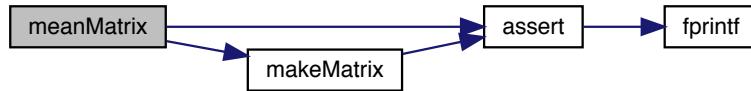
Definition at line **142** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **covarianceMatrix()**.

```
00142     int i, j;
00143     matrix* out;
00144
00145     assert(m->height > 0, "Height of matrix cannot be zero.");
00146
00147     out = makeMatrix(m->width, 1);
00148
00149     for (i = 0; i < m->width; i++) {
00150         double* ptr;
00151         out->data[i] = 0.0;
00152         ptr = &m->data[i];
00153         for (j = 0; j < m->height; j++) {
00154             out->data[i] += *ptr;
00155             ptr += out->width;
00156         }
00157         out->data[i] /= (double) m->height;
00158     }
00159     return out;
00160 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.3.10 multiplyMatrix()

```
matrix* multiplyMatrix (
    matrix * a,
    matrix * b )
```

Given a two matrices, returns the multiplication of the two.

Parameters

<i>a</i>	the first matrix
<i>b</i>	the seconf matrix return the result of the multiplication

Definition at line **230** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```

00230
00231     int i, j, k;
00232     matrix* out;
00233     double* ptrOut;
00234     double* ptrA;
00235     double* ptrB;
00236
00237     assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");
00238
00239     out = makeMatrix(b->width, a->height);
00240     ptrOut = out->data;
00241
00242     for (i = 0; i < a->height; i++) {
00243
00244         for (j = 0; j < b->width; j++) {
00245             ptrA = &a->data[ i * a->width ];
00246             ptrB = &b->data[ j ];
00247
00248             *ptrOut = 0;
00249             for (k = 0; k < a->width; k++) {
00250                 *ptrOut += *ptrA * *ptrB;
00251                 ptrA++;
00252                 ptrB += b->width;
00253             }
00254             ptrOut++;
00255         }
00256     }
00257
00258     return out;
00259 }
```

Here is the call graph for this function:



5.25.3.11 printMatrix()

```
void printMatrix (
    matrix * m )
```

Prints a matrix.

Parameters

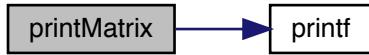
<i>the</i>	matrix
------------	--------

Definition at line 75 of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **printf()**, and **_matrix::width**.

```
00075             {
00076     int i, j;
00077     double* ptr = m->data;
00078     printf("%d %d\n", m->width, m->height);
00079     for (i = 0; i < m->height; i++) {
00080         for (j = 0; j < m->width; j++) {
00081             printf(" %9.6f", *(ptr++));
00082         }
00083         printf("\n");
00084     }
00085     return;
00086 }
```

Here is the call graph for this function:

**5.25.3.12 rowSwap()**

```
void rowSwap (
    matrix * a,
    int p,
    int q )
```

swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q, provided that p and q are less than or equal to the height of matrix A and p and q are different values.

Parameters

<i>the</i>	matrix to swap. This method changes the input matrix.
<i>the</i>	first row
<i>the</i>	second row

Definition at line 290 of file **matrix.c**.

References `assert()`, `_matrix::data`, `_matrix::height`, and `_matrix::width`.

```

00290             {
00291     int i;
00292     double temp;
00293     double* pRow;
00294     double* qRow;
00295
00296     assert(a->height > 2, "Matrix must have at least two rows to swap.");
00297     assert(p < a->height && q < a->height, "Values p and q must be less than the height of the matrix.");
00298
00299     // If p and q are equal, do nothing.
00300     if (p == q) {
00301         return;
00302     }
00303
00304     pRow = a->data + (p * a->width);
00305     qRow = a->data + (q * a->width);
00306
00307     // Swap!
00308     for (i = 0; i < a->width; i++) {
00309         temp = *pRow;
00310         *pRow = *qRow;
00311         *qRow = temp;
00312         pRow++;
00313         qRow++;
00314     }
00315
00316     return;
00317 }
```

Here is the call graph for this function:



5.25.3.13 scaleMatrix()

```
matrix* scaleMatrix (
    matrix * m,
    double value )
```

scales a matrix.

Parameters

<i>m</i>	the matrix to scale
<i>the</i>	value to scale by

Returns

a new matrix where each element in the input matrix is multiplied by the scalar value

Definition at line **268** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **copyMatrix()**.

```
00268      int i, elements = m->width * m->height;
00269      matrix* out = makeMatrix(m->width, m->height);
00270      double* ptrM = m->data;
00271      double* ptrOut = out->data;
00272
00273      for (i = 0; i < elements; i++) {
00274          *(ptrOut++) = *(ptrM++) * value;
00275      }
00276
00277      return out;
00278  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.3.14 traceMatrix()

```
double traceMatrix (
    matrix * m )
```

Given an "m rows by n columns" matrix.

Returns

the sum of the elements along the diagonal.

Given an "m rows by n columns" matrix.

Returns

the sum of the elements along the diagonal.

Definition at line **116** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
00116     int i;
00117     int size;
00118     double* ptr = m->data;
00119     double sum = 0.0;
00120
00121     if (m->height < m->width) {
00122         size = m->height;
00123     }
00124     else {
00125         size = m->width;
00126     }
00127
00128     for (i = 0; i < size; i++) {
00129         sum += *ptr;
00130         ptr += m->width + 1;
00131     }
00132
00133     return sum;
00134 }
```

5.25.3.15 transposeMatrix()

```
matrix* transposeMatrix (
    matrix * m )
```

returns the transpose matrix.

Parameters

<i>the</i>	matrix to transpose.
------------	----------------------

Returns

the transposed matrix.

Definition at line **206** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```

00206     {
00207     matrix* out = makeMatrix(m->height, m->width);
00208     double* ptrM = m->data;
00209     int i, j;
00210
00211     for (i = 0; i < m->height; i++) {
00212         double* ptrOut;
00213         ptrOut = &out->data[i];
00214         for (j = 0; j < m->width; j++) {
00215             *ptrOut = *ptrM;
00216             ptrM++;
00217             ptrOut += out->width;
00218         }
00219     }
00220
00221     return out;
00222 }
```

Here is the call graph for this function:

**5.26 matrix.h**

```

00001
00008 #ifndef _MATRIX_H_
00009 #define _MATRIX_H_
00010
00014 typedef struct _matrix {
00015     int height;
00016     int width;
00017     double* data;
00018 } matrix;
00019
00028 void assert(int assertion, const char* message);
00029
00033 matrix* makeMatrix(int width, int height);
00034
00042 matrix* copyMatrix(matrix* m);
00043
00048 void freeMatrix(matrix* m);
00049
00054 void printMatrix(matrix* m);
00055
00056 //=====
00057 // Basic Matrix operations
00058 //=====
00064 matrix* identityMatrix(int n);
00065
00071 double traceMatrix(matrix* m);
00072
```

```

00078 matrix* transposeMatrix(matrix* m);
00079
00085 matrix* meanMatrix(matrix* m);
00086
00093 matrix* multiplyMatrix(matrix* a, matrix* b);
00094
00102 matrix* scaleMatrix(matrix* m, double value);
00103
00109 matrix* covarianceMatrix(matrix* m);
00110
00120 void rowSwap(matrix* a, int p, int q);
00135 matrix* dotProductMatrix(matrix* a, matrix* b);
00136
00151 matrix* dotDiagonalMatrix(matrix* a, matrix* b);
00152
00153 #endif

```

5.27 include/menu.h File Reference

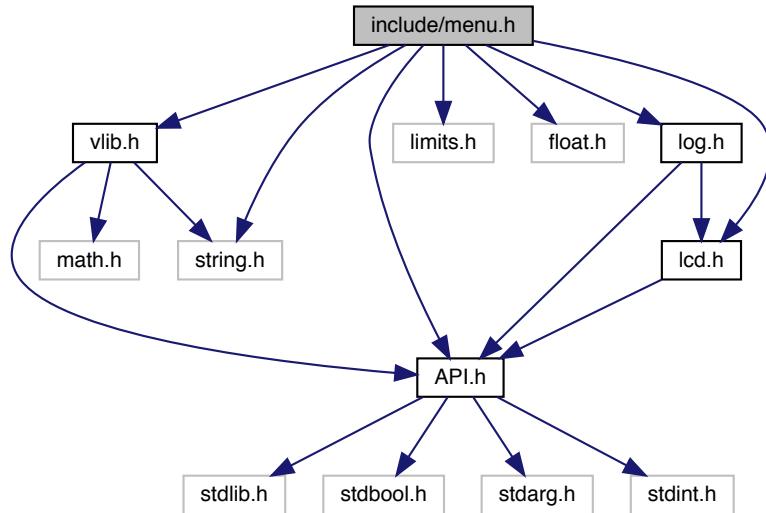
Contains menu functionality and abstraction.

```

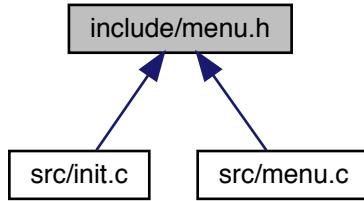
#include "lcd.h"
#include "API.h"
#include <string.h>
#include <limits.h>
#include <float.h>
#include <vlib.h>
#include "log.h"

```

Include dependency graph for menu.h:



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



Data Structures

- struct **menu_t**

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

Typedefs

- typedef struct **menu_t** **menu_t**

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

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, const 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, const 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, const char *prompt, char *options,...)

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

5.27.1 Detailed Description

Contains menu functionality and abstraction.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **menu.h**.

5.27.2 Typedef Documentation

5.27.2.1 menu_t

```
typedef struct menu_t menu_t
```

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via `denint_menu`.

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#) (p. 196)
[menu_t](#) (p. 10)
[create_menu](#) (p. 347)
[init_menu](#)
[display_menu](#) (p. 348)
[menu_type](#) (p. 198)
[denint_menu](#) (p. 348)

5.27.3 Enumeration Type Documentation

5.27.3.1 menu_type

```
enum menu_type
```

Represents the different types of menus.

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#) (p. 196)
[menu_t](#) (p. 10)
[create_menu](#) (p. 201)
[init_menu](#)
[display_menu](#) (p. 202)
[menu_type](#) (p. 198)

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 30 of file [menu.h](#).

```
00030
00031     {
00032     INT_TYPE,
00033     FLOAT_TYPE,
00034     STRING_TYPE
00035     };
00036 }
```

5.27.4 Function Documentation

5.27.4.1 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

5.27.4.2 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

5.27.4.3 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 **101** of file **menu.c**.

References **menu_t::options**, and **menu_t::prompt**.

```
00101     {  
00102     free(menu->prompt);  
00103     if(menu->options != NULL) free(menu->options);  
00104     free(menu);  
00105 }
```

5.27.4.4 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 (p. 198)

Author

Chris Jerrett

Date

9/8/17

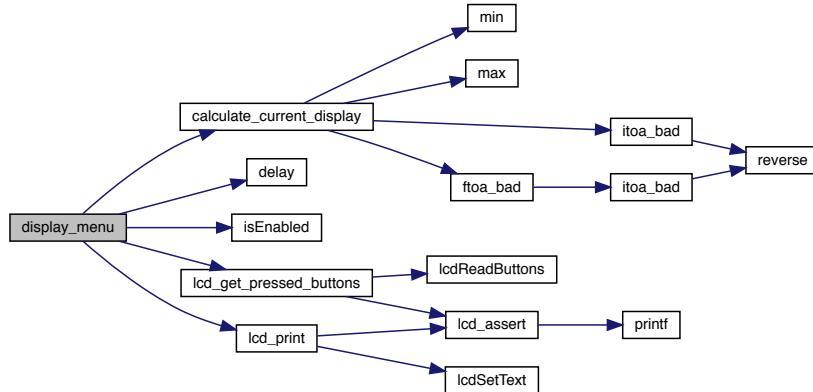
Definition at line 83 of file **menu.c**.

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

```

00083     {
00084     lcd_print(TOP_ROW, menu->prompt);
00085     //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
00086     while(lcd_get_pressed_buttons().middle == RELEASED && !isEnabled()) {
00087         char val[16];
00088         calculate_current_display(val, menu);
00089
00090         if(lcd_get_pressed_buttons().right == PRESSED) {
00091             menu->current += 1;
00092         }
00093         if(lcd_get_pressed_buttons().left == PRESSED) {
00094             menu->current -= 1;
00095         }
00096         delay(500);
00097     }
00098     return menu->current;
00099 }
```

Here is the call graph for this function:



5.27.4.5 init_menu_float()

```

menu_t* init_menu_float (
    enum menu_type type,
    float min,
    float max,
    float step,
    const 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 (p. 198)

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

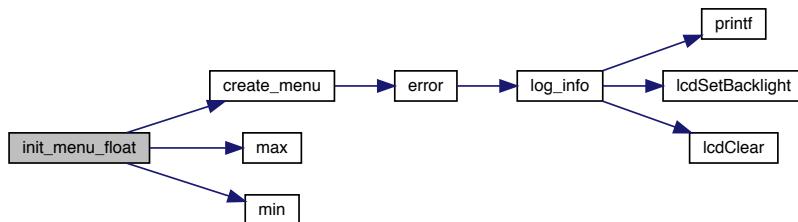
Definition at line 48 of file **menu.c**.

References **create_menu()**, **max()**, **menu_t::max_f**, **min()**, **menu_t::min_f**, and **menu_t::step_f**.

```

00048     menu_t* menu = create_menu(type, prompt);
00049     menu->min_f = min;
00050     menu->max_f = max;
00051     menu->step_f = step;
00052     return menu;
00053 }
```

Here is the call graph for this function:



5.27.4.6 init_menu_int()

```
menu_t* init_menu_int (
    enum menu_type type,
    int min,
    int max,
    int step,
    const 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](#) (p. 198)

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

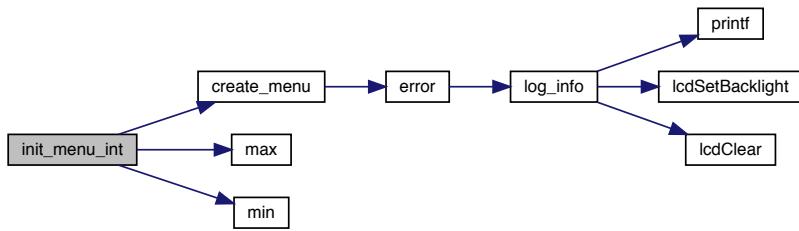
9/8/17

Definition at line **40** of file **menu.c**.

References [create_menu\(\)](#), [max\(\)](#), [menu_t::max](#), [min\(\)](#), [menu_t::min](#), and [menu_t::step](#).

```
00040
00041     menu_t* menu = create_menu(type, prompt);
00042     menu->min = min;
00043     menu->max = max;
00044     menu->step = step;
00045     return menu;
00046 }
```

Here is the call graph for this function:



5.27.4.7 init_menu_var()

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

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

Parameters

<code>type</code>	the type of menu
-------------------	------------------

See also

[menu_type](#) (p. 198)

Parameters

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

Author

Chris Jerrett

Date

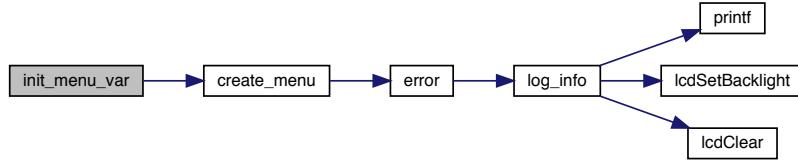
9/8/17

Definition at line 26 of file **menu.c**.References **create_menu()**, **menu_t::length**, and **menu_t::options**.

```

00026
00027     menu_t* menu = create_menu(type, prompt);
00028     va_list values;
00029     char **options_array = (char**)calloc(sizeof(char*), nums);
00030     va_start(values, options);
00031     for(unsigned int i = 0; i < nums; i++){
00032         options_array[i] = va_arg(values, char*);
00033     }
00034     va_end(values);
00035     menu->options = options_array;
00036     menu->length = nums;
00037     return menu;
00038 }
```

Here is the call graph for this function:



5.28 menu.h

```

00001
00008 #ifndef _MENU_H_
00009 #define _MENU_H_
00010
00011 #include "lcd.h"
00012 #include "API.h"
00013 #include <string.h>
00014 #include <limits.h>
00015 #include <float.h>
00016 #include <vlib.h>
00017 #include "log.h"
00018
00030 enum menu_type {
00037     INT_TYPE,
00044     FLOAT_TYPE,
00050     STRING_TYPE
00051 };
00052
00066 typedef struct menu_t{
00072     enum menu_type type;
00073     char **options;
00080
00086     unsigned int length;
00087     int min;
00095 }
```

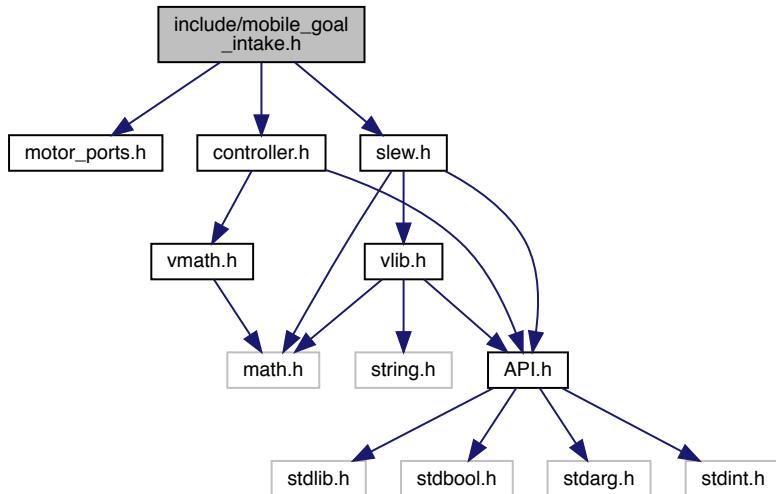
```

00102     int max;
00103     int step;
00110
00118     float min_f;
00119
00126     float max_f;
00127
00134     float step_f;
00140     int current;
00147     char *prompt;
00148 } menu_t;
00149
00156 static menu_t* create_menu(enum menu_type type, const char *prompt);
00157
00170 menu_t* init_menu_var(enum menu_type type, unsigned int nums, const char *prompt, char*
options,...);
00171
00185 menu_t* init_menu_int(enum menu_type type, int min, int max, int step, const char*
prompt);
00186
00200 menu_t* init_menu_float(enum menu_type type, float min, float max, float step, const char*
prompt);
00201
00210 static void calculate_current_display(char* rtn, menu_t *menu);
00211
00222 int display_menu(menu_t *menu);
00223
00233 void denint_menu(menu_t *menu);
00234
00235 #endif

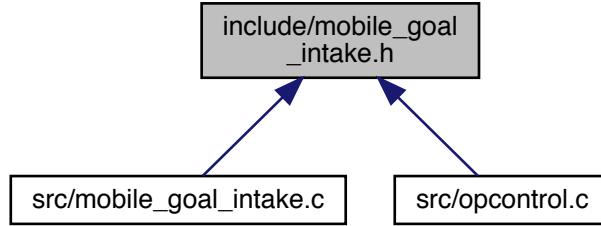
```

5.29 include/mobile_goal_intake.h File Reference

```
#include "motor_ports.h"
#include "controller.h"
#include "slew.h"
Include dependency graph for mobile_goal_intake.h:
```



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



Functions

- **void updateIntake ()**
updates the mobile goal intake in teleop.

5.29.1 Function Documentation

5.29.1.1 updateIntake()

```
void updateIntake ( )
```

updates the mobile goal intake in teleop.

Definition at line 16 of file **mobile_goal_intake.c**.

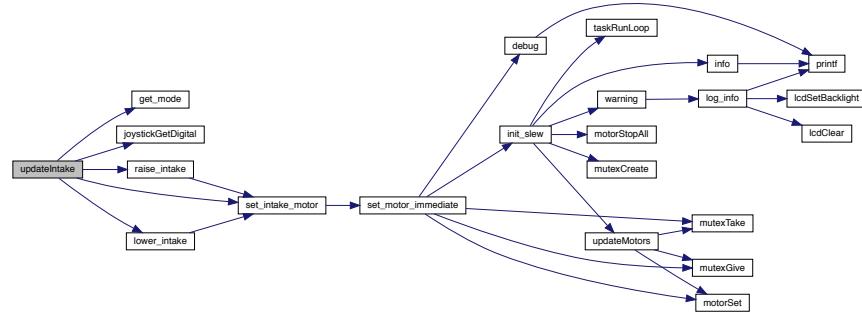
References **get_mode()**, **JOY_DOWN**, **JOY_UP**, **joystickGetDigital()**, **lower_intake()**, **MAIN_CONTROLLER_MODE**, **MASTER**, **PARTNER**, **PARTNER_CONTROLLER_MODE**, **raise_intake()**, and **set_intake_motor()**.

Referenced by **operatorControl()**.

```

00016      {
00017      if(joystickGetDigital(MASTER, 7, JOY_UP) && (get_mode() ==
00018          MAIN_CONTROLLER_MODE)
00018          || joystickGetDigital(PARTNER, 6, JOY_UP) && get_mode() ==
00019              PARTNER_CONTROLLER_MODE) {
00020              raise_intake();
00021      } else if(joystickGetDigital(MASTER, 7, JOY_DOWN) && (get_mode() ==
00021          MAIN_CONTROLLER_MODE)
00022          || joystickGetDigital(PARTNER, 6, JOY_DOWN) && get_mode() ==
00022              PARTNER_CONTROLLER_MODE){
00023              lower_intake();
00024      }
00025      else set_intake_motor(0);
00026  }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.30 mobile_goal_intake.h

```

00001 #ifndef _MOBLE_GOAL_INTAKE_
00002 #define _MOBLE_GOAL_INTAKE_
00003
00004 #include "motor_ports.h"
00005 #include "controller.h"
00006 #include "slew.h"
00007
00011 void updateIntake();
00012
00013 #endif

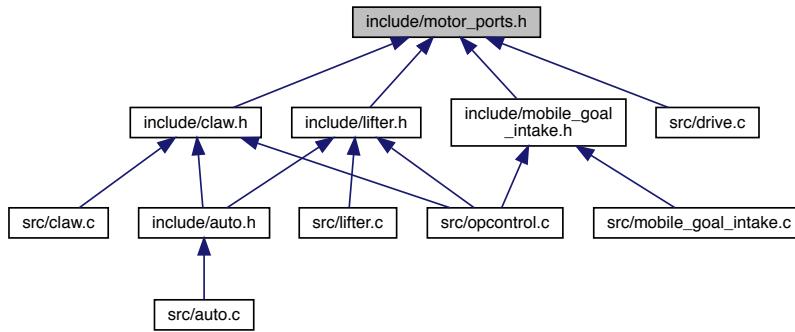
```

5.31 include/motor_ports.h File Reference

The motor port definitions

Macros for the different motors ports.

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



Macros

- #define **_MOTOR_PORTS_H_**
- #define **CLAW_MOTOR** 10
- #define **INTAKE_MOTOR** 1
- #define **MAX_SPEED** 127

Max motor speed.
- #define **MIN_SPEED** -127
- #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** 7

Front left drive motor of robot base.
- #define **MOTOR_FRONT_RIGHT** 2

Front right drive motor of robot base.
- #define **MOTOR_LIFT_BOTTOM_LEFT** 9
- #define **MOTOR_LIFT_BOTTOM_RIGHT** 8
- #define **MOTOR_LIFT_TOP_LEFT** 9
- #define **MOTOR_LIFT_TOP_RIGHT** 8
- #define **MOTOR_MIDDLE_LEFT** 6

Middle left drive motor of robot base.
- #define **MOTOR_MIDDLE_RIGHT** 3

Middle right drive motor of robot base.

5.31.1 Detailed Description

The motor port definitions

Macros for the different motors ports.

Definition in file **motor_ports.h**.

5.31.2 Macro Definition Documentation

5.31.2.1 _MOTOR_PORTS_H_

```
#define _MOTOR_PORTS_H_
```

Definition at line **7** of file **motor_ports.h**.

5.31.2.2 CLAW_MOTOR

```
#define CLAW_MOTOR 10
```

Definition at line **61** of file **motor_ports.h**.

Referenced by **close_claw()**, **open_claw()**, and **set_claw_motor()**.

5.31.2.3 INTAKE_MOTOR

```
#define INTAKE_MOTOR 1
```

Definition at line **62** of file **motor_ports.h**.

Referenced by **set_intake_motor()**.

5.31.2.4 MAX_SPEED

```
#define MAX_SPEED 127
```

Max motor speed.

Definition at line **12** of file **motor_ports.h**.

Referenced by **raise_lifter()**.

5.31.2.5 MIN_SPEED

```
#define MIN_SPEED -127
```

Definition at line **13** of file **motor_ports.h**.

Referenced by **lower_lifter()**.

5.31.2.6 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 **54** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

5.31.2.7 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 **48** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

5.31.2.8 MOTOR_FRONT_LEFT

```
#define MOTOR_FRONT_LEFT 7
```

Front left drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **27** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

5.31.2.9 MOTOR_FRONT_RIGHT

```
#define MOTOR_FRONT_RIGHT 2
```

Front right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **20** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

5.31.2.10 MOTOR_LIFT_BOTTOM_LEFT

```
#define MOTOR_LIFT_BOTTOM_LEFT 9
```

Definition at line **57** of file **motor_ports.h**.

5.31.2.11 MOTOR_LIFT_BOTTOM_RIGHT

```
#define MOTOR_LIFT_BOTTOM_RIGHT 8
```

Definition at line **56** of file **motor_ports.h**.

5.31.2.12 MOTOR_LIFT_TOP_LEFT

```
#define MOTOR_LIFT_TOP_LEFT 9
```

Definition at line **59** of file **motor_ports.h**.

Referenced by **set_lifter_motors()**.

5.31.2.13 MOTOR_LIFT_TOP_RIGHT

```
#define MOTOR_LIFT_TOP_RIGHT 8
```

Definition at line **58** of file **motor_ports.h**.

Referenced by **set_lifter_motors()**.

5.31.2.14 MOTOR_MIDDLE_LEFT

```
#define MOTOR_MIDDLE_LEFT 6
```

Middle left drive motor of robot base.

Date

9/7/2017

Author

Christian Desimone

Definition at line **41** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

5.31.2.15 MOTOR_MIDDLE_RIGHT

```
#define MOTOR_MIDDLE_RIGHT 3
```

Middle right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **34** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

5.32 motor_ports.h

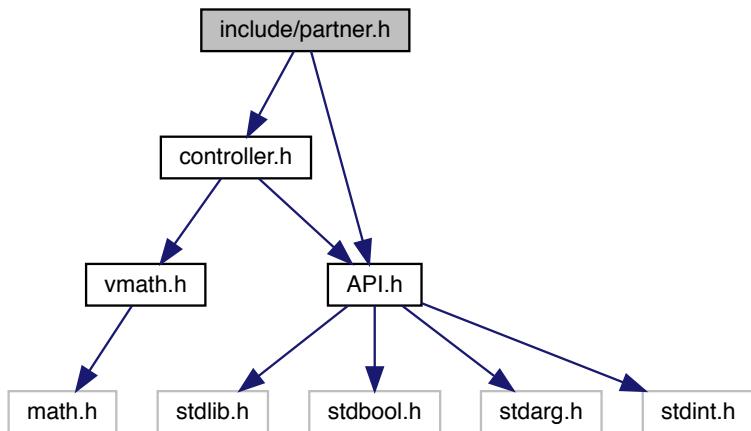
```

00001
00006 #ifndef _MOTOT_PORTS_H_
00007 #define _MOTOT_PORTS_H_
00008
00012 #define MAX_SPEED 127
00013 #define MIN_SPEED -127
00014
00020 #define MOTOR_FRONT_RIGHT 2
00021
00027 #define MOTOR_FRONT_LEFT 7
00028
00034 #define MOTOR_MIDDLE_RIGHT 3
00035
00041 #define MOTOR_MIDDLE_LEFT 6
00042
00048 #define MOTOR_BACK_RIGHT 4
00049
00054 #define MOTOR_BACK_LEFT 5
00055
00056 #define MOTOR_LIFT_BOTTOM_RIGHT 8
00057 #define MOTOR_LIFT_BOTTOM_LEFT 9
00058 #define MOTOR_LIFT_TOP_RIGHT 8
00059 #define MOTOR_LIFT_TOP_LEFT 9
00060
00061 #define CLAW_MOTOR 10
00062 #define INTAKE_MOTOR 1
00063
00064 #endif

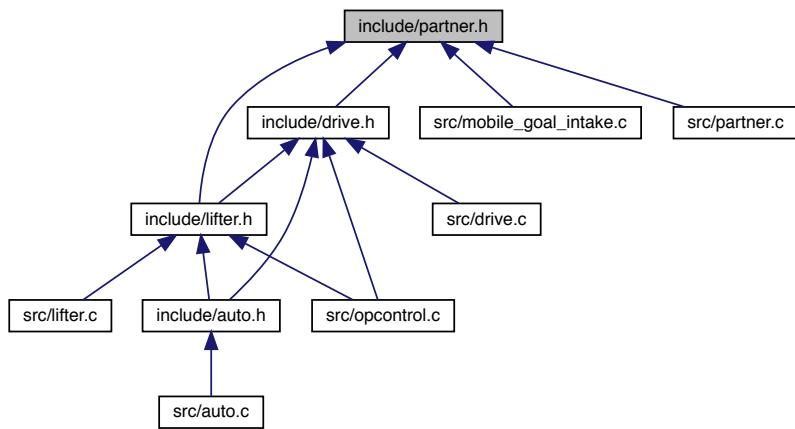
```

5.33 include/partner.h File Reference

```
#include "controller.h"
#include "API.h"
Include dependency graph for partner.h:
```



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



Enumerations

- enum **CONTROL_MODE** { **MAIN_CONTROLLER_MODE**, **PARTNER_CONTROLLER_MODE** }

Functions

- enum **CONTROL_MODE** **get_mode** ()
- void **update_control** ()

5.33.1 Enumeration Type Documentation

5.33.1.1 CONTROL_MODE

enum **CONTROL_MODE**

Enumerator

MAIN_CONTROLLER_MODE	
PARTNER_CONTROLLER_MODE	

Definition at line 7 of file **partner.h**.

```

00007      {
00008      MAIN_CONTROLLER_MODE,
00009      PARTNER_CONTROLLER_MODE
0010  };
  
```

5.33.2 Function Documentation

5.33.2.1 get_mode()

```
enum CONTROLL_MODE get_mode ( )
```

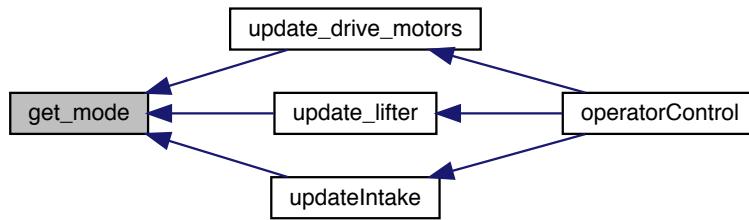
Definition at line 5 of file **partner.c**.

References **mode**.

Referenced by **update_drive_motors()**, **update_lifter()**, and **updateIntake()**.

```
00005
00006     return mode;
00007 }
```

Here is the caller graph for this function:



5.33.2.2 update_control()

```
void update_control ( )
```

Definition at line 9 of file **partner.c**.

References **JOY_LEFT**, **JOY_RIGHT**, **joystickGetDigital()**, **MAIN_CONTROLLER_MODE**, **mode**, **PARTNER**, and **PARTNER_CONTROLLER_MODE**.

Referenced by **operatorControl()**.

```
00009
00010     if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
00011         mode = MAIN_CONTROLLER_MODE;
00012     } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
00013         mode = PARTNER_CONTROLLER_MODE;
00014     }
00015 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

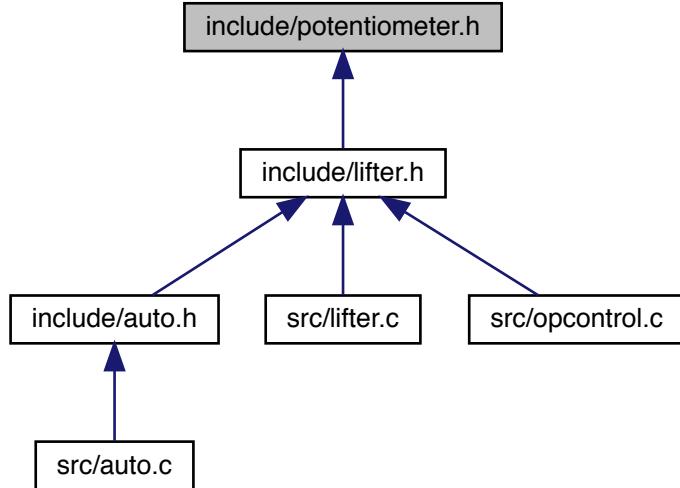


5.34 partner.h

```
00001 #ifndef _PARTNER_H_
00002 #define _PARTNER_H_
00003
00004 #include "controller.h"
00005 #include "API.h"
00006
00007 enum CONTROL_MODE {
00008     MAIN_CONTROLLER_MODE,
00009     PARTNER_CONTROLLER_MODE
00010 };
00011
00012 void update_control();
00013
00014 enum CONTROL_MODE get_mode();
00015
00016 #endif
```

5.35 include/potentiometer.h File Reference

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



Macros

- `#define DEG_MAX 250.0`
- `#define TICK_MAX 4095.0`

5.35.1 Macro Definition Documentation

5.35.1.1 DEG_MAX

```
#define DEG_MAX 250.0
```

Definition at line **5** of file **potentiometer.h**.

Referenced by **lifterPotentiometerToDegree()**.

5.35.1.2 TICK_MAX

```
#define TICK_MAX 4095.0
```

Definition at line 4 of file **potentiometer.h**.

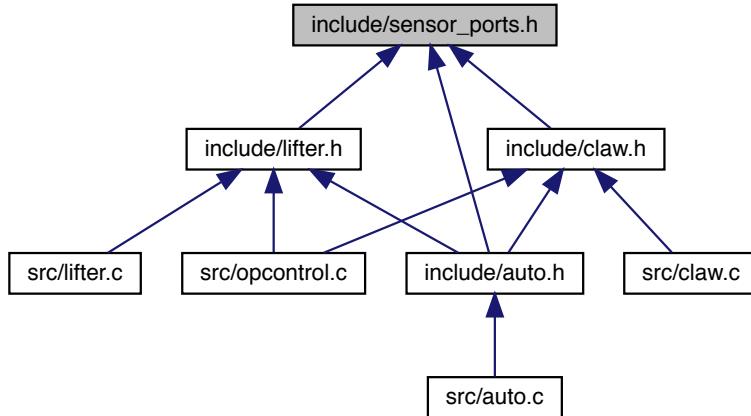
Referenced by **lifterPotentiometerToDegree()**.

5.36 potentiometer.h

```
00001 #ifndef _POTENTIOMETER_H_
00002 #define _POTENTIOMETER_H_
00003
00004 #define TICK_MAX 4095.0
00005 #define DEG_MAX 250.0
00006
00007 #endif
```

5.37 include/sensor_ports.h File Reference

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



Macros

- `#define CLAW_POT 1`
- `#define IME_FRONT_RIGHT 0`

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

- `#define LIFTER 2`

5.37.1 Macro Definition Documentation

5.37.1.1 CLAW_POT

```
#define CLAW_POT 1
```

Definition at line **21** of file **sensor_ports.h**.

Referenced by **getClawTicks()**.

5.37.1.2 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 (p. 278)

Author

Christian Desimone

Date

9/7/2017

Definition at line **18** of file **sensor_ports.h**.

5.37.1.3 LIFTER

```
#define LIFTER 2
```

Definition at line **20** of file **sensor_ports.h**.

Referenced by **autonomous()**, and **getLifterTicks()**.

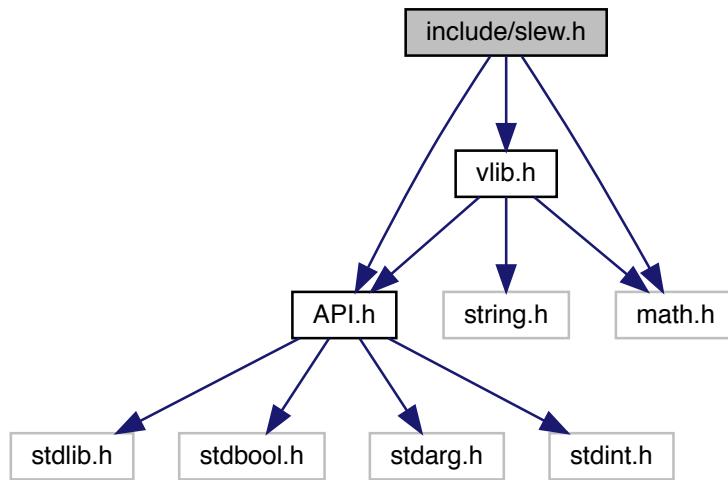
5.38 sensor_ports.h

```
00001
00008 #ifndef _PORTS_H_
00009 #define _PORTS_H_
00010
00018 #define IME_FRONT_RIGHT 0
00019 //#define POTENTIOMETER_PORT 2
00020 #define LIFTER 2
00021 #define CLAW_POT 1
00022
00023 #endif
```

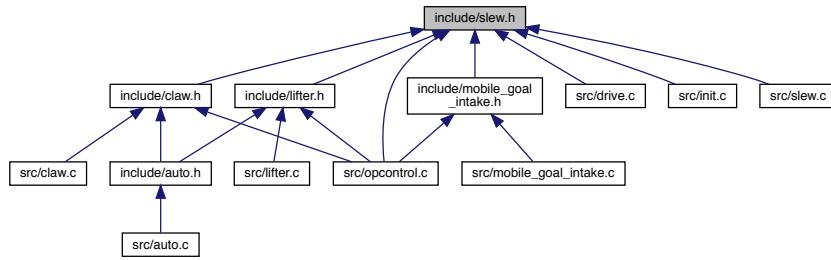
5.39 include/slew.h File Reference

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 1`
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_immediate(int motor, int speed)`
- `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.

5.39.1 Detailed Description

Contains the slew rate controller wrapper for the motors.

Author

Chris Jerrett

Date

9/14/17

Definition in file **slew.h**.

5.39.2 Macro Definition Documentation

5.39.2.1 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**.

5.39.2.2 RAMP_PROPORTION

```
#define RAMP_PROPORTION 1
```

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**.

5.39.2.3 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**.

5.39.3 Function Documentation

5.39.3.1 deinitSlew()

```
void deinitSlew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line **43** of file **slew.c**.

References **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **slew**, and **taskDelete()**.

Referenced by **autonomous()**.

```
00043     {
00044     taskDelete(slew);
00045     memset(motors_set_speeds, 0, sizeof(int) * 10);
00046     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00047     initialized = false;
00048 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

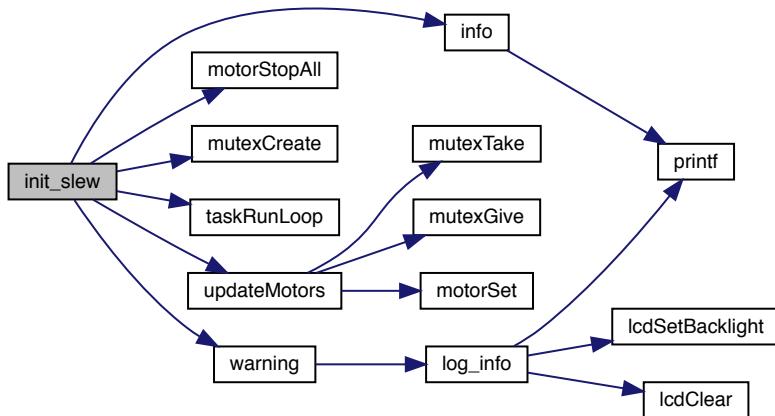
Definition at line **30** of file **slew.c**.

References **info()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **motorStopAll()**, **mutexCreate()**, **slew**, **speeds_mutex**, **taskRunLoop()**, **updateMotors()**, and **warning()**.

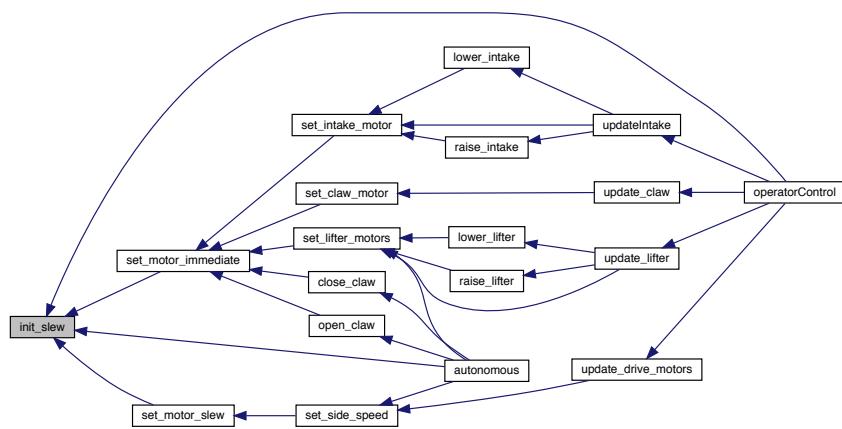
Referenced by **autonomous()**, **operatorControl()**, **set_motor_immediate()**, and **set_motor_slew()**.

```
00030             {
00031     if(initialized) {
00032         warning("Trying to init already init slew");
00033     }
00034     memset(motors_set_speeds, 0, sizeof(int) * 10);
00035     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00036     motorStopAll();
00037     info("Did Init Slew");
00038     speeds_mutex = mutexCreate();
00039     slew = taskRunLoop(updateMotors, 100);
00040     initialized = true;
00041 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.3 `set_motor_immediate()`

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

Definition at line **60** of file **slew.c**.

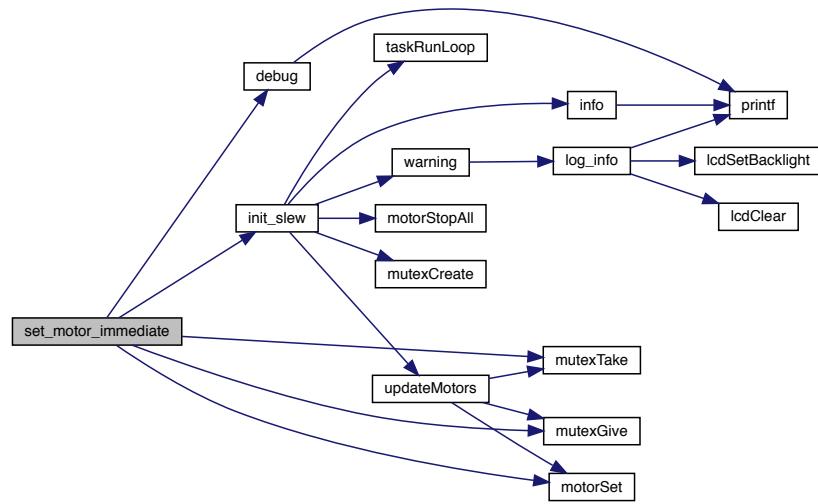
References `debug()`, `init_slew()`, `initialized`, `motors_curr_speeds`, `motors_set_speeds`, `motorSet()`, `mutexGive()`, `mutexTake()`, and `speeds_mutex`.

Referenced by `close_claw()`, `open_claw()`, `set_claw_motor()`, `set_intake_motor()`, and `set_lifter_motors()`.

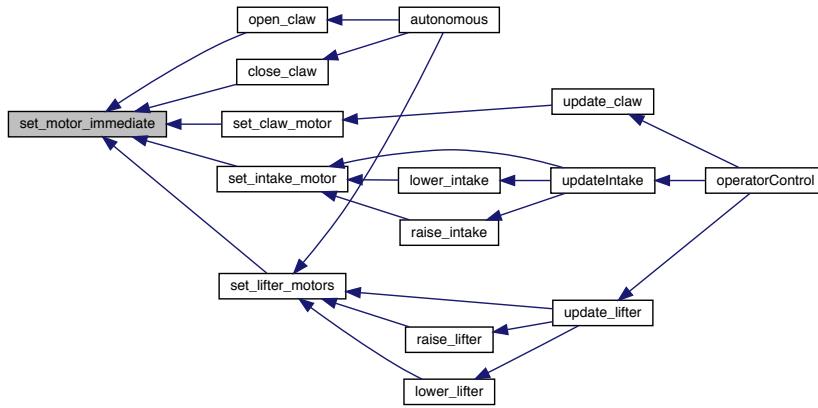
```

00060
00061     if(!initialized) {
00062         debug("Slew Not Initialized! Initializing");
00063         init_slew();
00064     }
00065     motorSet(motor, speed);
00066     mutexTake(speeds_mutex, 10);
00067     motors_curr_speeds[motor-1] = speed;
00068     motors_set_speeds[motor-1] = speed;
00069     mutexGive(speeds_mutex);
00070 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.4 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 **50** of file **slew.c**.

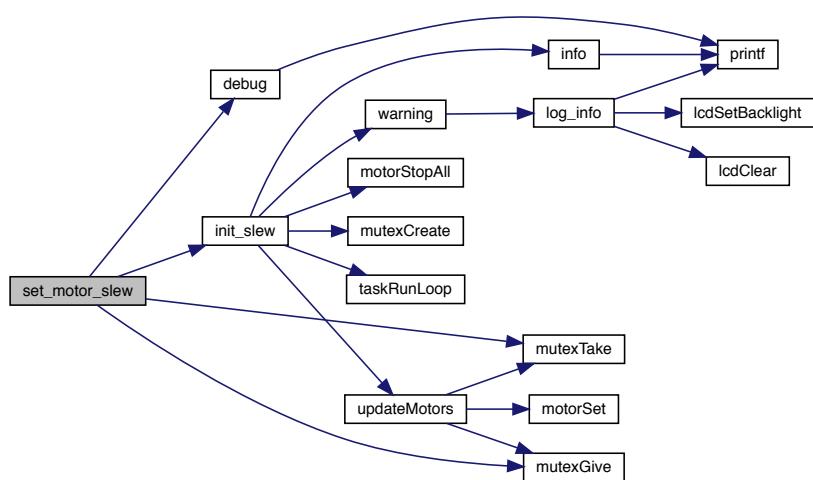
References **debug()**, **init_slew()**, **initialized**, **motors_set_speeds**, **mutexGive()**, **mutexTake()**, and **speeds_mutex**.

Referenced by **set_side_speed()**.

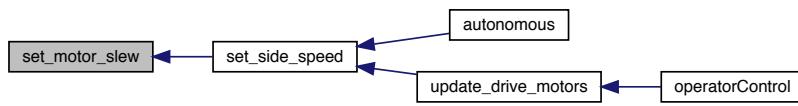
```

00050
00051     if(!initialized) {
00052         debug("Slew Not Initialized! Initializing");
00053         init_slew();
00054     }
00055     mutexTake(speeds_mutex, 10);
00056     motors_set_speeds[motor-1] = speed;
00057     mutexGive(speeds_mutex);
00058 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.5 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 13 of file **slew.c**.

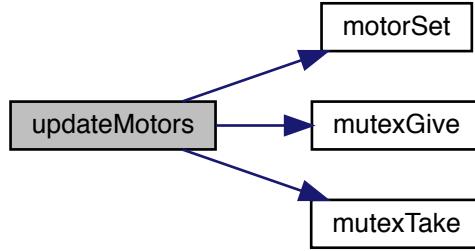
References **motors_curr_speeds**, **motors_set_speeds**, **motorSet()**, **mutexGive()**, **mutexTake()**, and **speeds_mutex**.

Referenced by **init_slew()**.

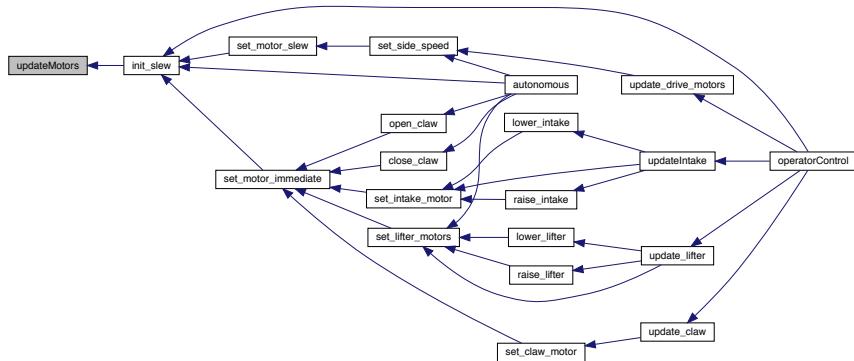
```

00013     {
00014     //Take back half approach
00015     //Not linear but equal to setSpeed(1-(1/2)^x)
00016     for(unsigned int i = 0; i < 9; i++) {
00017         if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
00018         mutexTake(speeds_mutex, 10);
00019         int set_speed = (motors_set_speeds[i]);
00020         int curr_speed = motors_curr_speeds[i];
00021         mutexGive(speeds_mutex);
00022         int diff = set_speed - curr_speed;
00023         int offset = diff;
00024         int n = curr_speed + offset;
00025         motors_curr_speeds[i] = n;
00026         motorSet(i+1, n);
00027     }
00028 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.40 slew.h

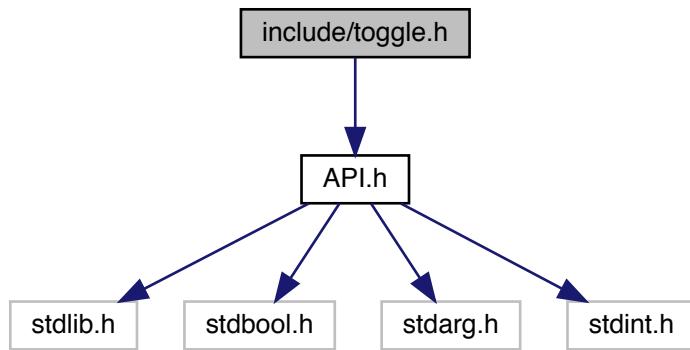
```

00001
00008 #ifndef _SLEW_H_
00009 #define _SLEW_H_
00010
00011 #include <API.h>
00012 #include <math.h>
00013 #include <vlib.h>
00014
00020 #define UPDATE_PERIOD_MS 25
00021
00027 #define MOTOR_PORTS 12
00028
00034 #define RAMP_PROPORTION 1
00035
00041 void updateMotors();
00042
00048 void deinitSlew();
00049
00055 void initSlew();
00056
  
```

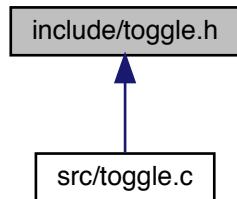
```
00064 void set_motor_slew(int motor, int speed);
00065
00066 void set_motor_immediate(int motor, int speed);
00067
00068 #endif
```

5.41 include/toggle.h File Reference

```
#include <API.h>
Include dependency graph for toggle.h:
```



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



Enumerations

- enum **button_t**{
 JOY1_5D = 0, **JOY1_5U** = 1, **JOY1_6D** = 2, **JOY1_6U** = 3,

```
JOY1_7U = 4, JOY1_7L = 5, JOY1_7R = 6, JOY1_7D = 7,
JOY1_8U = 8, JOY1_8L = 9, JOY1_8R = 10, JOY1_8D = 11,
JOY2_5D = 12, JOY2_5U = 13, JOY2_6D = 14, JOY2_6U = 15,
JOY2_7U = 16, JOY2_7L = 17, JOY2_7R = 18, JOY2_7D = 19,
JOY2_8U = 20, JOY2_8L = 21, JOY2_8R = 22, JOY2_8D = 23,
LCD_LEFT = 24, LCD_CENT = 25, LCD_RIGHT = 26 }
```

Functions

- **bool buttonGetState (button_t)**
Returns the current status of a button (pressed or not pressed)
- **void buttonInit ()**
Initializes the buttons.
- **bool buttonIsNewPress (button_t)**
Detects if button is a new press from most recent check by comparing previous value to current value.

5.41.1 Enumeration Type Documentation

5.41.1.1 button_t

```
enum button_t
```

Renames the input channels

Enumerator

JOY1_5D	
JOY1_5U	
JOY1_6D	
JOY1_6U	
JOY1_7U	
JOY1_7L	
JOY1_7R	
JOY1_7D	
JOY1_8U	
JOY1_8L	
JOY1_8R	
JOY1_8D	
JOY2_5D	
JOY2_5U	
JOY2_6D	
JOY2_6U	
JOY2_7U	
JOY2_7L	
JOY2_7R	

Enumerator

JOY2_7D	
JOY2_8U	
JOY2_8L	
JOY2_8R	
JOY2_8D	
LCD_LEFT	
LCD_CENT	
LCD_RIGHT	

Definition at line **20** of file **toggle.h**.

```

00020      {
00021      JOY1_5D = 0,
00022      JOY1_5U = 1,
00023      JOY1_6D = 2,
00024      JOY1_6U = 3,
00025      JOY1_7U = 4,
00026      JOY1_7L = 5,
00027      JOY1_7R = 6,
00028      JOY1_7D = 7,
00029      JOY1_8U = 8,
00030      JOY1_8L = 9,
00031      JOY1_8R = 10,
00032      JOY1_8D = 11,
00033
00034      JOY2_5D = 12,
00035      JOY2_5U = 13,
00036      JOY2_6D = 14,
00037      JOY2_6U = 15,
00038      JOY2_7U = 16,
00039      JOY2_7L = 17,
00040      JOY2_7R = 18,
00041      JOY2_7D = 19,
00042      JOY2_8U = 20,
00043      JOY2_8L = 21,
00044      JOY2_8R = 22,
00045      JOY2_8D = 23,
00046
00047      LCD_LEFT = 24,
00048      LCD_CENT = 25,
00049      LCD_RIGHT = 26
00050 } button_t;

```

5.41.2 Function Documentation**5.41.2.1 buttonGetState()**

```
bool buttonGetState (
    button_t   )
```

Returns the current status of a button (pressed or not pressed)

Parameters

<i>button</i>	The button to detect from the Buttons enumeration.
---------------	--

Returns

true (pressed) or false (not pressed)

Definition at line 25 of file **toggle.c**.

References **JOY_DOWN**, **JOY_LEFT**, **JOY_RIGHT**, **JOY_UP**, **joystickGetDigital()**, **LCD_BTN_CENTER**, **LCD_BTN_LEFT**, **LCD_BTN_RIGHT**, **LCD_CENT**, **LCD_LEFT**, **LCD_RIGHT**, **lcdReadButtons()**, and **uart1**.

Referenced by **buttonIsNewPress()**.

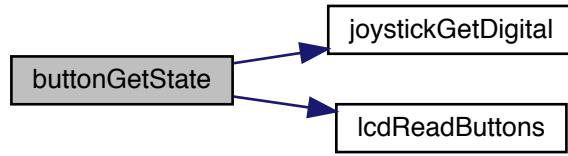
```

00025
00026     bool currentButton = false;
00027
00028 // Determine how to get the current button value (from what function) and where it
00029 // is, then get it.
00030 if (button < LCD_LEFT) {
00031     // button is a joystick button
00032     unsigned char joystick;
00033     unsigned char buttonGroup;
00034     unsigned char buttonLocation;
00035
00036     button_t newButton;
00037     if (button <= 11) {
00038         // button is on joystick 1
00039         joystick = 1;
00040         newButton = button;
00041     }
00042     else {
00043         // button is on joystick 2
00044         joystick = 2;
00045         // shift button down to joystick 1 buttons in order to
00046         // detect which button on joystick is queried
00047         newButton = (button_t)(button - 12);
00048     }
00049
00050     switch (newButton) {
00051     case 0:
00052         buttonGroup = 5;
00053         buttonLocation = JOY_DOWN;
00054         break;
00055     case 1:
00056         buttonGroup = 5;
00057         buttonLocation = JOY_UP;
00058         break;
00059     case 2:
00060         buttonGroup = 6;
00061         buttonLocation = JOY_DOWN;
00062         break;
00063     case 3:
00064         buttonGroup = 6;
00065         buttonLocation = JOY_UP;
00066         break;
00067     case 4:
00068         buttonGroup = 7;
00069         buttonLocation = JOY_UP;
00070         break;
00071     case 5:
00072         buttonGroup = 7;
00073         buttonLocation = JOY_LEFT;
00074         break;
00075     case 6:
00076         buttonGroup = 7;
00077         buttonLocation = JOY_RIGHT;
00078         break;
00079     case 7:
00080         buttonGroup = 7;
00081         buttonLocation = JOY_DOWN;
00082         break;
00083     case 8:
00084         buttonGroup = 8;
00085         buttonLocation = JOY_UP;
00086         break;
00087     case 9:
00088         buttonGroup = 8;

```

```
00089         buttonLocation = JOY_LEFT;
00090         break;
00091     case 10:
00092         buttonGroup = 8;
00093         buttonLocation = JOY_RIGHT;
00094         break;
00095     case 11:
00096         buttonGroup = 8;
00097         buttonLocation = JOY_DOWN;
00098         break;
00099     default:
00100         break;
00101     }
00102     currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
00103 }
00104 else {
00105     // button is on LCD
00106     if (button == LCD_LEFT)
00107         currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);
00108
00109     if (button == LCD_CENT)
00110         currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);
00111
00112     if (button == LCD_RIGHT)
00113         currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
00114 }
00115 return currentButton;
00116 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.41.2.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons.

Initializes the buttons.

Definition at line **20** of file **toggle.c**.

References **buttonPressed**.

```
00020      {
00021      for (int i = 0; i < 27; i++)
00022          buttonPressed[i] = false;
00023 }
```

5.41.2.3 buttonIsNewPress()

```
bool buttonIsNewPress (
    button_t button )
```

Detects if button is a new press from most recent check by comparing previous value to current value.

Parameters

<i>button</i>	The button to detect from the Buttons enumeration (see include/buttons.h).
---------------	--

Returns

true or false depending on if there was a change in button state.

Parameters

<i>button</i>	The button to detect from the Buttons enumeration (see include/buttons.h).
---------------	--

Returns

true or false depending on if there was a change in button state.

Example code:

```
...
if(buttonIsNewPress(JOY1_8D))
    digitalWrite(1, !digitalRead(1));
...
```

Definition at line 135 of file **toggle.c**.

References **buttonGetState()**, and **buttonPressed**.

```

00135
00136     bool currentButton = buttonGetState(button);
00137
00138     if (!currentButton) // buttons is not currently pressed
00139         buttonPressed[button] = false;
00140
00141     if (currentButton && !buttonPressed[button]) {
00142         // button is currently pressed and was not detected as being pressed during last check
00143         buttonPressed[button] = true;
00144         return true;
00145     }
00146     else return false; // button is not pressed or was already detected
00147 }
```

Here is the call graph for this function:



5.42 toggle.h

```

00001
00012 #ifndef BUTTONS_H_
00013 #define BUTTONS_H_
00014
00015 #include <API.h>
00016
00020 typedef enum {
00021     JOY1_5D = 0,
00022     JOY1_5U = 1,
00023     JOY1_6D = 2,
00024     JOY1_6U = 3,
00025     JOY1_7U = 4,
00026     JOY1_7L = 5,
00027     JOY1_7R = 6,
00028     JOY1_7D = 7,
00029     JOY1_8U = 8,
00030     JOY1_8L = 9,
00031     JOY1_8R = 10,
00032     JOY1_8D = 11,
00033
00034     JOY2_5D = 12,
00035     JOY2_5U = 13,
00036     JOY2_6D = 14,
00037     JOY2_6U = 15,
00038     JOY2_7U = 16,
00039     JOY2_7L = 17,
00040     JOY2_7R = 18,
00041     JOY2_7D = 19,
00042     JOY2_8U = 20,
00043     JOY2_8L = 21,
00044     JOY2_8R = 22,
00045     JOY2_8D = 23,
00046 }
```

```

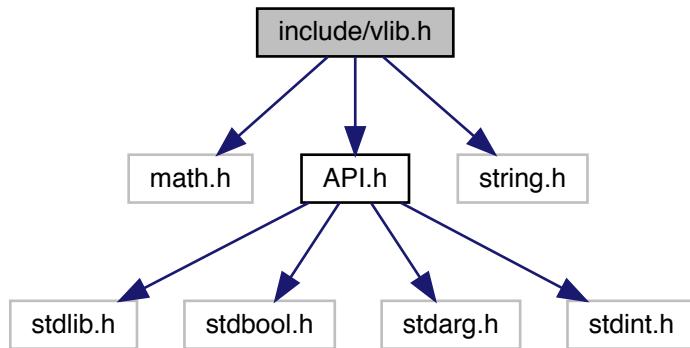
00047     LCD_LEFT = 24,
00048     LCD_CENT = 25,
00049     LCD_RIGHT = 26
00050 } button_t;
00051
00055 void buttonInit();
00056
00066 bool buttonIsNewPress(button_t);
00067
00076 bool buttonGetState(button_t);
00077
00078 #endif

```

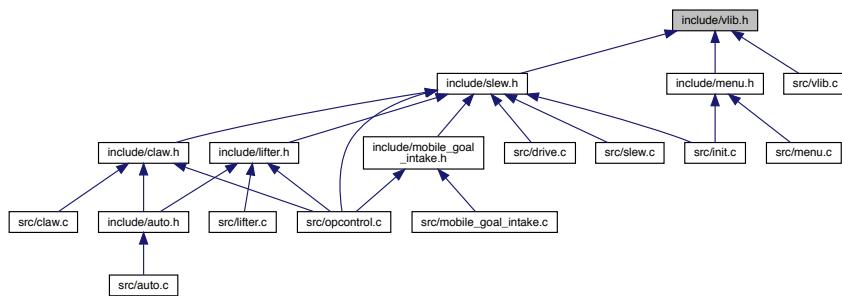
5.43 include/vlib.h File Reference

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 **ftoa_bad** (float *a*, char **buffer*, int *precision*)
converts a float to string.
- int **itoa_bad** (int *a*, char **buffer*, int *digits*)
converts a int to string.
- void **reverse** (char **str*, int *len*)
reverses a string 'str' of length 'len'

5.43.1 Detailed Description

Contains misc helpful functions.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **vlib.h**.

5.43.2 Function Documentation

5.43.2.1 ftoa_bad()

```
void ftoa_bad (
    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

Author

Christian DeSimone

Date

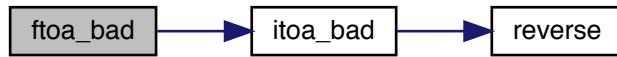
9/26/2017

Definition at line **30** of file **vlib.c**.References **itoa_bad()**.Referenced by **calculate_current_display()**.

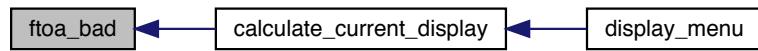
```

00030
00031     // Extract integer part
00032     int ipart = (int)a;
00033
00034     // Extract floating part
00035     float fpart = a - (float)ipart;
00036
00037     // convert integer part to string
00038     int i = itoa_bad(ipart, buffer, 0);
00039
00040     // check for display option after point
00041     if(precision != 0) {
00042         buffer[i] = '.'; // add dot
00043
00044         // Get the value of fraction part up to given num.
00045         // of points after dot. The third parameter is needed
00046         // to handle cases like 233.007
00047         fpart = fpart * pow(10, precision);
00048
00049         itoa_bad((int)fpart, buffer + i + 1, precision);
00050     }
00051 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.43.2.2 itoa_bad()

```
int itoa_bad (
    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, Christian DeSimone

Date

9/9/2017

Definition at line 13 of file **vlib.c**.

References **reverse()**.

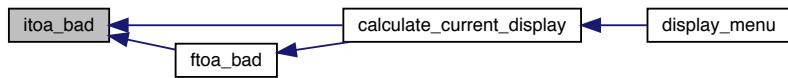
Referenced by **calculate_current_display()**, and **ftoa_bad()**.

```
00013     {
00014     int i = 0;
00015     while (a) {
00016         buffer[i++] = (a%10) + '0';
00017         a = a/10;
00018     }
00019
00020     // If number of digits required is more, then
00021     // add 0s at the beginning
00022     while (i < digits)
00023         buffer[i++] = '0';
00024
00025     reverse(buffer, i);
00026     buffer[i] = '\0';
00027     return i;
00028 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.43.2.3 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

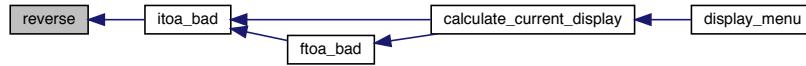
Definition at line 3 of file **vlib.c**.

Referenced by **itoa_bad()**.

```

00003     int i=0, j=len-1, temp;
00004     while (i<j) {
00005         temp = str[i];
00006         str[i] = str[j];
00007         str[j] = temp;
00008         i++; j--;
00009     }
00010 }
00011 }
```

Here is the caller graph for this function:



5.44 vlib.h

```

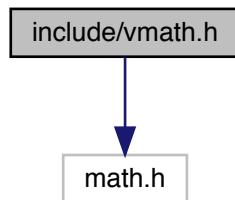
00001
00008 #ifndef _VLIB_H_
00009 #define _VLIB_H_
00010
00011 #include <math.h>
00012 #include <API.h>
00013 #include <string.h>
00014
00022 void reverse(char *str, int len);
00023
00033 int itoa_bad(int a, char *buffer, int digits);
00034
00043 void ftoa_bad(float a, char *buffer, int precision);
00044
00045 #endif
```

5.45 include/vmath.h File Reference

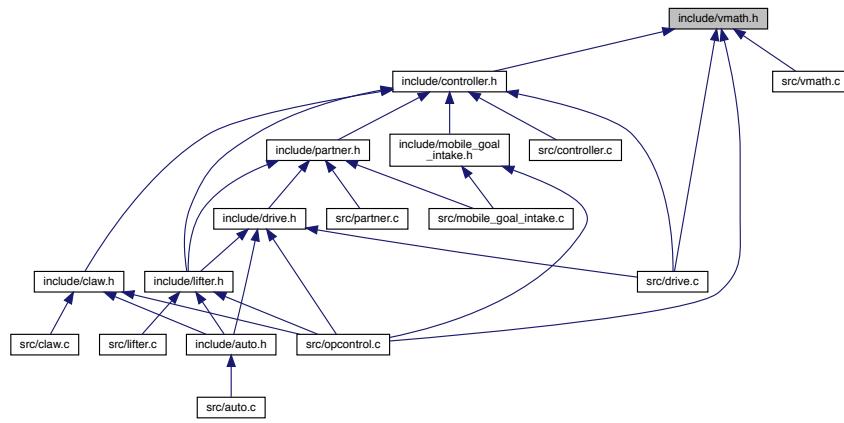
Vex Specific Math Functions, includes: Cartesian to polar coordinates.

```
#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.

Macros

- #define **M_PI** 3.14159265358979323846

Functions

- struct **polar_cord cartesian_cord_to_polar** (struct **cord** cords)

Function to convert x and y 2 dimensional cartesian cordinated 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.

- int **max** (int a, int b)

the min of two values

- int **min** (int a, int b)

the min of two values

- double **sind** (double angle)

sine of a angle in degrees

5.45.1 Detailed Description

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

Author

Christian Desimone
Chris Jerrett

Date

9/9/2017

Definition in file **vmath.h**.

5.45.2 Macro Definition Documentation

5.45.2.1 M_PI

```
#define M_PI 3.14159265358979323846
```

Definition at line **13** of file **vmath.h**.

Referenced by **sind()**.

5.45.3 Function Documentation

5.45.3.1 cartesian_cord_to_polar()

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

Function to convert x and y 2 dimensional cartesian coordinates 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 (p. 16)

cord (p. 6)

Definition at line 33 of file **vmath.c**.

References **cartesian_to_polar()**.

```
00033 {  
00034     return cartesian_to_polar(cords.x, cords.y);  
00035 }
```

Here is the call graph for this function:

**5.45.3.2 cartesian_to_polar()**

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

Function to convert x and y 2 dimensional cartesian coordinates 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 (p. 16)

Definition at line 3 of file **vmath.c**.

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

Referenced by **cartesian_cord_to_polar()**.

```

00003                                     {
00004     float degree = 0;
00005     double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
00006
00007     if(x < 0){
00008         degree += 180.0;
00009     }
00010     else if(x > 0 && y < 0){
00011         degree += 360.0;
00012     }
00013
00014     if(x != 0 && y != 0){
00015         degree += atan((float)y / (float)x);
00016     }
00017     else if(x == 0 && y > 0){
00018         degree = 90.0;
00019     }
00020     else if(y == 0 && x < 0){
00021         degree = 180.0;
00022     }
00023     else if(x == 0 && y < 0){
00024         degree = 270.0;
00025     }
00026
00027     struct polar_cord p;
00028     p.angle = degree;
00029     p.magnitue = magnitude;
00030     return p;
00031 }
```

Here is the caller graph for this function:



5.45.3.3 max()

```
int max (
    int a,
    int b )
```

the min of two values

Parameters

<i>a</i>	the first
<i>b</i>	the second

Returns

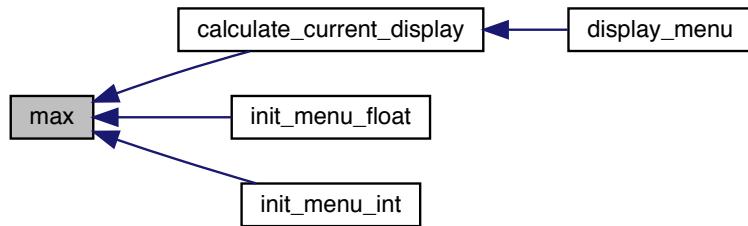
the smaller of *a* and *b*

Definition at line **48** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
00048
00049     if(a > b)  return a;
00050     return b;
00051 }
```

Here is the caller graph for this function:



5.45.3.4 min()

```
int min (
    int a,
    int b )
```

the min of two values

Parameters

<i>a</i>	the first
<i>b</i>	the second

Returns

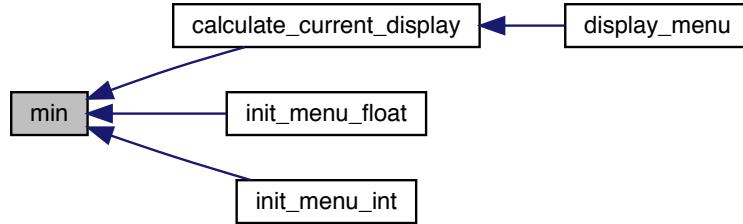
the smaller of *a* and *b*

Definition at line **42** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
00042
00043     if(a < b)  return a;
00044     return b;
00045 }
```

Here is the caller graph for this function:

**5.45.3.5 sind()**

```
double sind (
    double angle )
```

sine of a angle in degrees

Definition at line **37** of file **vmath.c**.

References **M_PI**.

```
00037
00038     double angleradians = angle * M_PI / 180.0f;
00039     return sin(angleradians);
00040 }
```

5.46 vmath.h

```

00001
00009 #ifndef _VMATH_H_
00010 #define _VMATH_H_
00011
00012 #include <math.h>
00013 #define M_PI 3.14159265358979323846
00014
00020 struct polar_cord {
00022     float angle;
00024     float magnitude;
00025 };
00026
00032 struct cord {
00034     float x;
00036     float y;
00037 };
00038
00050 struct polar_cord cartesian_to_polar(float x, float y);
00051
00063 struct polar_cord cartesian_cord_to_polar(struct cord cords);
00064
00071 int min(int a, int b);
00072
00079 int max(int a, int b);
00080
00084 double sind(double angle);
00085 #endif

```

5.47 README.md File Reference

5.48 README.md

```

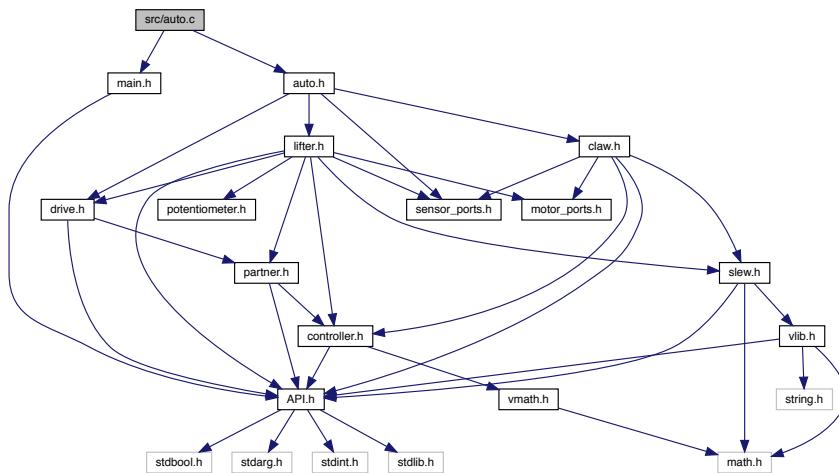
00001 # InTheZoneA
00002 Team A code for In The Zone

```

5.49 src/auto.c File Reference

File for autonomous code.

```
#include "main.h"
#include "auto.h"
Include dependency graph for auto.c:
```



Functions

- void **autonomous()**

5.49.1 Detailed Description

File for autonomous code.

This file should contain the user **autonomous()** (p. 253) 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.

Definition in file **auto.c**.

5.49.2 Function Documentation

5.49.2.1 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.

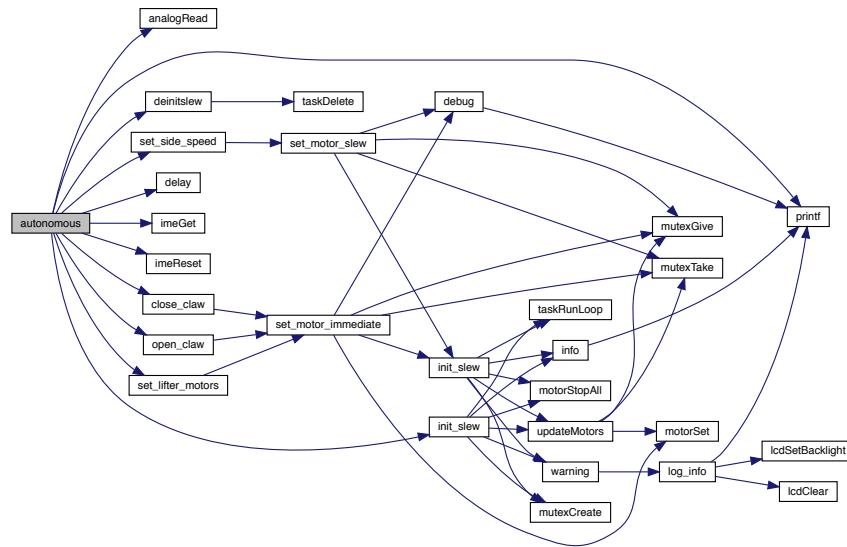
The autonomous task may exit, unlike **operatorControl()** (p. 175) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

Definition at line **30** of file **auto.c**.

References **analogRead()**, **BOTH**, **close_claw()**, **deinitSlew()**, **delay()**, **GOAL_HEIGHT**, **imeGet()**, **imeReset()**, **init_slew()**, **LIFTER**, **MID_LEFT_DRIVE**, **MID_RIGHT_DRIVE**, **open_claw()**, **printf()**, **set_lifter_motors()**, and **set_side_speed()**.

```
00030      {
00031      init_slew();
00032
00033      delay(10);
00034      printf("auto\n");
00035      //How far the left wheels have gone
00036      int counts_drive_left;
00037      //How far the right wheels have gone
00038      int counts_drive_right;
00039      //The average distance traveled forward
00040      int counts_drive;
00041
00042      //Reset the integrated motor controllers
00043      imeReset(MID_LEFT_DRIVE);
00044      imeReset(MID_RIGHT_DRIVE);
00045      //Set initial values for how far the wheels have gone
00046      imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00047      imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00048      counts_drive = counts_drive_left + counts_drive_right;
00049      counts_drive /= 2;
00050
00051      //Grab pre-load cone
00052      close_claw();
00053      delay(300);
00054
00055      //Raise the lifter
00056      while(analogRead(LIFTER) < GOAL_HEIGHT){
00057          set_lifter_motors(-127);
00058      }
00059      set_lifter_motors(0);
00060      //Drive towards the goal
00061      while(counts_drive < 530){
00062          set_side_speed(BOTH, 127);
00063          //Restablish the distance traveled
00064          imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00065          imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00066          counts_drive = counts_drive_left + counts_drive_right;
00067          counts_drive /= 2;
00068      }
00069      //Stop moving
00070      set_side_speed(BOTH, 0);
00071      delay(1000);
00072
00073      //Drop the cone on the goal
00074      open_claw();
00075      delay(1000);
00076      deinitSlew();
00077 }
```

Here is the call graph for this function:



5.50 auto.c

```

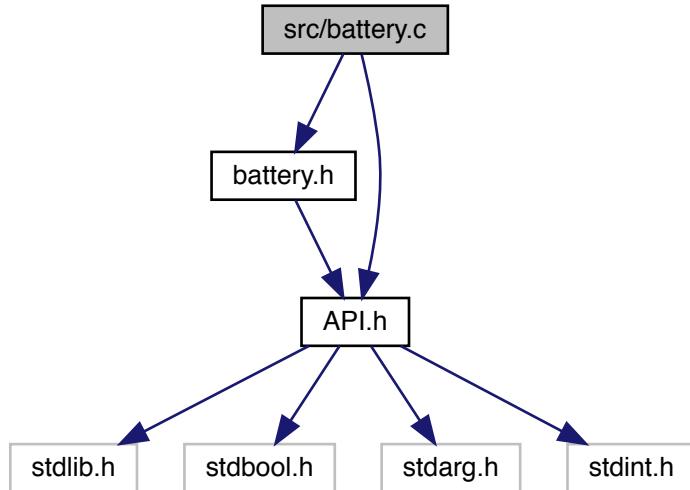
00001
00013 #include "main.h"
00014 #include "auto.h"
00015
00016 /*
00017 * Runs the user autonomous code. This function will be started in its own task with the default
00018 * priority and stack size whenever the robot is enabled via the Field Management System or the
00019 * VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is
00020 * lost, the autonomous task will be stopped by the kernel. Re-enabling the robot will restart
00021 * the task, not re-start it from where it left off.
00022 *
00023 * Code running in the autonomous task cannot access information from the VEX Joystick. However,
00024 * the autonomous function can be invoked from another task if a VEX Competition Switch is not
00025 * available, and it can access joystick information if called in this way.
00026 *
00027 * The autonomous task may exit, unlike operatorControl() which should never exit. If it does
00028 * so, the robot will await a switch to another mode or disable/enable cycle.
00029 */
00030 void autonomous() {
00031     init_slew();
00032
00033     delay(10);
00034     printf("auto\n");
00035     //How far the left wheels have gone
00036     int counts_drive_left;
00037     //How far the right wheels have gone
00038     int counts_drive_right;
00039     //The average distance traveled forward
00040     int counts_drive;
00041
00042     //Reset the integrated motor controllers
00043     imeReset(MID_LEFT_DRIVE);
00044     imeReset(MID_RIGHT_DRIVE);
00045     //Set initial values for how far the wheels have gone
00046     imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00047     imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00048     counts_drive = counts_drive_left + counts_drive_right;
00049     counts_drive /= 2;
00050
00051     //Grab pre-load cone
  
```

```

00052     close_claw();
00053     delay(300);
00054
00055     //Raise the lifter
00056     while(analogRead(LIFTER) < GOAL_HEIGHT) {
00057         set_lifter_motors(-127);
00058     }
00059     set_lifter_motors(0);
00060     //Drive towards the goal
00061     while(counts_drive < 530) {
00062         set_side_speed(BOTH, 127);
00063         //Establish the distance traveled
00064         imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00065         imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00066         counts_drive = counts_drive_left + counts_drive_right;
00067         counts_drive /= 2;
00068     }
00069     //Stop moving
00070     set_side_speed(BOTH, 0);
00071     delay(1000);
00072
00073     //Drop the cone on the goal
00074     open_claw();
00075     delay(1000);
00076     deinitSlew();
00077 }
```

5.51 src/battery.c File Reference

```
#include "battery.h"
#include <API.h>
Include dependency graph for battery.c:
```



Functions

- double **backup_battery_voltage ()**

- **bool battery_level_acceptable ()**
gets the backup battery voltage
- **double main_battery_voltage ()**
returns if the batteries are acceptable
- **double main_battery_voltage ()**
gets the main battery voltage

5.51.1 Function Documentation

5.51.1.1 backup_battery_voltage()

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

Author

Chris Jerrett

Definition at line 17 of file **battery.c**.

References **powerLevelBackup()**.

Referenced by **battery_level_acceptable()**.

```
00017     {  
00018     return powerLevelBackup() / 1000.0;  
00019 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.51.1.2 battery_level_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

See also

MIN_MAIN_VOLTAGE (p. 99)
MIN_BACKUP_VOLTAGE (p. 99)

Author

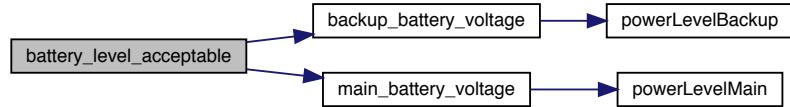
Chris Jerrett

Definition at line 28 of file **battery.c**.

References **backup_battery_voltage()**, **main_battery_voltage()**, **MIN_BACKUP_VOLTAGE**, and **MIN_MAIN_VOLTAGE**.

```
00028     {
00029     if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
00030     if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
00031     return true;
00032 }
```

Here is the call graph for this function:



5.51.1.3 main_battery_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

Author

Chris Jerrett

Definition at line 9 of file **battery.c**.

References **powerLevelMain()**.

Referenced by **battery_level_acceptable()**.

```
00009     {
00010     return powerLevelMain() / 1000.0;
00011 }
```

Here is the call graph for this function:



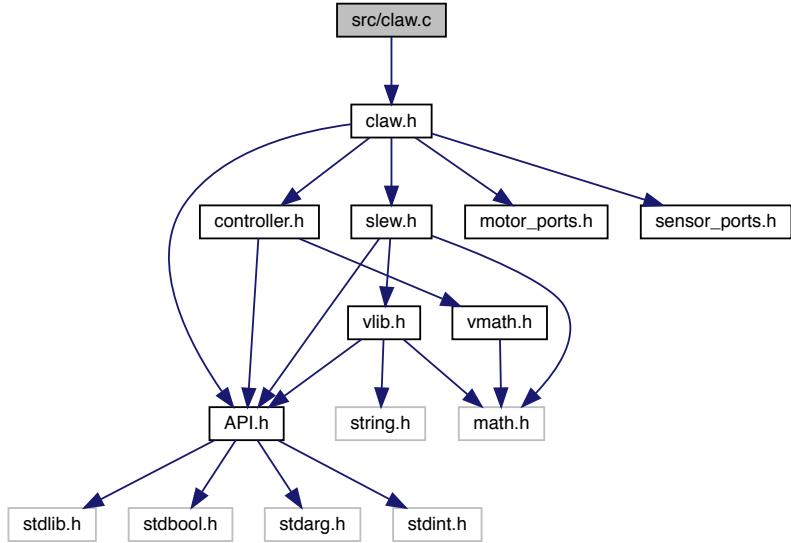
Here is the caller graph for this function:

**5.52 battery.c**

```
00001 #include "battery.h"
00002 #include <API.h>
00003
00004
00009 double main_battery_voltage() {
00010     return powerLevelMain() / 1000.0;
00011 }
00012
00017 double backup_battery_voltage() {
00018     return powerLevelBackup() / 1000.0;
00019 }
00020
00028 bool battery_level_acceptable() {
00029     if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
00030     if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
00031     return true;
00032 }
```

5.53 src/claw.c File Reference

```
#include "claw.h"
Include dependency graph for claw.c:
```



Functions

- void **close_claw ()**
Drives the motors to close the claw.
- unsigned int **getClawTicks ()**
Gets the claw position in potentiometer ticks.
- void **open_claw ()**
Drives the motors to open the claw.
- void **set_claw_motor** (const int v)
sets the claw motor speed
- void **update_claw ()**
Updates the claw motor values.

5.53.1 Function Documentation

5.53.1.1 close_claw()

```
void close_claw ( )
```

Drives the motors to close the claw.

Author

Chris Jerrett

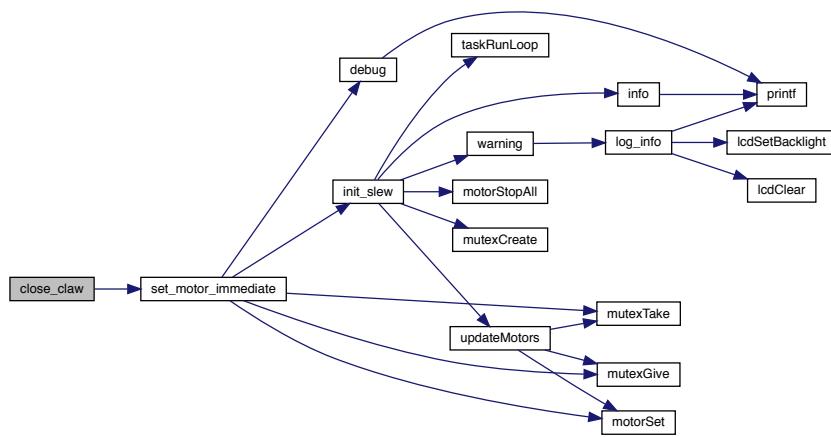
Definition at line **67** of file **claw.c**.

References **CLAW_MOTOR**, **MIN_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00067      {
00068      set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
00069 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.53.1.2 getClawTicks()

```
unsigned int getClawTicks ( )
```

Gets the claw position in potentiometer ticks.

Author

Chris Jerrett

Definition at line **51** of file **claw.c**.

References **analogRead()**, and **CLAW_POT**.

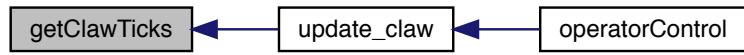
Referenced by **update_claw()**.

```
00051     {  
00052     return analogRead(CLAW_POT);  
00053 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.53.1.3 open_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

Author

Chris Jerrett

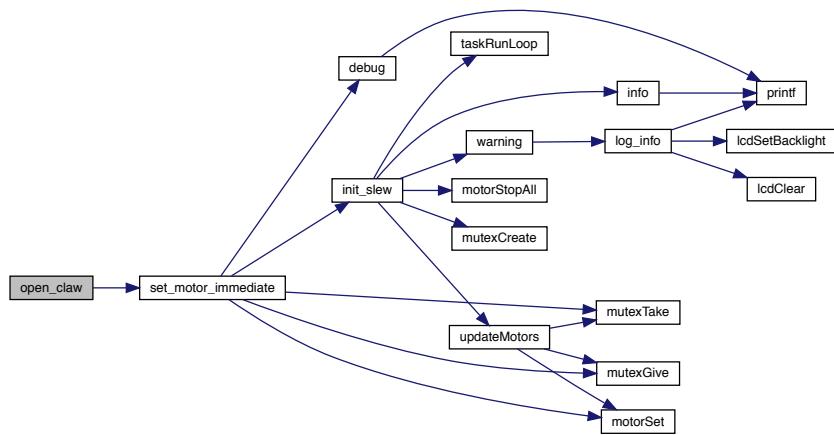
Definition at line **59** of file **claw.c**.

References **CLAW_MOTOR**, **MAX_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00059           {  
00060     set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);  
00061 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.53.1.4 set_claw_motor()

```
void set_claw_motor (
    const int v )
```

sets the claw motor speed

Author

Chris Jerrett

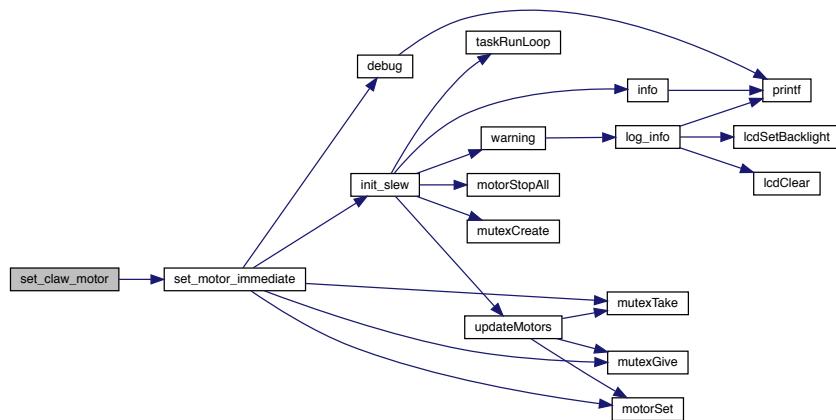
Definition at line **43** of file **claw.c**.

References **CLAW_MOTOR**, and **set_motor_immediate()**.

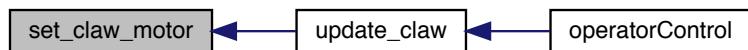
Referenced by **update_claw()**.

```
00043     {
00044     set_motor_immediate(CLAW_MOTOR, v);
00045 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.53.1.5 update_claw()

```
void update_claw ( )
```

Updates the claw motor values.

Author

Chris Jerrett

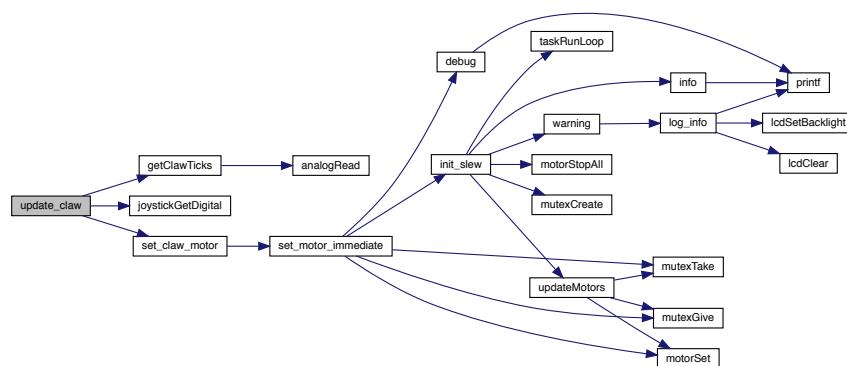
Definition at line 7 of file **claw.c**.

References **CLAW_CLOSE**, **CLAW_CLOSE_STATE**, **CLAW_CLOSE_VAL**, **CLAW_D**, **CLAW_OPEN**, **CLAW_OPEN_STATE**, **CLAW_OPEN_VAL**, **CLAW_P**, **getClawTicks()**, **joystickGetDigital()**, and **set_claw_motor()**.

Referenced by **operatorControl()**.

```
00007      {
00008      //Set the Error used in calculating d
00009      static int last_error = 0;
00010      //Set the initial claw state to open
00011      static enum claw_state state = CLAW_OPEN_STATE;
00012      //Listen for input and either close or open the claw
00013      if(joystickGetDigital(CLAW_CLOSE)){
00014          state = CLAW_CLOSE_STATE;
00015      }
00016      else if(joystickGetDigital(CLAW_OPEN)){
00017          state = CLAW_OPEN_STATE;
00018      } else {
00019          //set the default motor speed
00020          int p = 0;
00021          //Change the base speed to the difference between the target
00022          // and the current value
00023          if(state == CLAW_OPEN_STATE) {
00024              p = getClawTicks() - CLAW_OPEN_VAL;
00025          } else {
00026              p = getClawTicks() - CLAW_CLOSE_VAL;
00027          }
00028          //Set the d value to the difference between the current p and the last p
00029          int d = (p - last_error);
00030          //Set last error for use the next time the function is run
00031          last_error = p;
00032          //Construct the final motor speed value
00033          int motor = CLAW_P * p + CLAW_D * d;
00034          //Set the motor speed
00035          set_claw_motor(motor);
00036      }
00037 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.54 claw.c

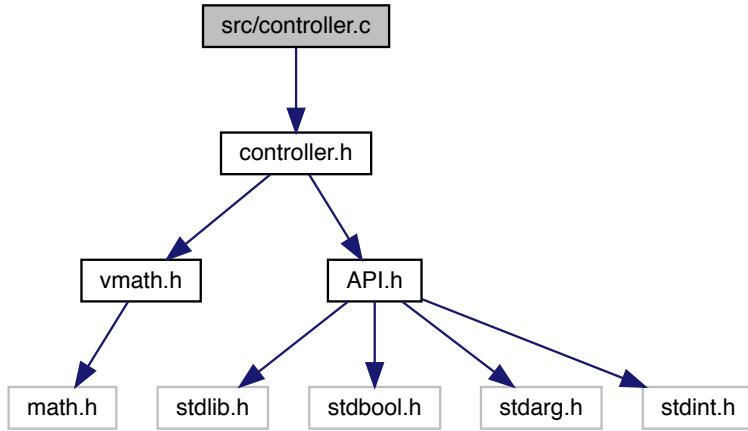
```

00001 #include "claw.h"
00002
00007 void update_claw() {
00008     //Set the Error used in calculating d
00009     static int last_error = 0;
00010     //Set the initial claw state to open
00011     static enum claw_state state = CLAW_OPEN_STATE;
00012     //Listen for input and either close or open the claw
00013     if(joystickGetDigital(CLAW_CLOSE)){
00014         state = CLAW_CLOSE_STATE;
00015     }
00016     else if(joystickGetDigital(CLAW_OPEN) ){
00017         state = CLAW_OPEN_STATE;
00018     } else {
00019         //set the default motor speed
00020         int p = 0;
00021         //Change the base speed to the difference between the target
00022         // and the current value
00023         if(state == CLAW_OPEN_STATE) {
00024             p = getClawTicks() - CLAW_OPEN_VAL;
00025         } else {
00026             p = getClawTicks() - CLAW_CLOSE_VAL;
00027         }
00028         //Set the d value to the difference between the current p and the last p
00029         int d = (p - last_error);
00030         //Set last error for use the next time the function is run
00031         last_error = p;
00032         //Construct the final motor speed value
00033         int motor = CLAW_P * p + CLAW_D * d;
00034         //Set the motor speed
00035         set_claw_motor(motor);
00036     }
00037 }
00038
00043 void set_claw_motor(const int v){
00044     set_motor_immediate(CLAW_MOTOR, v);
00045 }
00046
00051 unsigned int getClawTicks(){
00052     return analogRead(CLAW_POT);
00053 }
00054
00059 void open_claw() {
00060     set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);
00061 }
00062
00067 void close_claw() {
00068     set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
00069 }
  
```

5.55 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)
Gets the location of a joystick on the controller.

5.55.1 Function Documentation

5.55.1.1 get_joystick_cord()

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

Gets the location of a joystick on the controller.

Author

Chris Jerrett

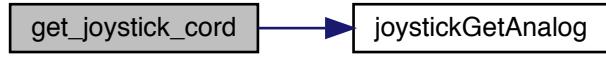
Definition at line 7 of file **controller.c**.

References **joystickGetAnalog()**, **LEFT_JOY_X**, **LEFT_JOY_Y**, **RIGHT_JOY**, **RIGHT_JOY_X**, **RIGHT_JOY_Y**, **cord::x**, and **cord::y**.

```

00007
00008     int x;
00009     int y;
00010    //Get the joystick value for either the right or left,
00011    //depending on the mode
00012    if(side == RIGHT_JOY) {
00013        y = joystickGetAnalog(controller, RIGHT_JOY_X);
00014        x = joystickGetAnalog(controller, RIGHT_JOY_Y);
00015    } else {
00016        y = joystickGetAnalog(controller, LEFT_JOY_X);
00017        x = joystickGetAnalog(controller, LEFT_JOY_Y);
00018    }
00019    //Define a coordinate for the joystick value
00020    struct cord c;
00021    c.x = x;
00022    c.y = y;
00023    return c;
00024 }
```

Here is the call graph for this function:



5.56 controller.c

```

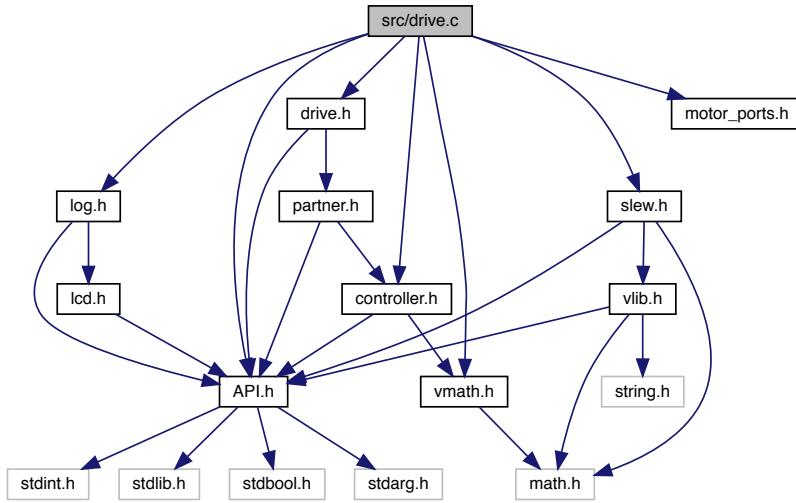
00001 #include "controller.h"
00002
00007 struct cord get_joystick_cord(enum joystick side, int controller) {
00008     int x;
00009     int y;
00010    //Get the joystick value for either the right or left,
00011    //depending on the mode
00012    if(side == RIGHT_JOY) {
00013        y = joystickGetAnalog(controller, RIGHT_JOY_X);
00014        x = joystickGetAnalog(controller, RIGHT_JOY_Y);
00015    } else {
00016        y = joystickGetAnalog(controller, LEFT_JOY_X);
00017        x = joystickGetAnalog(controller, LEFT_JOY_Y);
00018    }
00019    //Define a coordinate for the joystick value
00020    struct cord c;
00021    c.x = x;
00022    c.y = y;
00023    return c;
00024 }
```

5.57 src/drive.c File Reference

```

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

```
#include "log.h"
Include dependency graph for drive.c:
```



Functions

- int **getThresh ()**
Gets the deadzone threshold on the drive.
- static float **joystickExp** (int joystickVal)
Applies exponential scale to a joystick value.
- void **set_side_speed** (side_t side, int speed)
sets the speed of one side of the robot.
- void **setThresh** (int t)
Sets the deadzone threshold on the drive.
- void **update_drive_motors ()**
Updates the drive motors during teleop.

Variables

- static int **thresh** = 30

5.57.1 Function Documentation

5.57.1.1 getThresh()

```
int getThresh ( )
```

Gets the deadzone threshhold on the drive.

Author

Christian Desimone

Definition at line **17** of file **drive.c**.

References **thresh**.

```
00017      {
00018     return thresh;
00019 }
```

5.57.1.2 joystickExp()

```
static float joystickExp (
    int joystickVal ) [static]
```

Applies exponential scale to a joystick value.

Author

Christian DeSimone, Chris Jerrett

Parameters

<i>joystickVal</i>	the analog value from the joystick
--------------------	------------------------------------

Date

9/21/2017

Definition at line **87** of file **drive.c**.

References **THRESHOLD**.

```
00087      {
00088     //make the offset negative if moving backwards
00089     if (abs(joystickVal) < THRESHOLD) {
00090         return 0;
00091     }
00092 }
```

```

00093     int offset;
00094     //Use the threshold to ensure the joystick values are significant
00095     if (joystickVal < 0) {
00096         offset = - (THRESHOLD);
00097     } else {
00098         offset = THRESHOLD;
00099     }
00100    //Apply the function (((x/10)^3)/18) + offset) * 0.8 to the joystick value
00101    return (pow(joystickVal/10 , 3) / 18 + offset) * 0.8;
00102 }
```

5.57.1.3 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 **68** of file **drive.c**.

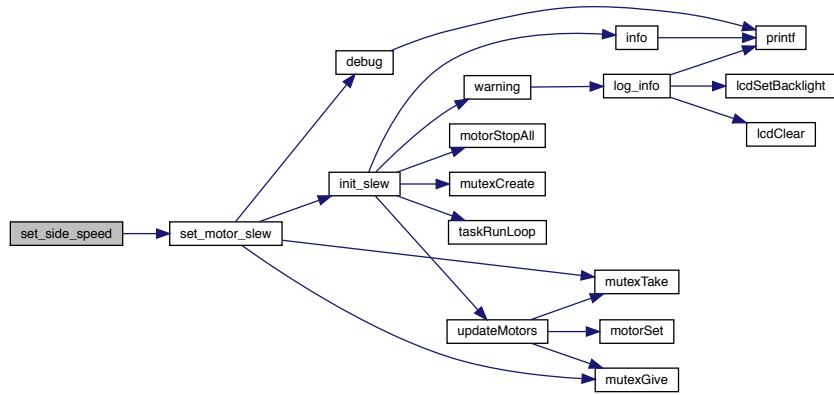
References **BOTH**, **LEFT**, **MOTOR_BACK_LEFT**, **MOTOR_BACK_RIGHT**, **MOTOR_FRONT_LEFT**, **MOTOR_FRONT_RIGHT**, **MOTOR_MIDDLE_LEFT**, **MOTOR_MIDDLE_RIGHT**, **RIGHT**, and **set_motor_slew()**.

Referenced by **autonomous()**, and **update_drive_motors()**.

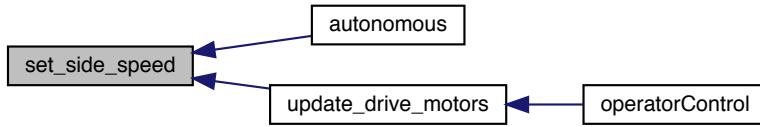
```

00068
00069     if(side == RIGHT || side == BOTH){
00070         set_motor_slew(MOTOR_BACK_RIGHT , -speed);
00071         set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
00072         set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
00073     }
00074     if(side == LEFT || side == BOTH){
00075         set_motor_slew(MOTOR_BACK_LEFT, speed);
00076         set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
00077         set_motor_slew(MOTOR_FRONT_LEFT, speed);
00078     }
00079 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.57.1.4 setThresh()

```
void setThresh (
    int t )
```

Sets the deadzone threshhold on the drive.

Author

Christian Desimone

Definition at line **25** of file **drive.c**.

References **thresh**.

```
00025
00026     thresh = t;
00027 }
```

5.57.1.5 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

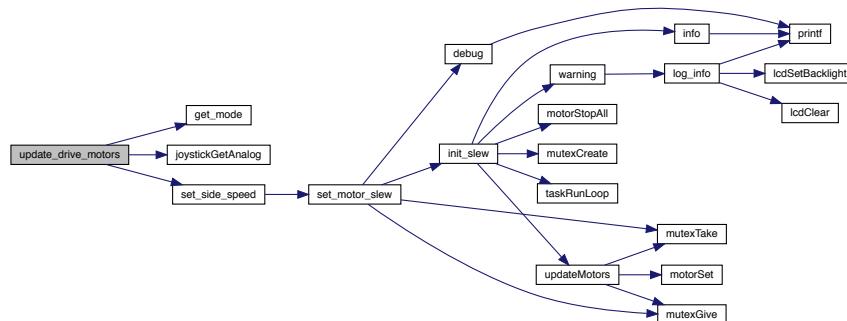
Definition at line 34 of file **drive.c**.

References **get_mode()**, **joystickGetAnalog()**, **LEFT**, **MASTER**, **PARTNER**, **PARTNER_CONTROLLER_MODE**, **RIGHT**, **set_side_speed()**, **thresh**, **cord::x**, and **cord::y**.

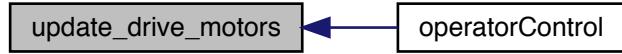
Referenced by **operatorControl()**.

```
00034 {
00035 //Get the joystick values from the controller
00036 int x = 0;
00037 int y = 0;
00038 if(get_mode() == PARTNER_CONTROLLER_MODE) {
00039     x = (joystickGetAnalog(PARTNER, 3));
00040     y = (joystickGetAnalog(PARTNER, 1));
00041 } else {
00042     x = -(joystickGetAnalog(MASTER, 3));
00043     y = (joystickGetAnalog(MASTER, 1));
00044 }
00045 //Make sure the joystick values are significant enough to change the motors
00046 if(x < thresh && x > -thresh){
00047     x = 0;
00048 }
00049 if(y < thresh && y > -thresh){
00050     y = 0;
00051 }
00052 //Create motor values for the left and right from the x and y of the joystick
00053 int r = (x + y);
00054 int l = -(x - y);
00055
00056 //Set the drive motors
00057 set_side_speed(LEFT, l);
00058 set_side_speed(RIGHT, -r);
00059 }
00060 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.57.2 Variable Documentation

5.57.2.1 thresh

```
int thresh = 30 [static]
```

Definition at line **10** of file **drive.c**.

Referenced by **getThresh()**, **setThresh()**, and **update_drive_motors()**.

5.58 drive.c

```

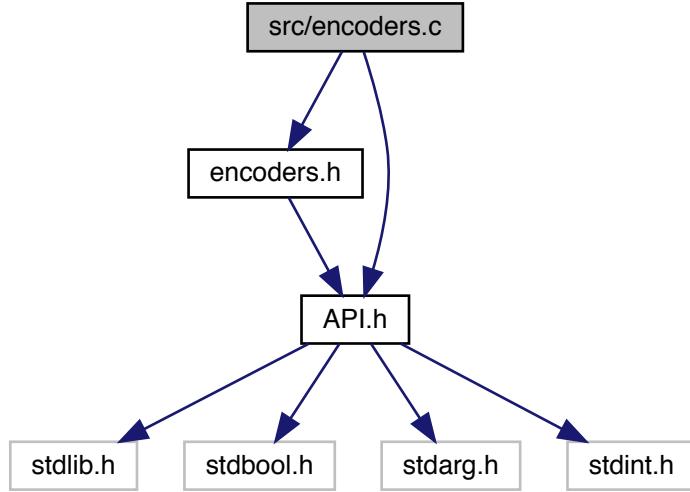
00001 #include "drive.h"
00002 #include "motor_ports.h"
00003 #include "vmath.h"
00004 #include "controller.h"
00005 #include "slew.h"
00006 #include "controller.h"
00007 #include <API.h>
00008 #include "log.h"
00009
00010 static int thresh = 30;
00011
00012
00017 int getThresh(){
00018     return thresh;
00019 }
00020
00025 void setThresh(int t){
00026     thresh = t;
00027 }
00028
00034 void update_drive_motors(){
00035     //Get the joystick values from the controller
00036     int x = 0;
00037     int y = 0;
00038     if(get_mode() == PARTNER_CONTROLLER_MODE) {
00039         x = (joystickGetAnalog(PARTNER, 3));
00040         y = (joystickGetAnalog(PARTNER, 1));
00041     } else {
00042         x = -(joystickGetAnalog(MASTER, 3));
00043         y = (joystickGetAnalog(MASTER, 1));
00044     }
00045     //Make sure the joystick values are significant enough to change the motors
00046     if(x < thresh && x > -thresh){
00047         x = 0;
00048     }
  
```

```
00049     if(y < thresh && y > -thresh){  
00050         y = 0;  
00051     }  
00052     //Create motor values for the left and right from the x and y of the joystick  
00053     int r = (x + y);  
00054     int l = -(x - y);  
00055  
00056     //Set the drive motors  
00057     set_side_speed(LEFT, l);  
00058     set_side_speed(RIGHT, -r);  
00059  
00060 }  
00061  
00068 void set_side_speed(side_t side, int speed){  
00069     if(side == RIGHT || side == BOTH){  
00070         set_motor_slew(MOTOR_BACK_RIGHT, -speed);  
00071         set_motor_slew(MOTOR_FRONT_RIGHT, -speed);  
00072         set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);  
00073     }  
00074     if(side == LEFT || side == BOTH){  
00075         set_motor_slew(MOTOR_BACK_LEFT, speed);  
00076         set_motor_slew(MOTOR_MIDDLE_LEFT, speed);  
00077         set_motor_slew(MOTOR_FRONT_LEFT, speed);  
00078     }  
00079 }  
00080  
00087 static float joystickExp(int joystickVal) {  
00088     //make the offset negative if moving backwards  
00089     if (abs(joystickVal) < THRESHOLD) {  
00090         return 0;  
00091     }  
00092  
00093     int offset;  
00094     //Use the threshold to ensure the joystick values are significant  
00095     if (joystickVal < 0) {  
00096         offset = - (THRESHOLD);  
00097     } else {  
00098         offset = THRESHOLD;  
00099     }  
00100    //Apply the function (((x/10)^3)/18) + offset * 0.8 to the joystick value  
00101    return (pow(joystickVal/10 , 3) / 18 + offset) * 0.8;  
00102 }
```

5.59 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.

5.59.1 Function Documentation

5.59.1.1 `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 **23** of file **encoders.c**.References **imeGet()**.

```
00023     {  
00024     int i = 0;  
00025     imeGet(address, &i);  
00026     return i;  
00027 }
```

Here is the call graph for this function:

**5.59.1.2 get_encoder_velocity()**

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

Gets the encoder reads.

Author

Chris Jerrett

Date

9/15/2017

Definition at line **34** of file **encoders.c**.References **imeGetVelocity()**.

```
00034     {  
00035     int i = 0;  
00036     imeGetVelocity(address, &i);  
00037     return i;  
00038 }
```

Here is the call graph for this function:



5.59.1.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

IME_NUMBER (p. 129)

Definition at line **10** of file **encoders.c**.

References **IME_NUMBER**, and **imeInitializeAll()**.

```
00010     {
00011     #ifdef IME_NUMBER
00012     return imeInitializeAll() == IME_NUMBER;
00013     #else
00014     return imeInitializeAll();
00015     #endif
00016 }
```

Here is the call graph for this function:



5.60 encoders.c

```

00001 #include "encoders.h"
00002 #include <API.h>
00003
00010 bool init_encoders() {
00011     #ifdef IME_NUMBER
00012         return imeInitializeAll() == IME_NUMBER;
00013     #else
00014         return imeInitializeAll();
00015     #endif
00016 }
00017
00023 int get_encoder_ticks(unsigned char address) {
00024     int i = 0;
00025     imeGet(address, &i);
00026     return i;
00027 }
00028
00034 int get_encoder_velocity(unsigned char address) {
00035     int i = 0;
00036     imeGetVelocity(address, &i);
00037     return i;
00038 }

```

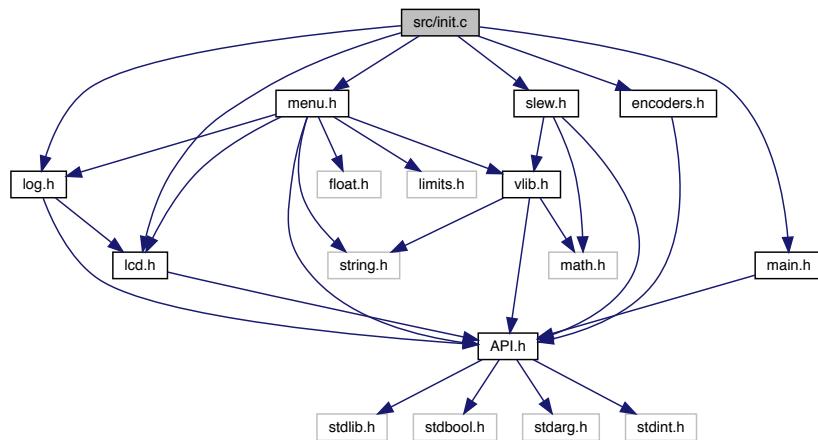
5.61 src/init.c File Reference

File for initialization code.

```

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

```



Functions

- void **initialize()**
- void **initializeIO()**

5.61.1 Detailed Description

File for initialization code.

This file should contain the user **initialize()** (p. 280) 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.

Definition in file **init.c**.

5.61.2 Function Documentation

5.61.2.1 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.

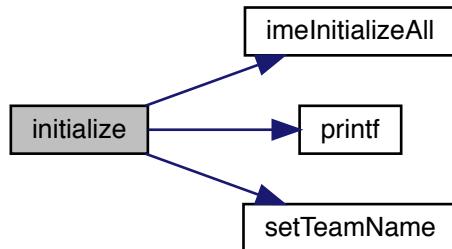
This function must exit relatively promptly, or the **operatorControl()** (p. 175) and **autonomous()** (p. 172) tasks will not start. An autonomous mode selection menu like the pre_auton() in other environments can be implemented in this task if desired.

Definition at line **47** of file **init.c**.

References **imeInitializeAll()**, **printf()**, and **setTeamName()**.

```
00047      {
00048      int c = imeInitializeAll();
00049      setTeamName("9228A");
00050      printf("Counts : %d\n", c);
00051 }
```

Here is the call graph for this function:



5.61.2.2 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()** (p. 74)) and port states (**digitalWrite()** (p. 40)) of limit switches, push buttons, and solenoids. It can also safely configure a UART port (**uartOpen()**) but cannot set up an LCD (**lcdInit()** (p. 65)).

Definition at line **30** of file **init.c**.

References **watchdogInit()**.

```
00030     {
00031     watchdogInit();
00032 }
```

Here is the call graph for this function:



5.62 init.c

```

00001
00012 #include "main.h"
00013 #include "slew.h"
00014 #include "lcd.h"
00015 #include "log.h"
00016 #include "encoders.h"
00017 #include "menu.h"
00018
00019 /*
00020 * Runs pre-initialization code. This function will be started in kernel mode one time while the
00021 * VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.
00022 *
00023 * The purpose of this function is solely to set the default pin modes (pinMode()) and port
00024 * states (digitalWrite()) of limit switches, push buttons, and solenoids. It can also safely
00025 * configure a UART port (uartOpen()) but cannot set up an LCD (lcdInit()).
00026 *
00027 * AKA DON'T USE
00028 * -Chris
00029 */
00030 void initializeIO() {
00031     watchdogInit();
00032 }
00033
00034 /*
00035 * Runs user initialization code. This function will be started in its own task with the default
00036 * priority and stack size once when the robot is starting up. It is possible that the VEXnet
00037 * communication link may not be fully established at this time, so reading from the VEX
00038 * Joystick may fail.
  
```

```

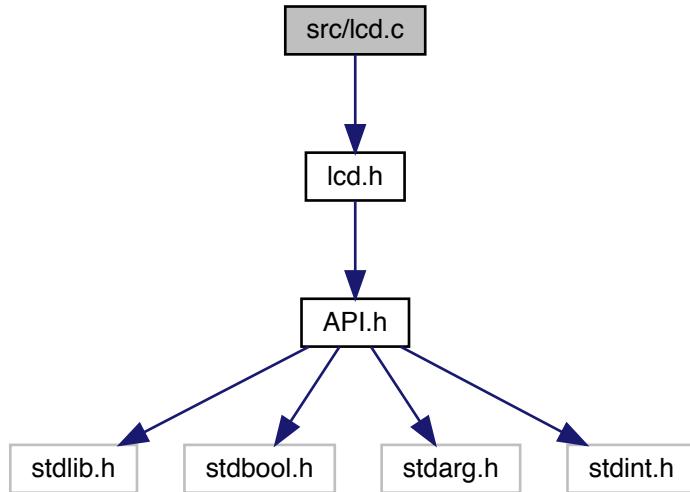
00039  *
00040  * This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global
00041  * variables, and IMEs.
00042  *
00043  * This function must exit relatively promptly, or the operatorControl() and autonomous() tasks
00044  * will not start. An autonomous mode selection menu like the pre_auton() in other environments
00045  * can be implemented in this task if desired.
00046  */
00047 void initialize() {
00048     int c = imeInitializeAll();
00049     setTeamName("9228A");
00050     printf("Counts : %d\n", c);
00051 }

```

5.63 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 formated string to a line on the lcd. Smilar 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

5.63.1 Function Documentation

5.63.1.1 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

lcd	the urart port of the lcd screen
------------	----------------------------------

See also

uart1 (p. 33)
uart2 (p. 33)

Author

Chris Jerrett

Date

9/9/2017

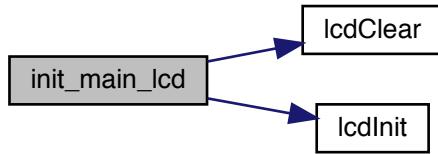
Definition at line **60** of file **lcd.c**.

References **lcd_port**, **lcdClear()**, and **lcdInit()**.

```

00060
00061     lcdInit(lcd);
00062     lcdClear(lcd);
00063     lcd_port = lcd;
00064 }
```

Here is the call graph for this function:



5.63.1.2 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**, and **printf()**.

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

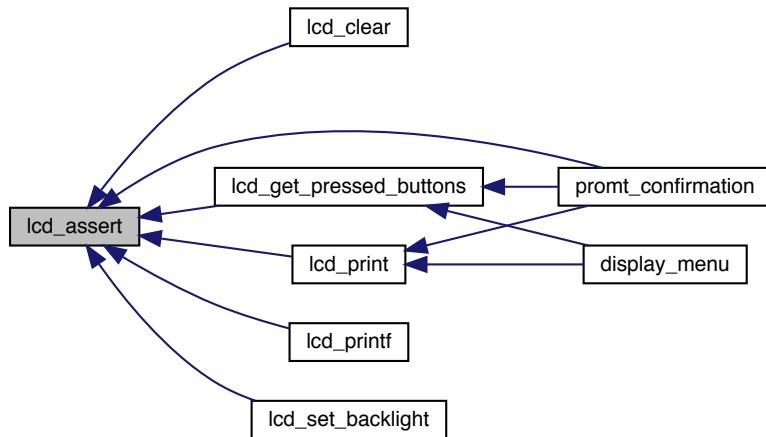
```

00013
00014     if(lcd_port != NULL) {
00015         printf("LCD NULL!");
00016         exit(1);
00017     }
00018 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.63.1.3 lcd_clear()

```
void lcd_clear( )
```

Clears the lcd.

Author

Chris Jerrett

Date

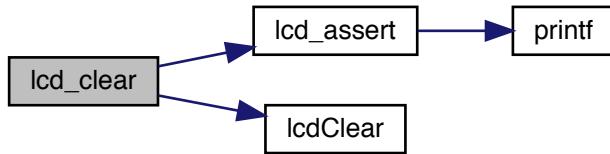
9/9/2017

Definition at line **46** of file **lcd.c**.

References **lcd_assert()**, **lcd_port**, and **lcdClear()**.

```
00046      {
00047      lcd_assert();
00048      lcdClear(lcd_port);
00049 }
```

Here is the call graph for this function:



5.63.1.4 `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 \(p. 8\)](#)Definition at line 27 of file **lcd.c**.

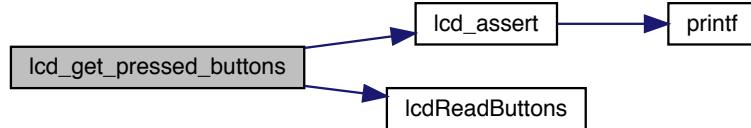
References **lcd_assert()**, **lcd_port**, **LcdReadButtons()**, **lcd_buttons::left**, **lcd_buttons::middle**, **PRESSED**, **R←RELEASED**, and **lcd_buttons::right**.

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

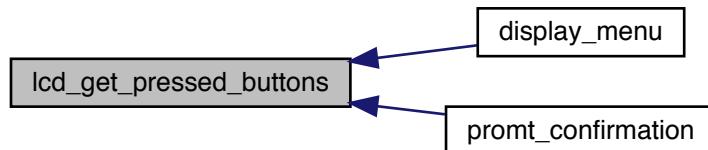
```

00027
00028     lcd_assert();                                {
00029     unsigned int btn_binary = lcdReadButtons(lcd_port);
00030     bool left = btn_binary & 0x1; //0001
00031     bool middle = btn_binary & 0x2; //0010
00032     bool right = btn_binary & 0x4; //0100
00033     lcd_buttons btns;
00034     btns.left = left ? PRESSED : RELEASED;
00035     btns.middle = middle ? PRESSED : RELEASED;
00036     btns.right = right ? PRESSED : RELEASED;
00037
00038     return btns;
00039 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.63.1.5 `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

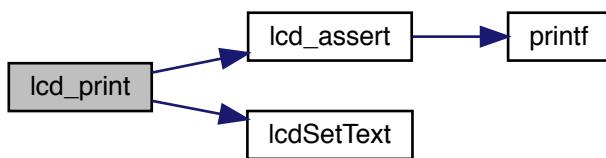
Definition at line **73** of file **lcd.c**.

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

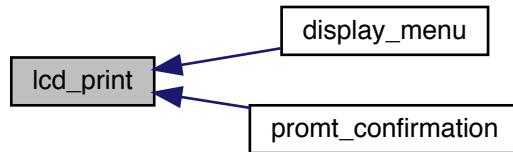
Referenced by `display_menu()`, and `prompt_confirmation()`.

```
00073     {
00074     lcd_assert();
00075     lcdSetText(lcd_port, line, str);
00076 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.63.1.6 `lcd_printf()`

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

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

Parameters

<code>line</code>	the line to print on
<code>format_str</code>	format string string to print

Author

Chris Jerrett

Date

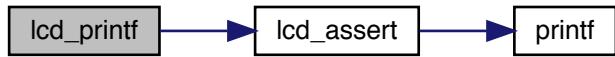
9/9/2017

Definition at line **85** of file **Lcd.c**.

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

```
00085                                         {
00086     lcd_assert();
00087     lcdPrint(lcd_port, line, format_str);
00088 }
```

Here is the call graph for this function:



5.63.1.7 lcd_set_backlight()

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

sets the backlight of the lcd

Parameters

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

Author

Chris Jerrett

Date

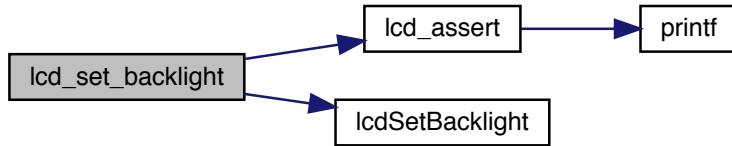
9/9/2017

Definition at line **96** of file **lcd.c**.

References **lcd_assert()**, **lcd_port**, and **LcdSetBacklight()**.

```
00096                               {
00097     lcd_assert();
00098     lcdSetBacklight(lcd_port, state);
00099 }
```

Here is the call graph for this function:



5.63.1.8 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

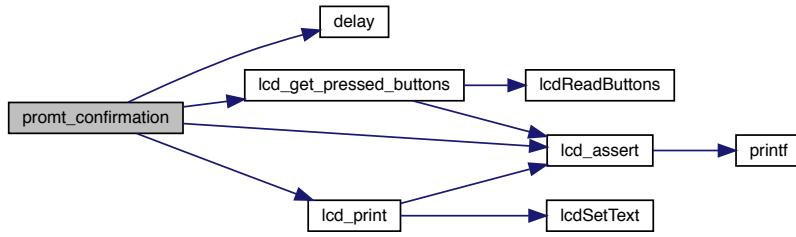
9/9/2017

Definition at line 110 of file **lcd.c**.

References **delay()**, **lcd_assert()**, **lcd_get_pressed_buttons()**, **lcd_print()**, and **PRESSED**.

```
00110
00111     lcd_assert();
00112     lcd_print(1, confirm_text);
00113     while(lcd_get_pressed_buttons().middle != PRESSED) {
00114         delay(200);
00115     }
00116 }
```

Here is the call graph for this function:



5.63.2 Variable Documentation

5.63.2.1 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_printf()**, **Lcd_set_backlight()**, and **Lcd_set_text()**.

5.64 Lcd.c

```

00001 #include "lcd.h"
00002
00004 static FILE *lcd_port = NULL;
00005
00013 static void lcd_assert() {
00014     if(lcd_port != NULL) {
00015         printf("LCD NULL!");
00016         exit(1);
00017     }
00018 }
00019
00027 lcd_buttons lcd_get_pressed_buttons(){
00028     lcd_assert();
00029     unsigned int btn_binary = lcdReadButtons(lcd_port);
00030     bool left = btn_binary & 0x1;//0001
00031     bool middle = btn_binary & 0x2;//0010
00032     bool right = btn_binary & 0x4;//0100
00033     lcd_buttons btns;
00034     btns.left = left ? PRESSSED : RELEASED;
00035     btns.middle = middle ? PRESSSED : RELEASED;
00036     btns.right = right ? PRESSSED : RELEASED;
00037
00038     return btns;
00039 }
00040
00046 void lcd_clear() {
  
```

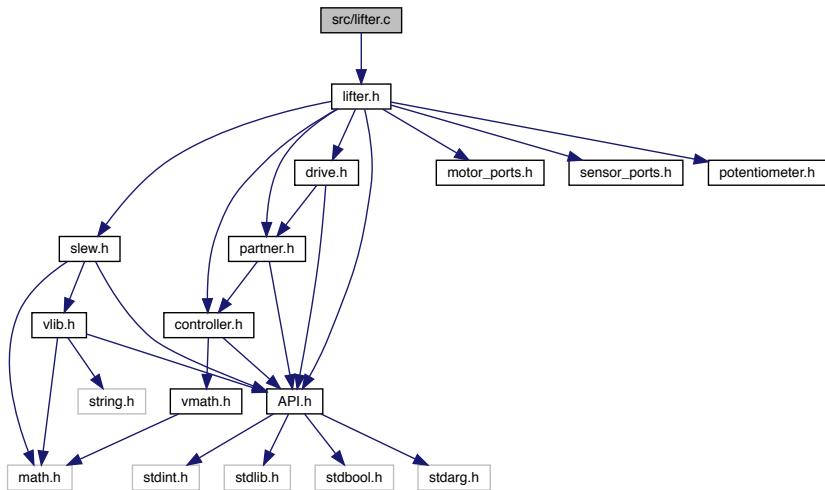
```

00047     lcd_assert();
00048     lcdClear(lcd_port);
00049 }
00050
00060 void init_main_lcd(FILE *lcd) {
00061     lcdInit(lcd);
00062     lcdClear(lcd);
00063     lcd_port = lcd;
00064 }
00065
00073 void lcd_print(unsigned int line, const char *str) {
00074     lcd_assert();
00075     lcdSetText(lcd_port, line, str);
00076 }
00077
00085 void lcd_printf(unsigned int line, const char *format_str, ...) {
00086     lcd_assert();
00087     lcdPrint(lcd_port, line, format_str);
00088 }
00089
00096 void lcd_set_backlight(bool state) {
00097     lcd_assert();
00098     lcdSetBacklight(lcd_port, state);
00099 }
00100
00110 void prompt_confirmation(const char *confirm_text) {
00111     lcd_assert();
00112     lcd_print(1, confirm_text);
00113     while(lcd_get_pressed_buttons().middle != PRESSED) {
00114         delay(200);
00115     }
00116 }

```

5.65 src/lifter.c File Reference

```
#include "lifter.h"
Include dependency graph for lifter.c:
```



Functions

- double **getLifterHeight ()**

- int **getLifterTicks ()**
Gets the value of the lifter pot.
- float **lifterPotentiometerToDegree (int x)**
height of the lifter in degrees from 0 height
- void **lower_lifter ()**
- void **raise_lifter ()**
- void **set_lifter_motors (const int v)**
Sets the lifter motors to the given value.
- void **set_lifter_pos (int pos)**
Sets the lifter positions to the given value.
- void **update_lifter ()**
Updates the lifter in teleop.

5.65.1 Function Documentation

5.65.1.1 getLifterHeight()

double getLifterHeight ()

Gets the height of the lifter in inches.

Returns

the height of the lifter.

Author

Chris Jerrett

Date

9/17/2017

Definition at line 137 of file **lifter.c**.

References **getLifterTicks()**.

```
00137     {
00138     unsigned int ticks = getLifterTicks();
00139     return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) + 0.0198 * ticks + 2.3033);
00140 }
```

Here is the call graph for this function:



5.65.1.2 getLifterTicks()

```
int getLifterTicks ( )
```

Gets the value of the lifter pot.

Returns

the value of the pot.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **126** of file **lifter.c**.

References **analogRead()**, and **LIFTER**.

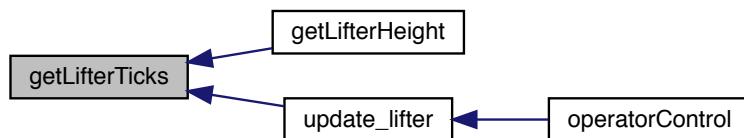
Referenced by **getLifterHeight()**, and **update_lifter()**.

```
00126      {  
00127     return analogRead(LIFTER);  
00128 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.65.1.3 lifterPotentiometerToDegree()

```
float lifterPotentiometerToDegree (
    int x )
```

height of the lifter in degrees from 0 height

Parameters

x	the pot value
---	---------------

Returns

the positions in degrees

Author

Chris Jerrett

Date

10/13/2017

Definition at line **115** of file **lifter.c**.

References **DEG_MAX**, **INIT_ROTATION**, and **TICK_MAX**.

```
00115     {
00116     return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
00117 }
```

5.65.1.4 lower_lifter()

```
void lower_lifter ( )
```

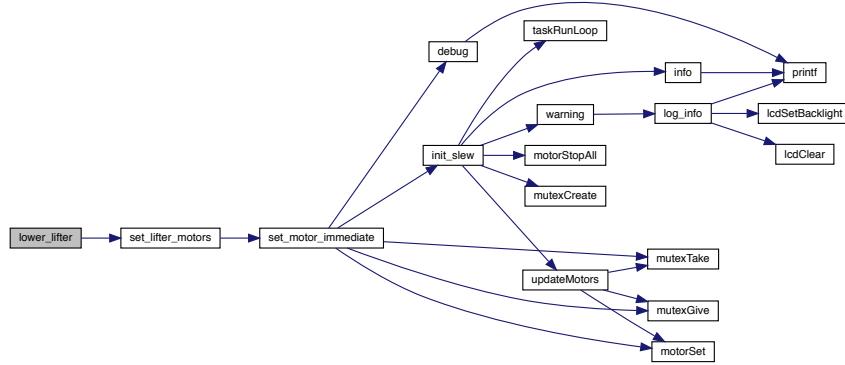
Definition at line **30** of file **lifter.c**.

References **MIN_SPEED**, and **set_lifter_motors()**.

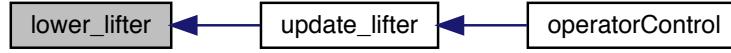
Referenced by **update_lifter()**.

```
00030     {
00031     set_lifter_motors(MIN_SPEED);
00032 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.65.1.5 raise_lifter()

```
void raise_lifter( )
```

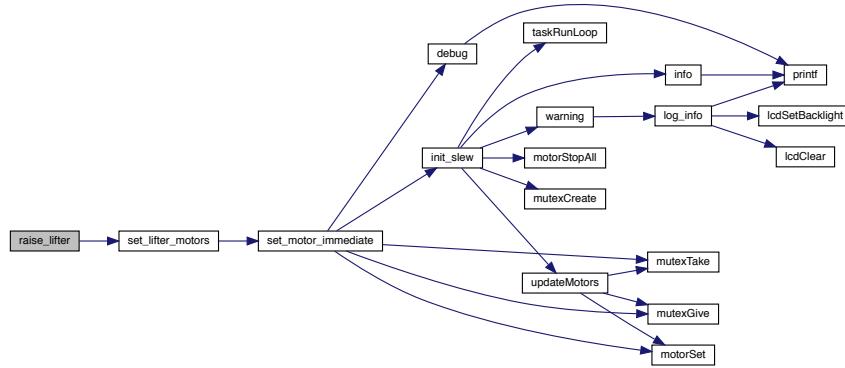
Definition at line **26** of file **lifter.c**.

References **MAX_SPEED**, and **set_lifter_motors()**.

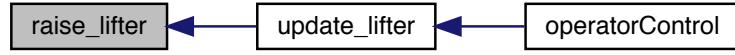
Referenced by **update_lifter()**.

```
00026
00027     set_lifter_motors(MAX_SPEED);
00028 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.65.1.6 set_lifter_motors()

```
void set_lifter_motors (
    const int v )
```

Sets the lifter motors to the given value.

Parameters

v	value for the lifter motor. Between -128 - 127, any values outside are clamped.
---	---

Author

Chris Jerrett

Date

9/9/2017

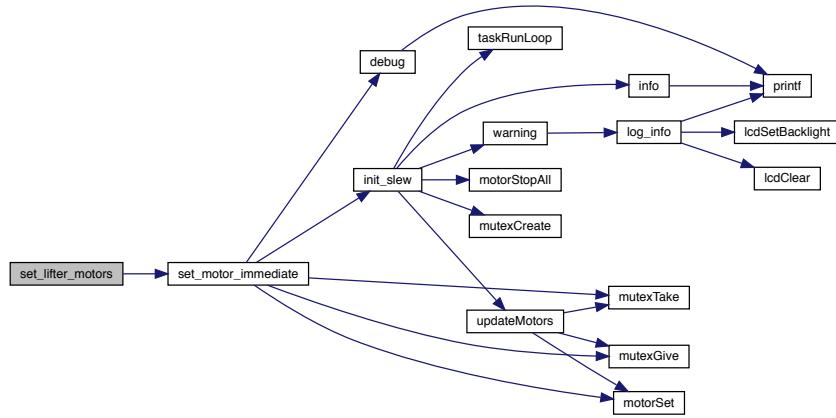
Definition at line 10 of file **lifter.c**.

References **MOTOR_LIFT_TOP_LEFT**, **MOTOR_LIFT_TOP_RIGHT**, and **set_motor_immediate()**.

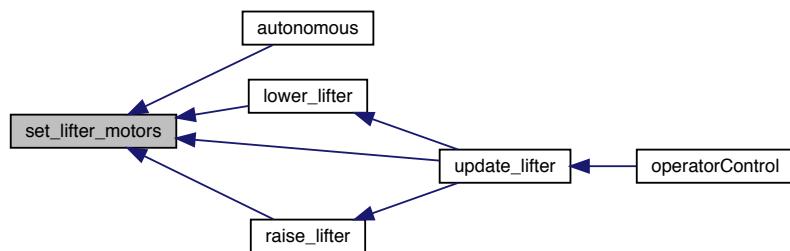
Referenced by **autonomous()**, **lower_lifter()**, **raise_lifter()**, and **update_lifter()**.

```
00010 {
00011     set_motor_immediate(MOTOR_LIFT_TOP_RIGHT, -v);
00012     set_motor_immediate(MOTOR_LIFT_TOP_LEFT, -v);
00013 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.65.1.7 set_lifter_pos()

```
void set_lifter_pos (
    int pos )
```

Sets the lifter positions to the given value.

Parameters

<i>pos</i>	The height in inches
------------	----------------------

Author

Chris Jerrett

Date

9/12/2017

Definition at line **22** of file **lifter.c**.

```
00022          {  
00023  
00024 }
```

5.65.1.8 update_lifter()

```
void update_lifter( )
```

Updates the lifter in teleop.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **40** of file **lifter.c**.

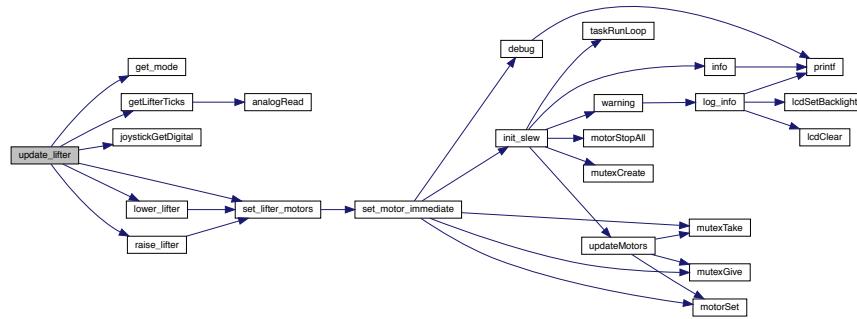
References **get_mode()**, **getLifterTicks()**, **joystickGetDigital()**, **LIFTER_D**, **LIFTER_DOWN**, **LIFTER_DOWN_**
PARTNER, **LIFTER_DRIVER_LOAD**, **LIFTER_I**, **LIFTER_P**, **LIFTER_UP**, **LIFTER_UP_PARTNER**, **lower_lifter()**,
MAIN_CONTROLLER_MODE, **PARTNER_CONTROLLER_MODE**, **raise_lifter()**, **set_lifter_motors()**, and **THR_**
ESHOLD.

Referenced by **operatorControl()**.

```

00040
00041     {
00042         //Establish variables to be used repeatedly
00043         static bool changed = true;
00044         static unsigned int target = 0;
00045         static bool first_run = true;
00046         //Set the target to the current height for the first run
00047         if(first_run) {
00048             target = getLifterTicks();
00049             first_run = false;
00050         }
00051         //Establish the error as 0
00052         static int last_error = 0;
00053         static long long i = 0;
00054         //Check the buttons on the controller indicated by the controller mode
00055         if((joystickGetDigital(LIFTER_UP) && get_mode() == MAIN_CONTROLLER_MODE)
00056             || (joystickGetDigital(LIFTER_UP_PARTNER) && get_mode() ==
00057                 PARTNER_CONTROLLER_MODE)){
00058             changed = true;
00059             i = 0;
00060             //Change the target and start the motion
00061             target = getLifterTicks() + 200;
00062             lower_lifter();
00063         }
00064         else if((joystickGetDigital(LIFTER_DOWN) && get_mode() == MAIN_CONTROLLER_MODE)
00065             || (joystickGetDigital(LIFTER_DOWN_PARTNER) && get_mode() ==
00066                 PARTNER_CONTROLLER_MODE)) {
00067             changed = true;
00068             i = 0;
00069             //Change the target and start the motion
00070             target = getLifterTicks();
00071             raise_lifter();
00072         }
00073         //Raise the lifter to the driver load height
00074         else if(joystickGetDigital(LIFTER_DRIVER_LOAD) && get_mode() ==
00075             MAIN_CONTROLLER_MODE){
00076             changed = true;
00077             i = 0;
00078             int k = 0;
00079             if(getLifterTicks() < 1270){
00080                 lower_lifter();
00081             }
00082             target = 1250;
00083         }
00084         //Change lifter motor values based upon the target
00085         else {
00086             //Don't if we are using the partner controller
00087             if(get_mode() == PARTNER_CONTROLLER_MODE){
00088                 set_lifter_motors(0);
00089                 return;
00090             }
00091             //Define the proportion, derivative, and integral to be used in the motor speed
00092             int p = getLifterTicks() - target;
00093             int d = p - last_error;
00094             last_error = p;
00095             i += p;
00096             int motor = LIFTER_P * p + LIFTER_D * d + LIFTER_I * i;
00097             //Avoid wasting battery if value is insignificant
00098             if (motor < THRESHOLD) {
00099                 set_lifter_motors(0);
00100             } else {
00101                 set_lifter_motors(motor);
00102             }
00103         }
00104     }
00105 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.66 lifter.c

```

00001 #include "lifter.h"
00002
00010 void set_lifter_motors(const int v) {
00011     set_motor_immediate(MOTOR_LIFT_TOP_RIGHT, -v);
00012     set_motor_immediate(MOTOR_LIFT_TOP_LEFT, -v);
00013 }
00014
00022 void set_lifter_pos(int pos) {
00023
00025
00026 void raise_lifter() {
00027     set_lifter_motors(MAX_SPEED);
00028 }
00029
00030 void lower_lifter() {
00031     set_lifter_motors(MIN_SPEED);
00032 }
00033
00040 void update_lifter() {
00041     //Establish variables to be used repeatedly
00042     static bool changed = true;
00043     static unsigned int target = 0;
00044     static bool first_run = true;
00045     //Set the target to the current height for the first run
00046     if(first_run) {
00047         target = getLifterTicks();
00048         first_run = false;
00049     }
00050     //Establish the error as 0
00051     static int last_error = 0;
  
```

```

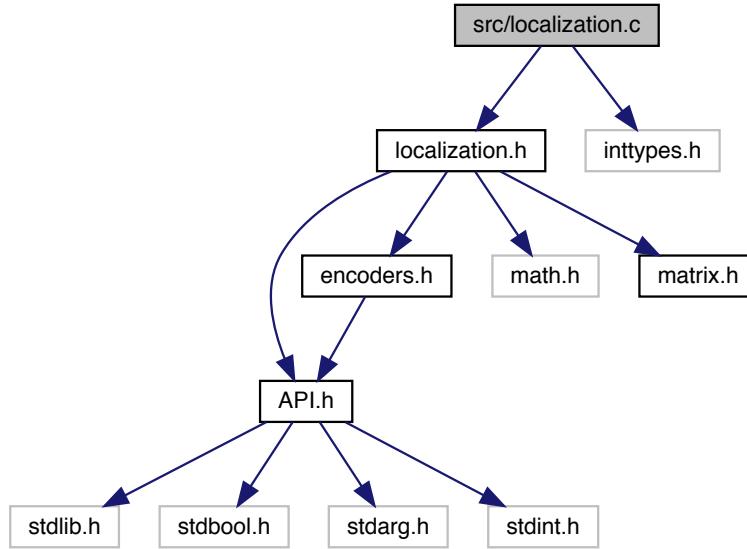
00052 static long long i = 0;
00053 //Check the buttons on the controller indicated by the controller mode
00054 if((joystickGetDigital(LIFTER_UP) && get_mode() == MAIN_CONTROLLER_MODE)
00055 || (joystickGetDigital(LIFTER_UP_PARTNER) && get_mode() ==
PARTNER_CONTROLLER_MODE)){
00056     changed = true;
00057     i = 0;
00058     //Change the target and start the motion
00059     target = getLifterTicks() + 200;
00060     lower_lifter();
00061 }
00062 else if((joystickGetDigital(LIFTER_DOWN) && get_mode() == MAIN_CONTROLLER_MODE)
00063 || (joystickGetDigital(LIFTER_DOWN_PARTNER) && get_mode() ==
PARTNER_CONTROLLER_MODE)){
00064     changed = true;
00065     i = 0;
00066     //Change the target and start the motion
00067     target = getLifterTicks();
00068     raise_lifter();
00069 }
00070 //Raise the lifter to the driver load height
00071 else if(joystickGetDigital(LIFTER_DRIVER_LOAD) && get_mode() ==
MAIN_CONTROLLER_MODE){
00072     changed = true;
00073     i = 0;
00074     int k = 0;
00075     if(getLifterTicks() < 1270){
00076         lower_lifter();
00077     }
00078     if(getLifterTicks() > 1230){
00079         raise_lifter();
00080     }
00081     target = 1250;
00082 }
00083 //Change lifter motor values based upon the target
00084 else {
00085     //Don't if we are using the partner controller
00086     if(get_mode() == PARTNER_CONTROLLER_MODE){
00087         set_lifter_motors(0);
00088         return;
00089     }
00090     //Define the proportion, derivative, and integral to be used in the motor speed
00091     int p = getLifterTicks() - target;
00092     int d = p - last_error;
00093     last_error = p;
00094     i += p;
00095     int motor = LIFTER_P * p + LIFTER_D * d + LIFTER_I * i;
00096     //Avoid wasting battery if value is insignificant
00097     if (motor < THRESHOLD) {
00098         set_lifter_motors(0);
00099     } else {
00100         set_lifter_motors(motor);
00101     }
00102 }
00103 }
00104 }
00105 }
00106
00115 float lifterPotentiometerToDegree(int x){
00116     return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
00117 }
00118
00126 int getLifterTicks() {
00127     return analogRead(LIFTER);
00128 }
00129
00137 double getLifterHeight() {
00138     unsigned int ticks = getLifterTicks();
00139     return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) + 0.0198 * ticks + 2.3033);
00140 }

```

5.67 src/localization.c File Reference

```
#include "localization.h"
#include <inttypes.h>
```

Include dependency graph for localization.c:



Data Structures

- struct `accelerometer_odometry`
- struct `encoder_odemtry`

Functions

- static struct `accelerometer_odometry` `calculate_accelerometer_odometry ()`
- static double `calculate_angle ()`
- static double `calculate_gryo_angular_velocity ()`
- struct `location` `get_position ()`

Gets the current position of the robot.
- bool `init_localization` (const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int start_theta)

Starts the localization process.
- static double `integrate_gyro_w` (int new_w)
- void `update_position ()`

Variables

- static `Gyro` `g1`
- static int `last_call` = 0
- static `TaskHandle` `localization_task`
- `matrix *` `state_matrix`

5.67.1 Function Documentation

5.67.1.1 calculate_accelerometer_odometry()

```
static struct accelerometer_odometry calculate_accelerometer_odometry () [static]
```

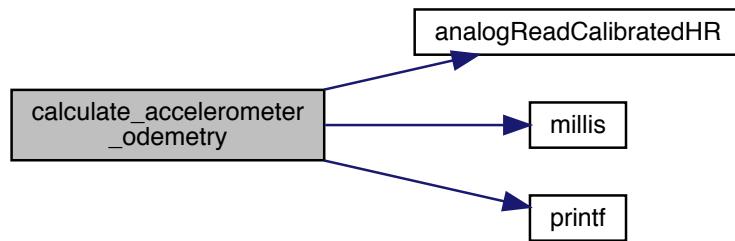
Definition at line 49 of file **localization.c**.

References **analogReadCalibratedHR()**, **last_call**, **millis()**, and **printf()**.

Referenced by **update_position()**.

```
00049
00050     static double vel_acumm_x = 0;
00051     static double vel_acumm_y = 0;
00052
00053     int32_t accel_x_rel = (int32_t) analogReadCalibratedHR(2);
00054     int32_t accel_y_rel = (int32_t) analogReadCalibratedHR(3);
00055
00056     //Ignore atom format string errors
00057     printf("x: %" PRId32 " y: %" PRId32 "\n", accel_x_rel, accel_y_rel);
00058
00059     double delta_time = ((millis() - last_call)/1000.0);
00060     //double accel_x_abs = (accel_x_rel * cos(theta) + accel_y_rel * sin(theta)) * delta_time;
00061     //double accel_y_abs = (accel_y_rel * cos(theta) + accel_x_rel * sin(theta)) * delta_time;
00062
00063     //vel_acumm_x += accel_x_abs;
00064     //vel_acumm_y += accel_y_abs;
00065
00066     //double new_x = x + vel_acumm_x * delta_time;
00067     //double new_y = y + vel_acumm_y * delta_time;
00068
00069     struct accelerometer_odometry od;
00070     //od.x = new_x;
00071     //od.y = new_y;
00072     return od;
00073 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.67.1.2 calculate_angle()

```
static double calculate_angle ( ) [static]
```

5.67.1.3 calculate_gryo_angular_velocity()

```
static double calculate_gryo_angular_velocity ( ) [static]
```

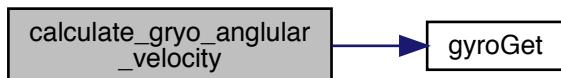
Definition at line 81 of file **localization.c**.

References **g1**, **gyroGet()**, and **LOCALIZATION_UPDATE_FREQUENCY**.

```

00081
00082     static int last_gyro = 0;
00083     int current = gyroGet(g1);
00084     // Calculate w (angular velocity in degrees per second)
00085     double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY/1000.0);
00086     return w;
00087 }
```

Here is the call graph for this function:



5.67.1.4 get_position()

```
struct location get_position ( )
```

Gets the current position of the robot.

Parameters

<i>gyro1</i>	The first gyro
--------------	----------------

Returns

the location of the robot as a struct.

Definition at line **25** of file **localization.c**.

```
00025
00026
00027 }
```

5.67.1.5 init_localization()

```
bool init_localization (
    const unsigned char gyro1,
    unsigned short multiplier,
    int start_x,
    int start_y,
    int start_theta )
```

Starts the localization process.

Author

Chris Jerrett

Parameters

<i>gyro1</i>	The first gyro The multiplier parameter can tune the gyro to adapt to specific sensors. The default value at this time is 196; higher values will increase the number of degrees reported for a fixed actual rotation, while lower values will decrease the number of degrees reported. If your robot is consistently turning too far, increase the multiplier, and if it is not turning far enough, decrease the multiplier.
--------------	---

Definition at line **89** of file **localization.c**.

References **g1**, **gyroInit()**, **last_call**, **localization_task**, **LOCALIZATION_UPDATE_FREQUENCY**, **makeMatrix()**, **millis()**, **taskRunLoop()**, and **update_position()**.

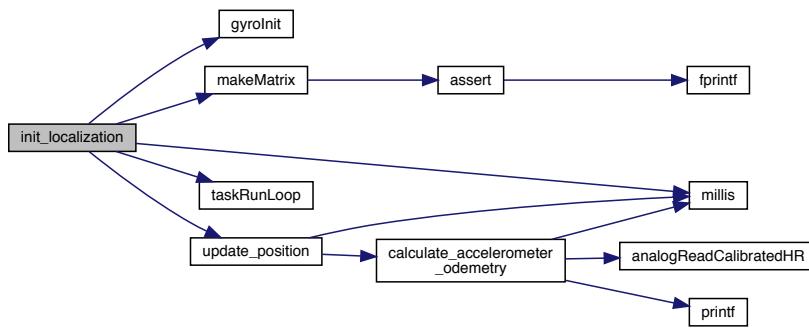
```
00089
00090     g1 = gyroInit(gyro1, multiplier);
00091 //init state matrix
00092
00093 //one dimensional vector with x, y, theta, acceleration in x and y
```

```

00094     state_matrix = makeMatrix(1, 5);
00095     localization_task = taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
00096     last_call = millis();
00097     return true;
00098 }

```

Here is the call graph for this function:



5.67.1.6 integrate_gyro_w()

```

static double integrate_gyro_w (
    int new_w ) [static]

```

Definition at line 75 of file **localization.c**.

References **LOCALIZATION_UPDATE_FREQUENCY**, and **encoder_odemtry::theta**.

```

00075
00076     static double theta = 0;
00077     double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;
00078     theta += delta_theta;
00079 }

```

5.67.1.7 update_position()

```

void update_position ( )

```

Definition at line 29 of file **localization.c**.

References **calculate_accelerometer_odometry()**, **last_call**, and **millis()**.

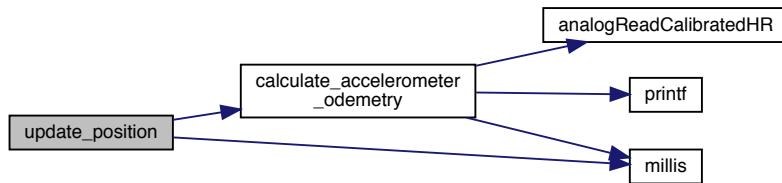
Referenced by **init_localization()**.

```

00029
00030     //int curr_theta = calculate_angle();
00031
00032     struct accelerometer_odometry odem = calculate_accelerometer_odometry();
00033     //printf("x: %d y: %d T: %d\n", a.x, a.y, 0);
00034
00035     /*int l = 1;
00036     int vr = get_encoder_velocity(1);
00037     int vl = get_encoder_velocity(2);
00038     int theta_dot = (vr - vl) / l;
00039     int curr_theta = theta + theta_dot;
00040     double dt = LOCALIZATION_UPDATE_FREQUENCY;
00041     double v_tot = (vr+vl)/2.0;
00042     int x_curr = x - v_tot*dt*sin(curr_theta);
00043     int y_curr = y + v_tot*dt*cos(curr_theta);
00044     x = x_curr;
00045     y = y_curr; */
00046     last_call = millis();
00047 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.67.2 Variable Documentation

5.67.2.1 g1

Gyro g1 [static]

Definition at line 4 of file **localization.c**.

Referenced by **calculate_gryo_angular_velocity()**, and **init_localization()**.

5.67.2.2 last_call

```
int last_call = 0 [static]
```

Definition at line 7 of file **localization.c**.

Referenced by **calculate_accelerometer_odometry()**, **init_localization()**, and **update_position()**.

5.67.2.3 localization_task

```
TaskHandle localization_task [static]
```

Definition at line 5 of file **localization.c**.

Referenced by **init_localization()**.

5.67.2.4 state_matrix

```
matrix* state_matrix
```

Definition at line 9 of file **localization.c**.

5.68 localization.c

```
00001 #include "localization.h"
00002 #include <inttypes.h>
00003
00004 static Gyro g1;
00005 static TaskHandle localization_task;
00006
00007 static int last_call = 0;
00008
00009 matrix *state_matrix;
00010
00011 struct encoder_odemtry {
00012     double x;
00013     double y;
00014     double theta;
00015 };
00016
00017 struct accelerometer_odometry {
00018     double x;
00019     double y;
00020 };
00021
00022 static double calculate_angle();
00023 static struct accelerometer_odometry calculate_accelerometer_odometry();
00024
00025 struct location get_position() {
00026
00027 }
00028
00029 void update_position() {
00030     //int curr_theta = calculate_angle();
00031
00032     struct accelerometer_odometry odem = calculate_accelerometer_odometry();
00033     //printf("x: %d y: %d T: %d\n", a.x, a.y, 0);
00034 }
```

```

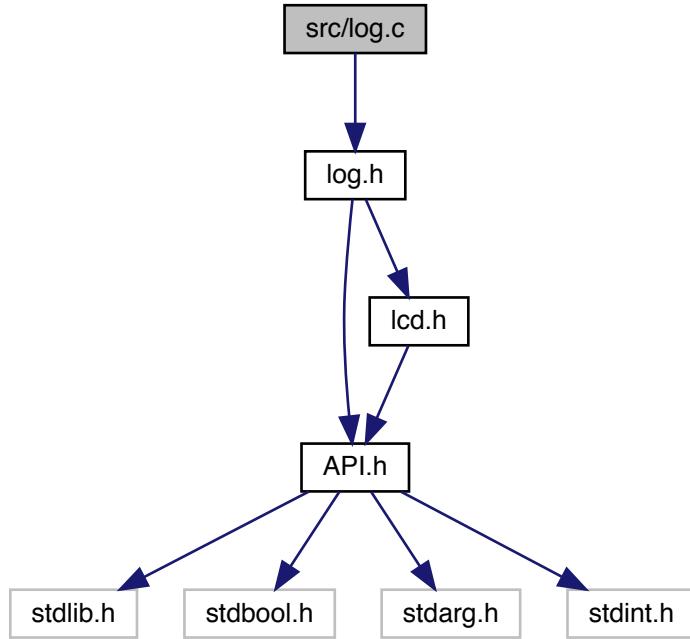
00035 /*int l = 1;
00036 int vr = get_encoder_velocity(1);
00037 int vl = get_encoder_velocity(2);
00038 int theta_dot = (vr - vl) / l;
00039 int curr_theta = theta + theta_dot;
00040 double dt = LOCALIZATION_UPDATE_FREQUENCY;
00041 double v_tot = (vr+vl)/2.0;
00042 int x_curr = x - v_tot*dt*sin(curr_theta);
00043 int y_curr = y + v_tot*dt*cos(curr_theta);
00044 x = x_curr;
00045 y = y_curr; */
00046 last_call = millis();
00047 }
00048
00049 static struct accelerometer_odometry calculate_accelerometer_odometry() {
00050     static double vel_acumm_x = 0;
00051     static double vel_acumm_y = 0;
00052
00053     int32_t accel_x_rel = (int32_t) analogReadCalibratedHR(2);
00054     int32_t accel_y_rel = (int32_t) analogReadCalibratedHR(3);
00055
00056 //Ignore atom format string errors
00057 printf("x: %" PRId32 " y: %" PRId32 "\n", accel_x_rel, accel_y_rel);
00058
00059     double delta_time = ((millis() - last_call)/1000.0);
00060 //double accel_x_abs = (accel_x_rel * cos(theta) + accel_y_rel * sin(theta)) * delta_time;
00061 //double accel_y_abs = (accel_y_rel * cos(theta) + accel_x_rel * sin(theta)) * delta_time;
00062
00063 //vel_acumm_x += accel_x_abs;
00064 //vel_acumm_y += accel_y_abs;
00065
00066 //double new_x = x + vel_acumm_x * delta_time;
00067 //double new_y = y + vel_acumm_y * delta_time;
00068
00069     struct accelerometer_odometry od;
00070 //od.x = new_x;
00071 //od.y = new_y;
00072     return od;
00073 }
00074
00075 static double integrate_gyro_w(int new_w) {
00076     static double theta = 0;
00077     double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;
00078     theta += delta_theta;
00079 }
00080
00081 static double calculate_gryo_angular_velocity() {
00082     static int last_gyro = 0;
00083     int current = gyroGet(g1);
00084 // Calculate w (angular velocity in degrees per second)
00085     double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY/1000.0);
00086     return w;
00087 }
00088
00089 bool init_localization(const unsigned char gyrol, unsigned short multiplier, int start_x, int start_y, int
start_theta) {
00090     g1 = gyroInit(gyrol, multiplier);
00091 //init state matrix
00092
00093 //one dimensional vector with x, y, theta, acceleration in x and y
00094 state_matrix = makeMatrix(1, 5);
00095 localization_task = taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
00096 last_call = millis();
00097     return true;
00098 }

```

5.69 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** = **INFO**

5.69.1 Function Documentation

5.69.1.1 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 (p. 321)

Parameters

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

Definition at line 37 of file **log.c**.

References **ERROR**, **log_level**, and **printf()**.

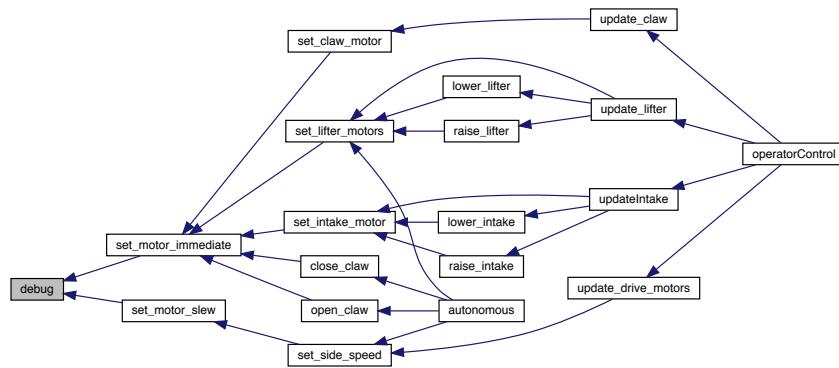
Referenced by **set_motor_immediate()**, and **set_motor_slew()**.

```
00037
00038     if(log_level>ERROR) {
00039         printf("[INFO]: %s\n", debug_message);
00040     }
00041 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.69.1.2 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 (p. 321)

Author

Chris Jerrett

Date

9/10/17

Parameters

<code>error_message</code>	the message
----------------------------	-------------

Definition at line **21** of file **log.c**.

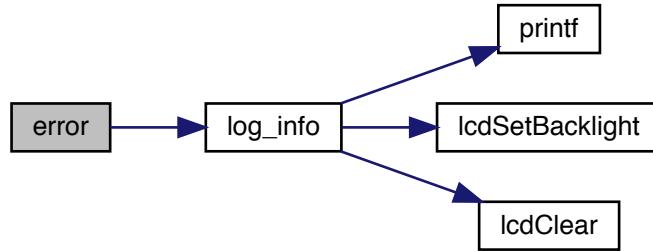
References **log_info()**, **log_level**, and **NONE**.

Referenced by **create_menu()**.

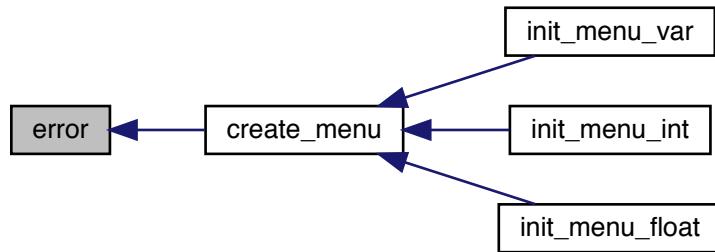
```

00021
00022     if(log_level>NONE)
00023         log_info("ERROR", error_message);
00024 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.69.1.3 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 (p. 321)

Parameters

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

Definition at line 31 of file **log.c**.

References **ERROR**, **log_level**, and **printf()**.

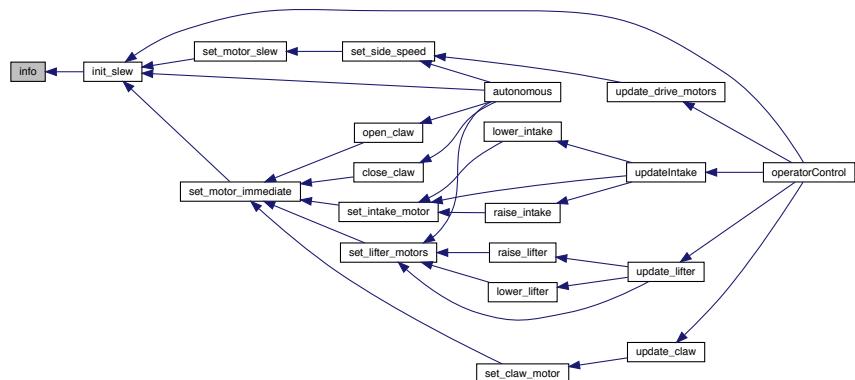
Referenced by **init_slew()**.

```
00031      {  
00032      if(log_level>ERROR) {  
00033          printf("[INFO]: %s\n", info_message);  
00034      }  
00035 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.69.1.4 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 **lcdInit()**, and **log_lcd**.

```
00006
00007     if(use_lcd) {
00008         lcdInit(lcd);
00009         log_lcd = lcd;
00010     }
00011 }
```

Here is the call graph for this function:



5.69.1.5 log_info()

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

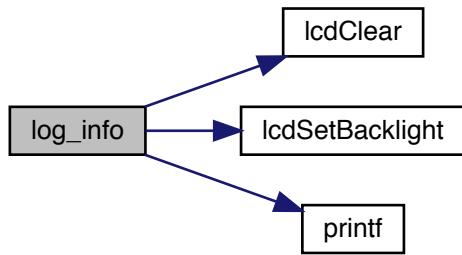
Definition at line 13 of file **log.c**.

References **BOTTOM_ROW**, **LcdClear()**, **LcdSetBacklight()**, **log_lcd**, **printf()**, and **TOP_ROW**.

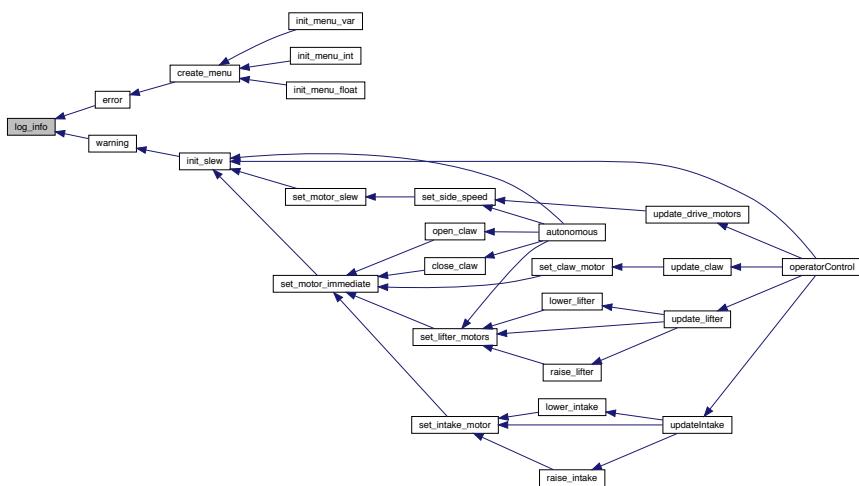
Referenced by **error()**, and **warning()**.

```
00013 {
00014     printf("[%s]: %s\n", s, mess);
00015     lcdSetBacklight(log_lcd, true);
00016     lcdClear(log_lcd);
00017     lcdPrint(log_lcd, TOP_ROW, s);
00018     lcdPrint(log_lcd, BOTTOM_ROW, mess);
00019 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.69.1.6 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](#) (p. 321)

Author

Chris Jerrett

Date

9/10/17

Parameters

<code>warning_message</code>	the message
------------------------------	-------------

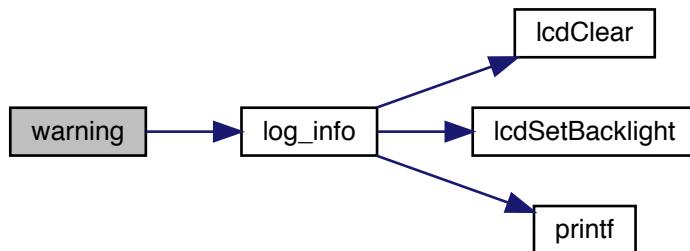
Definition at line **26** of file **log.c**.

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

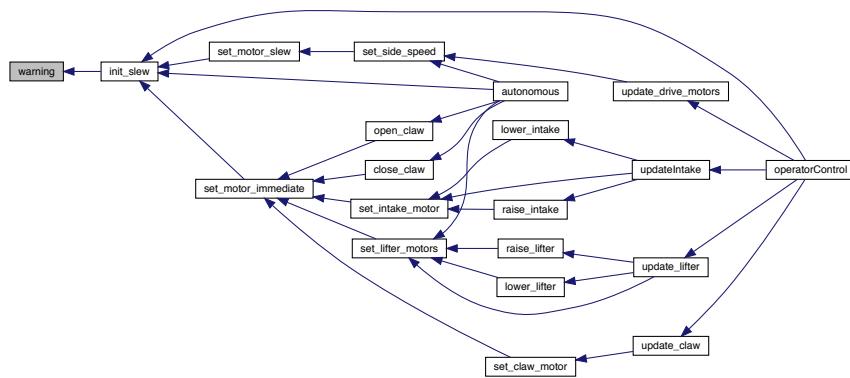
Referenced by [init_slew\(\)](#).

```
00026
00027     if(log_level>WARNING)
00028         log_info("WARNING", warning_message);
00029 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.69.2 Variable Documentation

5.69.2.1 log_lcd

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

Definition at line 4 of file **log.c**.

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

5.69.2.2 log_level

```
unsigned int log_level = INFO
```

Definition at line 3 of file **log.c**.

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

5.70 log.c

```

00001 #include "log.h"
00002
00003 unsigned int log_level = INFO;
00004 static FILE *log_lcd = NULL;
00005
00006 void init_error(bool use_lcd, FILE *lcd) {
00007     if(use_lcd) {
00008         lcdInit(lcd);
00009         log_lcd = lcd;
00010     }
00011 }
00012
00013 static void log_info(const char *s, const char *mess) {
00014     printf("[%s]: %s\n", s, mess);
00015     lcdSetBacklight(log_lcd, true);
00016     lcdClear(log_lcd);
00017     lcdPrint(log_lcd, TOP_ROW, s);
00018     lcdPrint(log_lcd, BOTTOM_ROW, mess);
00019 }
00020
00021 void error(const char *error_message) {
00022     if(log_level>NONE)
00023         log_info("ERROR", error_message);
00024 }
00025
00026 void warning(const char *warning_message) {
00027     if(log_level>WARNING)
00028         log_info("WARNING", warning_message);
00029 }
00030
00031 void info(const char *info_message) {
00032     if(log_level>ERROR) {
00033         printf("[INFO]: %s\n", info_message);
00034     }
00035 }
00036
00037 void debug(const char *debug_message) {
00038     if(log_level>ERROR) {
00039         printf("[INFO]: %s\n", debug_message);
00040     }
00041 }

```

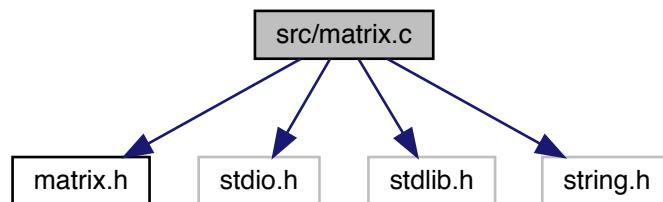
5.71 src/matrix.c File Reference

```

#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

Include dependency graph for matrix.c:



Functions

- **void assert (int assertion, const char *message)**

Asserts a condition is true.
- **matrix * copyMatrix (matrix *m)**

Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.
- **matrix * covarianceMatrix (matrix *m)**

returns the covariance of the matrix
- **matrix * dotDiagonalMatrix (matrix *a, matrix *b)**

performs a diagonal matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a $a \times a$ matrix of the cross product of each matrix along the diagonal.
- **matrix * dotProductMatrix (matrix *a, matrix *b)**

returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a $a \times b$ matrix of the cross product of each matrix.
- **void freeMatrix (matrix *m)**

Frees the resources of a matrix.
- **matrix * identityMatrix (int n)**

Returns an identity matrix of size $n \times n$.
- **matrix * makeMatrix (int width, int height)**

Makes a matrix with a width and height parameters.
- **matrix * meanMatrix (matrix *m)**

Given an " m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.
- **matrix * multiplyMatrix (matrix *a, matrix *b)**

Given a two matrices, returns the multiplication of the two.
- **void printMatrix (matrix *m)**

Prints a matrix.
- **void rowSwap (matrix *a, int p, int q)**

swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q , provided that p and q are less than or equal to the height of matrix A and p and q are different values.
- **matrix * scaleMatrix (matrix *m, double value)**

scales a matrix.
- **double traceMatrix (matrix *m)**

Given an " m rows by n columns" matrix returns the sum.
- **matrix * transposeMatrix (matrix *m)**

returns the transpose matrix.

5.71.1 Function Documentation

5.71.1.1 assert()

```
void assert (
    int assertion,
    const char * message )
```

Asserts a condition is true.

If the assertion is non-zero (i.e. true), then it returns. If the assertion is zero (i.e. false), then it display the string and aborts the program. This is ment to act like Python's assert keyword.

Definition at line **14** of file **matrix.c**.

References **fprintf()**.

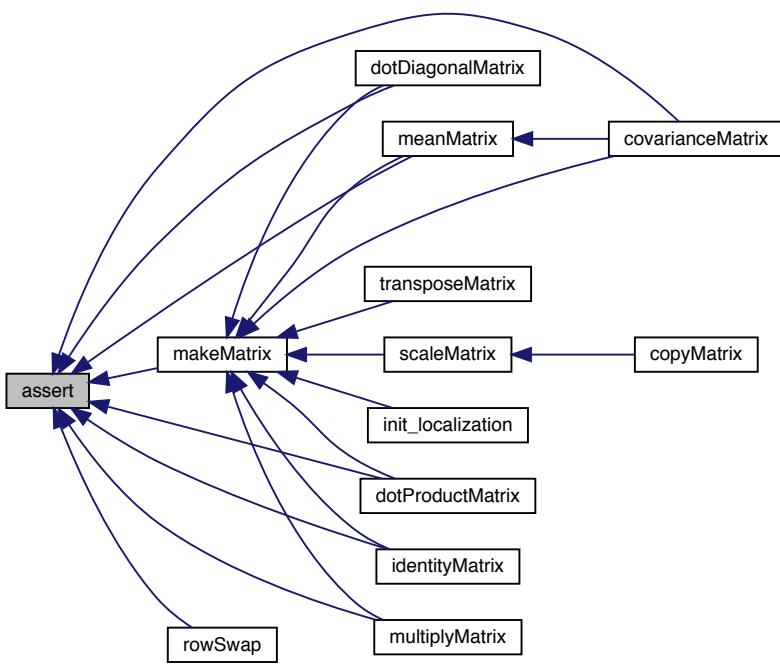
Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, and **rowSwap()**.

```
00014     {
00015     if (assertion == 0) {
00016         fprintf(stderr, "%s\n", message);
00017         exit(1);
00018     }
00019 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.71.1.2 copyMatrix()

```
matrix* copyMatrix (
    matrix * m )
```

Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.

Parameters

<i>m</i>	a pointer to the matrix
----------	-------------------------

Returns

a copied matrix

Definition at line 52 of file **matrix.c**.

References **scaleMatrix()**.

```

00052     {
00053         return scaleMatrix(m, 1);
00054     }

```

Here is the call graph for this function:



5.71.1.3 covarianceMatrix()

```

matrix* covarianceMatrix (
    matrix * m )

```

returns the covariance of the matrix

Parameters

<i>the</i>	matrix
------------	--------

Returns

a matrix with n row and n columns, where each element represents covariance of 2 columns.

Definition at line **168** of file **matrix.c**.

References **assert()**, **_matrix::data**, **freeMatrix()**, **_matrix::height**, **makeMatrix()**, **meanMatrix()**, and **_matrix::width**.

```

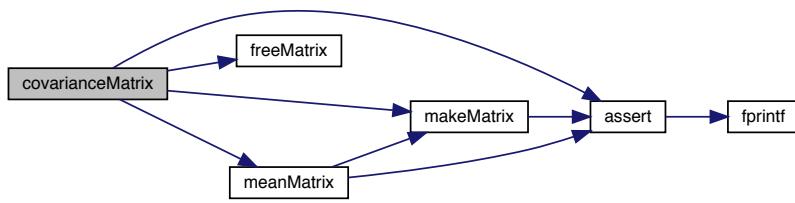
00168     {
00169         int i, j, k = 0;
00170         matrix* out;
00171         matrix* mean;
00172         double* ptrA;
00173         double* ptrB;
00174         double* ptrOut;
00175
00176         assert(m->height > 1, "Height of matrix cannot be zero or one.");
00177
00178         mean = meanMatrix(m);
00179         out = makeMatrix(m->width, m->width);
00180         ptrOut = out->data;
00181
00182         for (i = 0; i < m->width; i++) {
00183             for (j = 0; j < m->width; j++) {
00184                 ptrA = &m->data[i];
00185                 ptrB = &m->data[j];
00186                 *ptrOut = 0.0;
00187                 for (k = 0; k < m->height; k++) {
00188                     *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);

```

```

00189         ptrA += m->width;
00190         ptrB += m->width;
00191     }
00192     *ptrOut /= m->height - 1;
00193     ptrOut++;
00194 }
00195 }
00196
00197 freeMatrix(mean);
00198 return out;
00199 }
```

Here is the call graph for this function:



5.71.1.4 dotDiagonalMatrix()

```

matrix* dotDiagonalMatrix (
    matrix * a,
    matrix * b )
```

performs a diagonal matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a 1 by a->height matrix of the cross product of each matrix along the diagonal.

Dot product is essentially the sum-of-squares of two vectors.

If the second parameter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

<i>a</i>	the first matrix
<i>b</i>	the second matrix

Returns

the matrix result

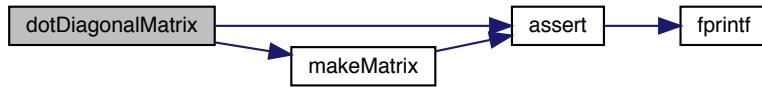
Definition at line 385 of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```

00385
00386     matrix* out;
00387     double* ptrOut;
00388     double* ptrA;
00389     double* ptrB;
00390     int i, j;
00391
00392     if (b != NULL) {
00393         assert(a->width == b->width && a->height == b->height, "Matrices must be of the same
00394         dimensionality.");
00395     }
00396
00397     // Are we computing the sum of squares of the same matrix?
00398     if (a == b || b == NULL) {
00399         b = a; // May not appear safe, but we can do this without risk of losing b.
00400     }
00401
00402     out = makeMatrix(1, a->height);
00403     ptrOut = out->data;
00404     ptrA = a->data;
00405     ptrB = b->data;
00406
00407     for (i = 0; i < a->height; i++) {
00408         *ptrOut = 0;
00409         for (j = 0; j < a->width; j++) {
00410             *ptrOut += *ptrA * *ptrB;
00411             ptrA++;
00412             ptrB++;
00413         }
00414         ptrOut++;
00415     }
00416
00417     return out;
00418 }
```

Here is the call graph for this function:



5.71.1.5 dotProductMatrix()

```

matrix* dotProductMatrix (
    matrix * a,
    matrix * b )
```

returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a a->height by b->height matrix of the cross product of each matrix.

Dot product is essentially the sum-of-squares of two vectors.

Also, if the second parameter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

<i>a</i>	the first matrix
<i>the</i>	second matrix

Returns

the result of the dot product

Definition at line 333 of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```

00333
00334     matrix* out;
00335     double* ptrOut;
00336     double* ptrA;
00337     double* ptrB;
00338     int i, j, k;
00339
00340     if (b != NULL) {
00341         assert(a->width == b->width, "Matrices must be of the same dimensionality.");
00342     }
00343
00344     // Are we computing the sum of squares of the same matrix?
00345     if (a == b || b == NULL) {
00346         b = a; // May not appear safe, but we can do this without risk of losing b.
00347     }
00348
00349     out = makeMatrix(b->height, a->height);
00350     ptrOut = out->data;
00351
00352     for (i = 0; i < a->height; i++) {
00353         ptrB = b->data;
00354
00355         for (j = 0; j < b->height; j++) {
00356             ptrA = &a->data[ i * a->width ];
00357
00358             *ptrOut = 0;
00359             for (k = 0; k < a->width; k++) {
00360                 *ptrOut += *ptrA * *ptrB;
00361                 ptrA++;
00362                 ptrB++;
00363             }
00364             ptrOut++;
00365         }
00366     }
00367
00368     return out;
00369 }
```

Here is the call graph for this function:



5.71.1.6 freeMatrix()

```
void freeMatrix (
    matrix * m )
```

Frees the resources of a matrix.

Parameters

<i>the</i>	matrix to free
------------	----------------

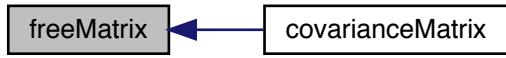
Definition at line **60** of file **matrix.c**.

References **_matrix::data**.

Referenced by **covarianceMatrix()**.

```
00060
00061     if (m != NULL) {
00062         if (m->data != NULL) {
00063             free(m->data);
00064             m->data = NULL;
00065         }
00066         free(m);
00067     }
00068     return;
00069 }
```

Here is the caller graph for this function:



5.71.1.7 identityMatrix()

```
matrix* identityMatrix (
    int n )
```

Returns an identity matrix of size n by n.

Parameters

<i>n</i>	the input matrix.
----------	-------------------

Returns

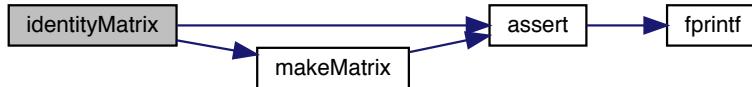
the identity matrix parameter.

Definition at line 94 of file **matrix.c**.

References **assert()**, **_matrix::data**, and **makeMatrix()**.

```
00094             {  
00095     int i;  
00096     matrix *out;  
00097     double* ptr;  
00098  
00099     assert(n > 0, "Identity matrix must have value greater than zero.");  
00100  
00101     out = makeMatrix(n, n);  
00102     ptr = out->data;  
00103     for (i = 0; i < n; i++) {  
00104         *ptr = 1.0;  
00105         ptr += n + 1;  
00106     }  
00107  
00108     return out;  
00109 }
```

Here is the call graph for this function:



5.71.1.8 makeMatrix()

```
matrix* makeMatrix (  
    int width,  
    int height )
```

Makes a matrix with a width and height parameters.

Parameters

<i>width</i>	The width of the matrix
<i>height</i>	the height of the matrix

Returns

the new matrix

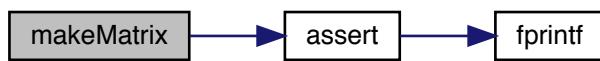
Definition at line **27** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, and **_matrix::width**.

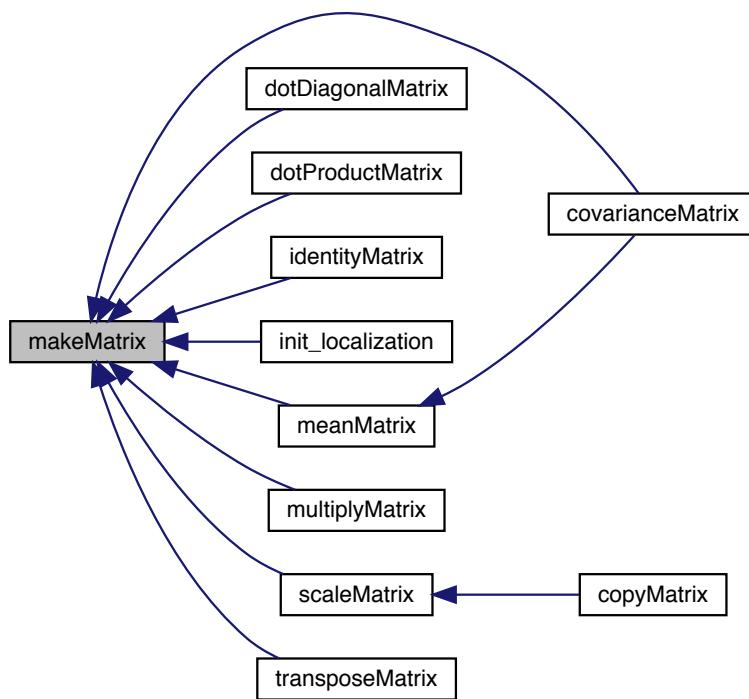
Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **init_localization()**, **meanMatrix()**, **multiplyMatrix()**, **scaleMatrix()**, and **transposeMatrix()**.

```
00027         {
00028     matrix* out;
00029     assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
00030     out = (matrix*) malloc(sizeof(matrix));
00031     assert(out != NULL, "Out of memory.");
00032     out->width = width;
00033     out->height = height;
00034     out->data = (double*) malloc(sizeof(double) * width * height);
00035     assert(out->data != NULL, "Out of memory.");
00036     memset(out->data, 0.0, width * height * sizeof(double));
00037     return out;
00038 }
00039
00040 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.71.1.9 meanMatrix()

```
matrix* meanMatrix (
    matrix * m )
```

Given an "m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.

Returns

matrix with 1 row and n columns each element represents the mean of that full column.

Definition at line **142** of file **matrix.c**.

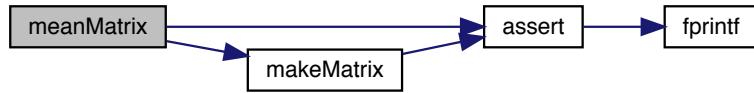
References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **covarianceMatrix()**.

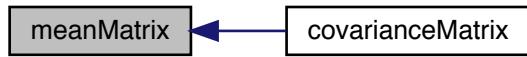
```

00142
00143     int i, j;
00144     matrix* out;
00145
00146     assert(m->height > 0, "Height of matrix cannot be zero.");
00147
00148     out = makeMatrix(m->width, 1);
00149
00150     for (i = 0; i < m->width; i++) {
00151         double* ptr;
00152         out->data[i] = 0.0;
00153         ptr = &m->data[i];
00154         for (j = 0; j < m->height; j++) {
00155             out->data[i] += *ptr;
00156             ptr += out->width;
00157         }
00158         out->data[i] /= (double) m->height;
00159     }
00160     return out;
00161 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.71.1.10 multiplyMatrix()

```

matrix* multiplyMatrix (
    matrix * a,
    matrix * b )
```

Given a two matrices, returns the multiplication of the two.

Parameters

<i>a</i>	the first matrix
<i>b</i>	the seconf matrix return the result of the multiplication

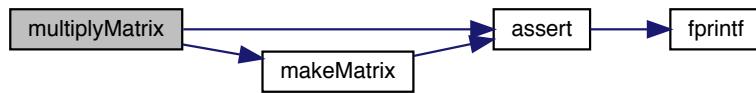
Definition at line 230 of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```

00230
00231     int i, j, k;
00232     matrix* out;
00233     double* ptrOut;
00234     double* ptrA;
00235     double* ptrB;
00236
00237     assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");
00238
00239     out = makeMatrix(b->width, a->height);
00240     ptrOut = out->data;
00241
00242     for (i = 0; i < a->height; i++) {
00243
00244         for (j = 0; j < b->width; j++) {
00245             ptrA = &a->data[ i * a->width ];
00246             ptrB = &b->data[ j ];
00247
00248             *ptrOut = 0;
00249             for (k = 0; k < a->width; k++) {
00250                 *ptrOut += *ptrA * *ptrB;
00251                 ptrA++;
00252                 ptrB += b->width;
00253             }
00254             ptrOut++;
00255         }
00256     }
00257
00258     return out;
00259 }
```

Here is the call graph for this function:



5.71.1.11 printMatrix()

```
void printMatrix (
    matrix * m )
```

Prints a matrix.

Parameters

<i>the</i>	matrix
------------	--------

Definition at line **75** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **printf()**, and **_matrix::width**.

```

00075
00076     int i, j;
00077     double* ptr = m->data;
00078     printf("%d %d\n", m->width, m->height);
00079     for (i = 0; i < m->height; i++) {
00080         for (j = 0; j < m->width; j++) {
00081             printf(" %9.6f", *(ptr++));
00082         }
00083         printf("\n");
00084     }
00085     return;
00086 }
```

Here is the call graph for this function:

**5.71.1.12 rowSwap()**

```

void rowSwap (
    matrix * a,
    int p,
    int q )
```

swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q, provided that p and q are less than or equal to the height of matrix A and p and q are different values.

Parameters

<i>the</i>	matrix to swap. This method changes the input matrix.
<i>the</i>	first row
<i>the</i>	second row

Definition at line **290** of file **matrix.c**.

References `assert()`, `_matrix::data`, `_matrix::height`, and `_matrix::width`.

```

00290             {
00291     int i;
00292     double temp;
00293     double* pRow;
00294     double* qRow;
00295
00296     assert(a->height > 2, "Matrix must have at least two rows to swap.");
00297     assert(p < a->height && q < a->height, "Values p and q must be less than the height of the matrix.");
00298
00299     // If p and q are equal, do nothing.
00300     if (p == q) {
00301         return;
00302     }
00303
00304     pRow = a->data + (p * a->width);
00305     qRow = a->data + (q * a->width);
00306
00307     // Swap!
00308     for (i = 0; i < a->width; i++) {
00309         temp = *pRow;
00310         *pRow = *qRow;
00311         *qRow = temp;
00312         pRow++;
00313         qRow++;
00314     }
00315
00316     return;
00317 }
```

Here is the call graph for this function:



5.71.1.13 scaleMatrix()

```
matrix* scaleMatrix (
    matrix * m,
    double value )
```

scales a matrix.

Parameters

<i>m</i>	the matrix to scale
<i>the</i>	value to scale by

Returns

a new matrix where each element in the input matrix is multiplied by the scalar value

Definition at line **268** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **copyMatrix()**.

```
00268     int i, elements = m->width * m->height;
00269     matrix* out = makeMatrix(m->width, m->height);
00270     double* ptrM = m->data;
00271     double* ptrOut = out->data;
00272
00273     for (i = 0; i < elements; i++) {
00274         *(ptrOut++) = *(ptrM++) * value;
00275     }
00276
00277     return out;
00278 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.71.1.14 traceMatrix()

```
double traceMatrix (
    matrix * m )
```

Given an "m rows by n columns" matrix returns the sum.

Given an "m rows by n columns" matrix.

Returns

the sum of the elements along the diagonal.

Definition at line **116** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
00116             {
00117     int i;
00118     int size;
00119     double* ptr = m->data;
00120     double sum = 0.0;
00121
00122     if (m->height < m->width) {
00123         size = m->height;
00124     }
00125     else {
00126         size = m->width;
00127     }
00128
00129     for (i = 0; i < size; i++) {
00130         sum += *ptr;
00131         ptr += m->width + 1;
00132     }
00133
00134     return sum;
00135 }
```

5.71.1.15 transposeMatrix()

```
matrix* transposeMatrix (
    matrix * m )
```

returns the transpose matrix.

Parameters

<i>the</i>	matrix to transpose.
------------	----------------------

Returns

the transposed matrix.

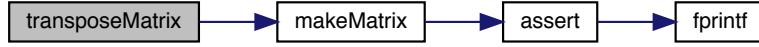
Definition at line **206** of file **matrix.c**.

References `_matrix::data`, `_matrix::height`, `makeMatrix()`, and `_matrix::width`.

```

00206             {
00207     matrix* out = makeMatrix(m->height, m->width);
00208     double* ptrM = m->data;
00209     int i, j;
00210
00211     for (i = 0; i < m->height; i++) {
00212         double* ptrOut;
00213         ptrOut = &out->data[i];
00214         for (j = 0; j < m->width; j++) {
00215             *ptrOut = *ptrM;
00216             ptrM++;
00217             ptrOut += out->width;
00218         }
00219     }
00220
00221     return out;
00222 }
```

Here is the call graph for this function:



5.72 matrix.c

```

00001 #include "matrix.h"
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <string.h>
00005
00014 void assert(int assertion, const char* message) {
00015     if (assertion == 0) {
00016         fprintf(stderr, "%s\n", message);
00017         exit(1);
00018     }
00019 }
00020
00027 matrix* makeMatrix(int width, int height) {
00028     matrix* out;
00029     assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
00030     out = (matrix*) malloc(sizeof(matrix));
00031
00032     assert(out != NULL, "Out of memory.");
00033
00034     out->width = width;
00035     out->height = height;
00036     out->data = (double*) malloc(sizeof(double) * width * height);
00037
00038     assert(out->data != NULL, "Out of memory.");
00039
00040     memset(out->data, 0.0, width * height * sizeof(double));
00041
00042     return out;
00043 }
00044
00052 matrix* copyMatrix(matrix* m) {
00053     return scaleMatrix(m, 1);
00054 }
00055
00060 void freeMatrix(matrix* m) {
00061     if (m != NULL) {
```

```
00062     if (m->data != NULL) {
00063         free(m->data);
00064         m->data = NULL;
00065     }
00066     free(m);
00067 }
00068 return;
00069 }
00070
00075 void printMatrix(matrix* m) {
00076     int i, j;
00077     double* ptr = m->data;
00078     printf("%d %d\n", m->width, m->height);
00079     for (i = 0; i < m->height; i++) {
00080         for (j = 0; j < m->width; j++) {
00081             printf(" %.9.6f", *(ptr++));
00082         }
00083         printf("\n");
00084     }
00085     return;
00086 }
00087
00094 matrix* identityMatrix(int n) {
00095     int i;
00096     matrix *out;
00097     double* ptr;
00098
00099     assert(n > 0, "Identity matrix must have value greater than zero.");
00100
00101     out = makeMatrix(n, n);
00102     ptr = out->data;
00103     for (i = 0; i < n; i++) {
00104         *ptr = 1.0;
00105         ptr += n + 1;
00106     }
00107
00108     return out;
00109 }
00110
00116 double traceMatrix(matrix* m) {
00117     int i;
00118     int size;
00119     double* ptr = m->data;
00120     double sum = 0.0;
00121
00122     if (m->height < m->width) {
00123         size = m->height;
00124     }
00125     else {
00126         size = m->width;
00127     }
00128
00129     for (i = 0; i < size; i++) {
00130         sum += *ptr;
00131         ptr += m->width + 1;
00132     }
00133
00134     return sum;
00135 }
00136
00142 matrix* meanMatrix(matrix* m) {
00143     int i, j;
00144     matrix* out;
00145
00146     assert(m->height > 0, "Height of matrix cannot be zero.");
00147
00148     out = makeMatrix(m->width, 1);
00149
00150     for (i = 0; i < m->width; i++) {
00151         double* ptr;
00152         out->data[i] = 0.0;
00153         ptr = &m->data[i];
00154         for (j = 0; j < m->height; j++) {
00155             out->data[i] += *ptr;
00156             ptr += out->width;
00157         }
00158         out->data[i] /= (double) m->height;
00159     }
00160     return out;
00161 }
00162 }
```

```

00168 matrix* covarianceMatrix(matrix* m) {
00169     int i, j, k = 0;
00170     matrix* out;
00171     matrix* mean;
00172     double* ptrA;
00173     double* ptrB;
00174     double* ptrOut;
00175
00176     assert(m->height > 1, "Height of matrix cannot be zero or one.");
00177
00178     mean = meanMatrix(m);
00179     out = makeMatrix(m->width, m->width);
00180     ptrOut = out->data;
00181
00182     for (i = 0; i < m->width; i++) {
00183         for (j = 0; j < m->width; j++) {
00184             ptrA = &m->data[i];
00185             ptrB = &m->data[j];
00186             *ptrOut = 0.0;
00187             for (k = 0; k < m->height; k++) {
00188                 *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
00189                 ptrA += m->width;
00190                 ptrB += m->width;
00191             }
00192             *ptrOut /= m->height - 1;
00193             ptrOut++;
00194         }
00195     }
00196
00197     freeMatrix(mean);
00198     return out;
00199 }
00200
00206 matrix* transposeMatrix(matrix* m) {
00207     matrix* out = makeMatrix(m->height, m->width);
00208     double* ptrM = m->data;
00209     int i, j;
00210
00211     for (i = 0; i < m->height; i++) {
00212         double* ptrOut;
00213         ptrOut = &out->data[i];
00214         for (j = 0; j < m->width; j++) {
00215             *ptrOut = *ptrM;
00216             ptrM++;
00217             ptrOut += out->width;
00218         }
00219     }
00220
00221     return out;
00222 }
00223
00230 matrix* multiplyMatrix(matrix* a, matrix* b) {
00231     int i, j, k;
00232     matrix* out;
00233     double* ptrOut;
00234     double* ptrA;
00235     double* ptrB;
00236
00237     assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");
00238
00239     out = makeMatrix(b->width, a->height);
00240     ptrOut = out->data;
00241
00242     for (i = 0; i < a->height; i++) {
00243
00244         for (j = 0; j < b->width; j++) {
00245             ptrA = &a->data[ i * a->width ];
00246             ptrB = &b->data[ j ];
00247
00248             *ptrOut = 0;
00249             for (k = 0; k < a->width; k++) {
00250                 *ptrOut += *ptrA * *ptrB;
00251                 ptrA++;
00252                 ptrB += b->width;
00253             }
00254             ptrOut++;
00255         }
00256     }
00257
00258     return out;
00259 }
```

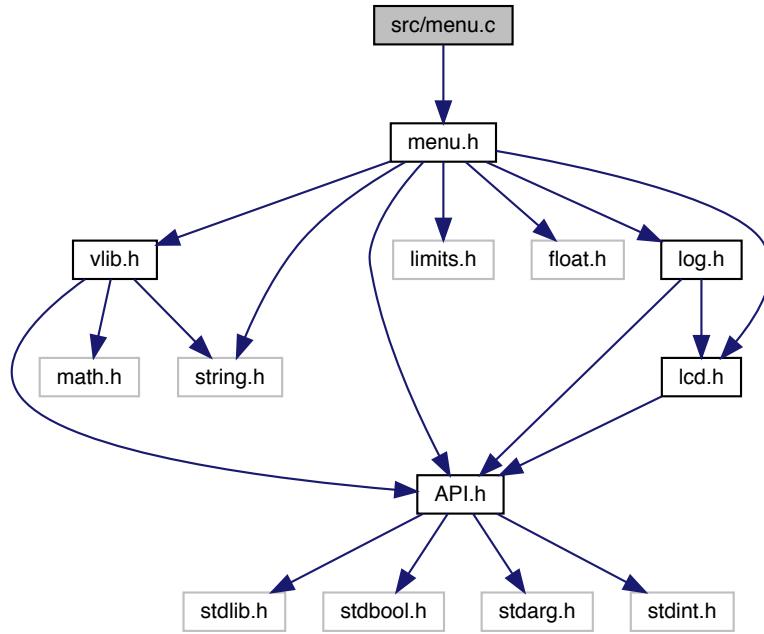
```
00260
00268 matrix* scaleMatrix(matrix* m, double value) {
00269     int i, elements = m->width * m->height;
00270     matrix* out = makeMatrix(m->width, m->height);
00271     double* ptrM = m->data;
00272     double* ptrOut = out->data;
00273
00274     for (i = 0; i < elements; i++) {
00275         *(ptrOut++) = *(ptrM++) * value;
00276     }
00277
00278     return out;
00279 }
00280
00290 void rowSwap(matrix* a, int p, int q) {
00291     int i;
00292     double temp;
00293     double* pRow;
00294     double* qRow;
00295
00296     assert(a->height > 2, "Matrix must have at least two rows to swap.");
00297     assert(p < a->height && q < a->height, "Values p and q must be less than the height of the matrix.");
00298
00299 // If p and q are equal, do nothing.
00300 if (p == q) {
00301     return;
00302 }
00303
00304 pRow = a->data + (p * a->width);
00305 qRow = a->data + (q * a->width);
00306
00307 // Swap!
00308 for (i = 0; i < a->width; i++) {
00309     temp = *pRow;
00310     *pRow = *qRow;
00311     *qRow = temp;
00312     pRow++;
00313     qRow++;
00314 }
00315
00316 return;
00317 }
00318
00333 matrix* dotProductMatrix(matrix* a, matrix* b) {
00334     matrix* out;
00335     double* ptrOut;
00336     double* ptrA;
00337     double* ptrB;
00338     int i, j, k;
00339
00340     if (b != NULL) {
00341         assert(a->width == b->width, "Matrices must be of the same dimensionality.");
00342     }
00343
00344 // Are we computing the sum of squares of the same matrix?
00345 if (a == b || b == NULL) {
00346     b = a; // May not appear safe, but we can do this without risk of losing b.
00347 }
00348
00349 out = makeMatrix(b->height, a->height);
00350 ptrOut = out->data;
00351
00352 for (i = 0; i < a->height; i++) {
00353     ptrB = b->data;
00354
00355     for (j = 0; j < b->height; j++) {
00356         ptrA = &a->data[ i * a->width ];
00357
00358         *ptrOut = 0;
00359         for (k = 0; k < a->width; k++) {
00360             *ptrOut += *ptrA * *ptrB;
00361             ptrA++;
00362             ptrB++;
00363         }
00364         ptrOut++;
00365     }
00366 }
00367
00368 return out;
00369 }
00370
```

```
00385 matrix* dotDiagonalMatrix(matrix* a, matrix* b) {
00386     matrix* out;
00387     double* ptrOut;
00388     double* ptrA;
00389     double* ptrB;
00390     int i, j;
00391
00392     if (b != NULL) {
00393         assert(a->width == b->width && a->height == b->height, "Matrices must be of the same
00394         dimensionality.");
00395     }
00396
00397     // Are we computing the sum of squares of the same matrix?
00398     if (a == b || b == NULL) {
00399         b = a; // May not appear safe, but we can do this without risk of losing b.
00400     }
00401
00402     out = makeMatrix(1, a->height);
00403     ptrOut = out->data;
00404     ptrA = a->data;
00405     ptrB = b->data;
00406
00407     for (i = 0; i < a->height; i++) {
00408         *ptrOut = 0;
00409         for (j = 0; j < a->width; j++) {
00410             *ptrOut += *ptrA * *ptrB;
00411             ptrA++;
00412             ptrB++;
00413         }
00414         ptrOut++;
00415     }
00416
00417     return out;
00418 }
```

5.73 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, const 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, const 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, const char *prompt, char *options,...)

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

5.73.1 Function Documentation

5.73.1.1 calculate_current_display()

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

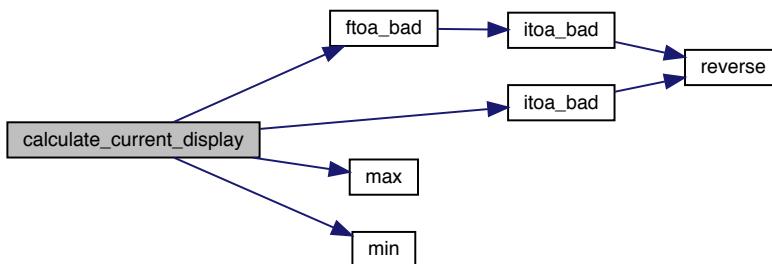
Definition at line 56 of file `menu.c`.

References `menu_t::current`, `FLOAT_TYPE`, `ftoa_bad()`, `INT_TYPE`, `itoa_bad()`, `menu_t::length`, `max()`, `menu_t::max`, `menu_t::max_f`, `min()`, `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()`.

```
00056
00057     if(menu->type == STRING_TYPE) {
00058         //Ignore warning
00059         rtn = (menu->options[menu->current % (menu->length)]);
00060     }
00061     if(menu->type == INT_TYPE) {
00062         int step = (menu->step);
00063         int min = (menu->min);
00064         int max = (menu->max);
00065         int value = menu->current * step;
00066         value = value < min ? min : value;
00067         value = value > max ? max : value;
00068         itoa_bad(value, rtn, 4);
00069     }
00070     if(menu->type == FLOAT_TYPE) {
00071         float step = (menu->step_f);
00072         float min = (menu->min_f);
00073         float max = (menu->max_f);
00074         float value = menu->current * step;
00075         value = value < min ? min : value;
00076         value = value > max ? max : value;
00077         ftoa_bad(value, rtn, 5);
00078     }
00079 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.73.1.2 create_menu()

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

Definition at line 6 of file **menu.c**.

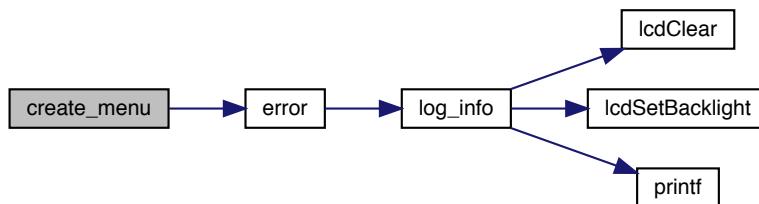
References **error()**, **menu_t::max**, **menu_t::max_f**, **menu_t::min**, **menu_t::min_f**, **menu_t::prompt**, **menu_t::step**, **menu_t::step_f**, and **menu_t::type**.

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

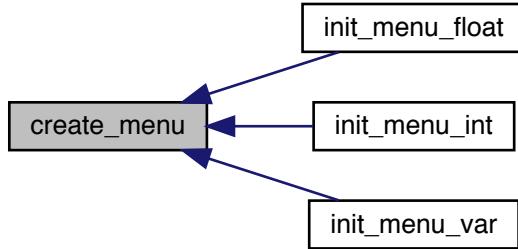
```

00006
00007     menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
00008     if (!menu) {
00009         error("Menu Malloc");
00010     }
00011     menu->type = type;
00012     // Add one for null terminator
00013     size_t strlength = strlen(prompt) + 1;
00014     menu->prompt = (char*) malloc(strlength * sizeof(char));
00015     memcpy(menu->prompt, prompt, strlength);
00016     menu->max = INT_MAX;
00017     menu->min = INT_MIN;
00018     menu->step = 1;
00019     menu->min_f = FLT_MIN;
00020     menu->max_f = FLT_MAX;
00021     menu->step_f = 1;
00022
00023     return menu;
00024 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.73.1.3 `denint_menu()`

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

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

Parameters

<code>menu</code>	the menu to free
-------------------	------------------

See also

`menu`

Author

Chris Jerrett

Date

9/8/17

Definition at line **101** of file `menu.c`.

References `menu_t::options`, and `menu_t::prompt`.

```

00101
00102     free(menu->prompt);
00103     if(menu->options != NULL) free(menu->options);
00104     free(menu);
00105 }
```

5.73.1.4 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](#) (p. 198)

Author

Chris Jerrett

Date

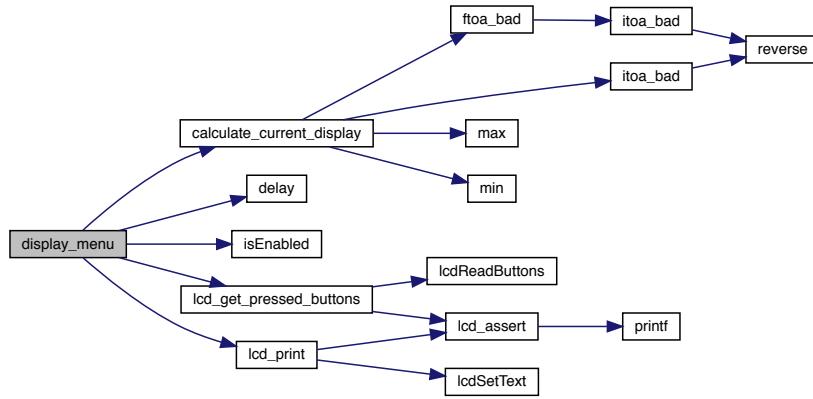
9/8/17

Definition at line **83** of file **menu.c**.

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

```
00083     {
00084     lcd_print(TOP_ROW, menu->prompt);
00085     //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
00086     while(lcd_get_pressed_buttons().middle == RELEASED && !isEnabled()) {
00087         char val[16];
00088         calculate_current_display(val, menu);
00089         if(lcd_get_pressed_buttons().right == PRESSED) {
00090             menu->current += 1;
00091         }
00092         if(lcd_get_pressed_buttons().left == PRESSED) {
00093             menu->current -= 1;
00094         }
00095         delay(500);
00096     }
00097 }
00098 return menu->current;
00099 }
```

Here is the call graph for this function:



5.73.1.5 init_menu_float()

```
menu_t* init_menu_float (
    enum menu_type type,
    float min,
    float max,
    float step,
    const 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 (p. 198)

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

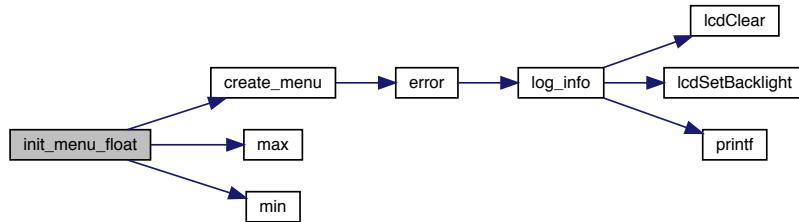
9/8/17

Definition at line **48** of file **menu.c**.

References **create_menu()**, **max()**, **menu_t::max_f**, **min()**, **menu_t::min_f**, and **menu_t::step_f**.

```
00048     menu_t* menu = create_menu(type, prompt);
00049     menu->min_f = min;
00050     menu->max_f = max;
00051     menu->step_f = step;
00052     return menu;
00053 }
00054 }
```

Here is the call graph for this function:

**5.73.1.6 init_menu_int()**

```
menu_t* init_menu_int (
    enum menu_type type,
    int min,
    int max,
    int step,
    const 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 \(p. 198\)](#)

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

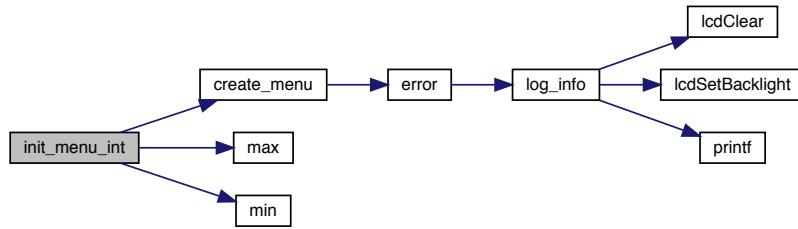
9/8/17

Definition at line 40 of file **menu.c**.References [create_menu\(\)](#), [max\(\)](#), [menu_t::max](#), [min\(\)](#), [menu_t::min](#), and [menu_t::step](#).

```

00040
00041     menu_t* menu = create_menu(type, prompt);
00042     menu->min = min;
00043     menu->max = max;
00044     menu->step = step;
00045     return menu;
00046 }
```

Here is the call graph for this function:

5.73.1.7 **init_menu_var()**

```

menu_t* init_menu_var (
    enum menu_type type,
    unsigned int nums,
    const 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](#) (p. 198)

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

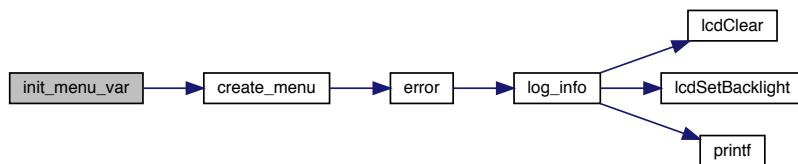
Definition at line **26** of file **menu.c**.

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

```

00026
00027     menu_t* menu = create_menu(type, prompt);
00028     va_list values;
00029     char **options_array = (char**)calloc(sizeof(char*), nums);
00030     va_start(values, options);
00031     for(unsigned int i = 0; i < nums; i++){
00032         options_array[i] = va_arg(values, char*);
00033     }
00034     va_end(values);
00035     menu->options = options_array;
00036     menu->length = nums;
00037     return menu;
00038 }
```

Here is the call graph for this function:



5.74 menu.c

```

00001 #include "menu.h"
00002
00003 static menu_t* create_menu(enum menu_type type, const char *prompt);
00004 static void calculate_current_display(char* rtn, menu_t *menu);
00005
00006 static menu_t* create_menu(enum menu_type type, const char *prompt) {
00007     menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
00008     if (!menu) {
00009         error("Menu Malloc");
00010     }
00011     menu->type = type;
00012     // Add one for null terminator
00013     size_t strlength = strlen(prompt) + 1;
00014     menu->prompt = (char*) malloc(strlength * sizeof(char));
00015     memcpy(menu->prompt, prompt, strlength);
00016     menu->max = INT_MAX;
00017     menu->min = INT_MIN;
00018     menu->step = 1;
00019     menu->min_f = FLT_MIN;
00020     menu->max_f = FLT_MAX;
00021     menu->step_f = 1;
00022
00023     return menu;
00024 }
00025
00026 menu_t* init_menu_var(enum menu_type type, unsigned int nums, const char *prompt, char* options,...){
00027     menu_t* menu = create_menu(type, prompt);
00028     va_list values;
00029     char **options_array = (char**)calloc(sizeof(char*), nums);
00030     va_start(values, options);
00031     for(unsigned int i = 0; i < nums; i++){
00032         options_array[i] = va_arg(values, char*);
00033     }
00034     va_end(values);
00035     menu->options = options_array;
00036     menu->length = nums;
00037     return menu;
00038 }
00039
00040 menu_t* init_menu_int(enum menu_type type, int min, int max, int step, const char* prompt){
00041     menu_t* menu = create_menu(type, prompt);
00042     menu->min = min;
00043     menu->max = max;
00044     menu->step = step;
00045     return menu;
00046 }
00047
00048 menu_t* init_menu_float(enum menu_type type, float min, float max, float step, const char* prompt){
00049     menu_t* menu = create_menu(type, prompt);
00050     menu->min_f = min;
00051     menu->max_f = max;
00052     menu->step_f = step;
00053     return menu;
00054 }
00055
00056 static void calculate_current_display(char* rtn, menu_t *menu) {
00057     if(menu->type == STRING_TYPE){
00058         //Ignore warning
00059         rtn = (menu->options[menu->current % (menu->length)]);
00060     }
00061     if(menu->type == INT_TYPE) {
00062         int step = (menu->step);
00063         int min = (menu->min);
00064         int max = (menu->max);
00065         int value = menu->current * step;
00066         value = value < min ? min : value;
00067         value = value > max ? max : value;
00068         itoa_bad(value, rtn, 4);
00069     }
00070     if(menu->type == FLOAT_TYPE) {
00071         float step = (menu->step_f);
00072         float min = (menu->min_f);
00073         float max = (menu->max_f);
00074         float value = menu->current * step;
00075         value = value < min ? min : value;
00076         value = value > max ? max : value;
00077         ftoa_bad(value, rtn, 5);
00078     }

```

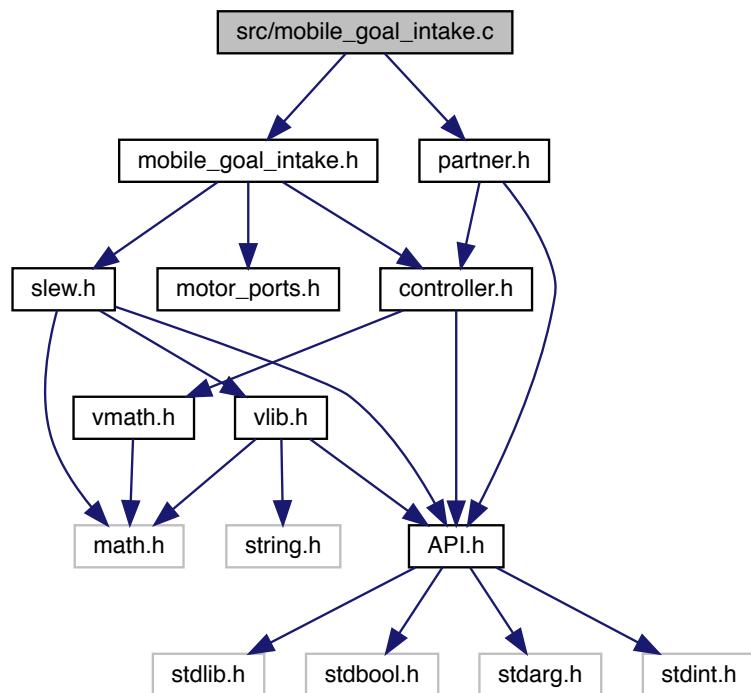
```

00079     }
00080 }
00081
00082
00083 int display_menu(menu_t *menu){
00084     lcd_print(TOP_ROW, menu->prompt);
00085     //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
00086     while(lcd_get_pressed_buttons().middle == RELEASED && !isEnabled()) {
00087         char val[16];
00088         calculate_current_display(val, menu);
00089
00090         if(lcd_get_pressed_buttons().right == PRESSED) {
00091             menu->current += 1;
00092         }
00093         if(lcd_get_pressed_buttons().left == PRESSED) {
00094             menu->current -= 1;
00095         }
00096         delay(500);
00097     }
00098     return menu->current;
00099 }
00100
00101 void denint_menu(menu_t *menu) {
00102     free(menu->prompt);
00103     if(menu->options != NULL) free(menu->options);
00104     free(menu);
00105 }

```

5.75 src/mobile_goal_intake.c File Reference

```
#include "mobile_goal_intake.h"
#include "partner.h"
Include dependency graph for mobile_goal_intake.c:
```



Functions

- static void **lower_intake ()**
- static void **raise_intake ()**
- static void **set_intake_motor** (int n)
- void **updateIntake ()**

updates the mobile goal intake in teleop.

5.75.1 Function Documentation

5.75.1.1 **lower_intake()**

```
static void lower_intake ( ) [static]
```

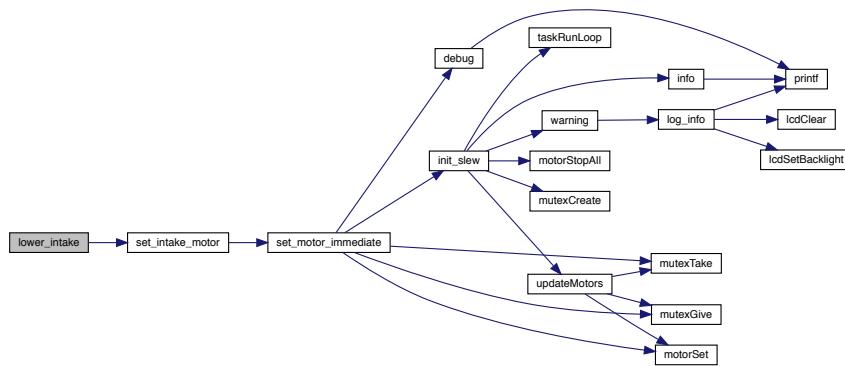
Definition at line 8 of file **mobile_goal_intake.c**.

References **set_intake_motor()**.

Referenced by **updateIntake()**.

```
00008
00009     set_intake_motor(100);
00010 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.75.1.2 raise_intake()

```
static void raise_intake ( ) [static]
```

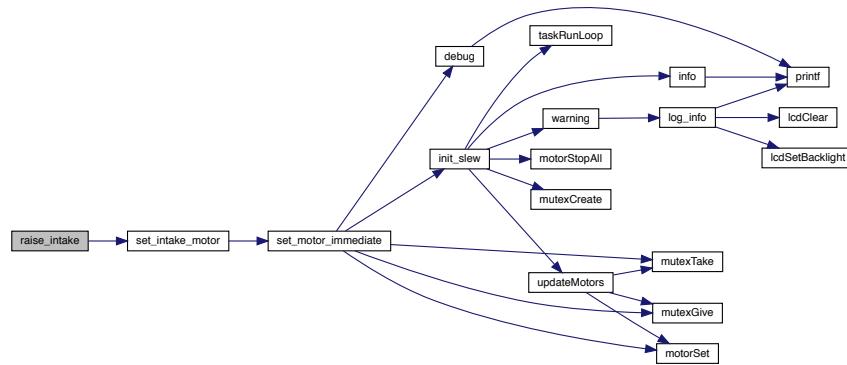
Definition at line 12 of file **mobile_goal_intake.c**.

References **set_intake_motor()**.

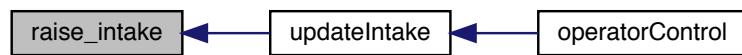
Referenced by **updateIntake()**.

```
00012     {  
00013     set_intake_motor(-100);  
00014 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.75.1.3 set_intake_motor()

```
static void set_intake_motor (
    int n ) [static]
```

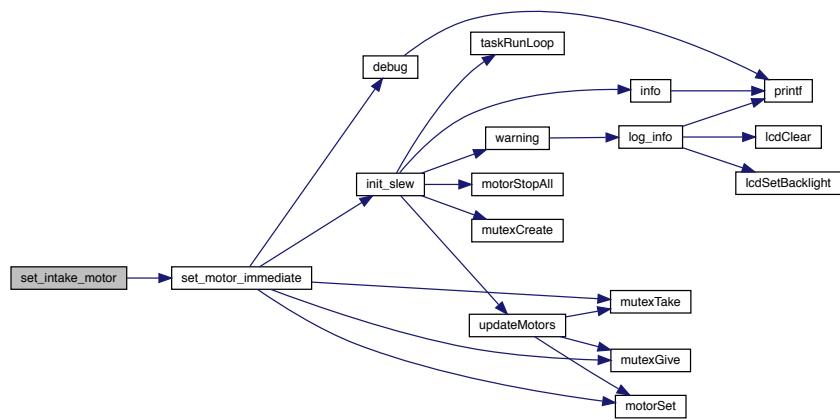
Definition at line 4 of file **mobile_goal_intake.c**.

References **INTAKE_MOTOR**, and **set_motor_immediate()**.

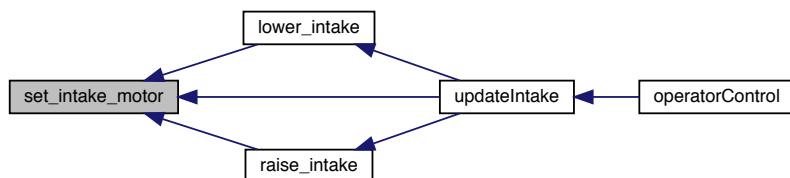
Referenced by **lower_intake()**, **raise_intake()**, and **updateIntake()**.

```
00004 {
00005     set_motor_immediate(INTAKE_MOTOR, n);
00006 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.75.1.4 updateIntake()

```
void updateIntake ( )
```

updates the mobile goal intake in teleop.

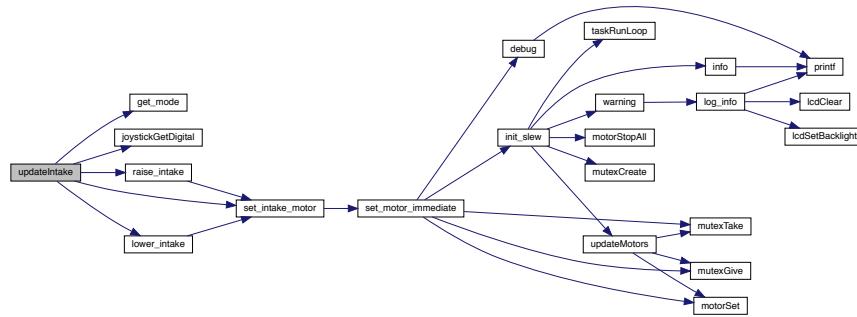
Definition at line 16 of file **mobile_goal_intake.c**.

References **get_mode()**, **JOY_DOWN**, **JOY_UP**, **joystickGetDigital()**, **lower_intake()**, **MAIN_CONTROLLER_MODE**, **MASTER**, **PARTNER**, **PARTNER_CONTROLLER_MODE**, **raise_intake()**, and **set_intake_motor()**.

Referenced by **operatorControl()**.

```
00016
00017     if(joystickGetDigital(MASTER, 7, JOY_UP) && (get_mode() ==
00018         MAIN_CONTROLLER_MODE)
00019     || joystickGetDigital(PARTNER, 6, JOY_UP) && get_mode() ==
00020         PARTNER_CONTROLLER_MODE) {
00021         raise_intake();
00022     }
00023     else if(joystickGetDigital(MASTER, 7, JOY_DOWN) && (get_mode() ==
00024         MAIN_CONTROLLER_MODE)
00025     || joystickGetDigital(PARTNER, 6, JOY_DOWN) && get_mode() ==
00026         PARTNER_CONTROLLER_MODE){
00027         lower_intake();
00028     }
00029     else set_intake_motor(0);
00030 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.76 mobile_goal_intake.c

```

00001 #include "mobile_goal_intake.h"
00002 #include "partner.h"
00003
00004 static void set_intake_motor(int n) {
00005     set_motor_immediate(INTAKE_MOTOR, n);
00006 }
00007
00008 static void lower_intake() {
00009     set_intake_motor(100);
00010 }
00011
00012 static void raise_intake() {
00013     set_intake_motor(-100);
00014 }
00015
00016 void updateIntake() {
00017     if(joystickGetDigital(MASTER, 7, JOY_UP) && (get_mode() ==
00018         MAIN_CONTROLLER_MODE)
00019     || joystickGetDigital(PARTNER, 6, JOY_UP) && get_mode() ==
00020         PARTNER_CONTROLLER_MODE) {
00021         raise_intake();
00022     } else if(joystickGetDigital(MASTER, 7, JOY_DOWN) && (get_mode() ==
00023         MAIN_CONTROLLER_MODE)
00024     || joystickGetDigital(PARTNER, 6, JOY_DOWN) && get_mode() ==
00025         PARTNER_CONTROLLER_MODE) {
00026         lower_intake();
00027     }
00028     else set_intake_motor(0);
00029 }
00030

```

5.77 src/opcontrol.c File Reference

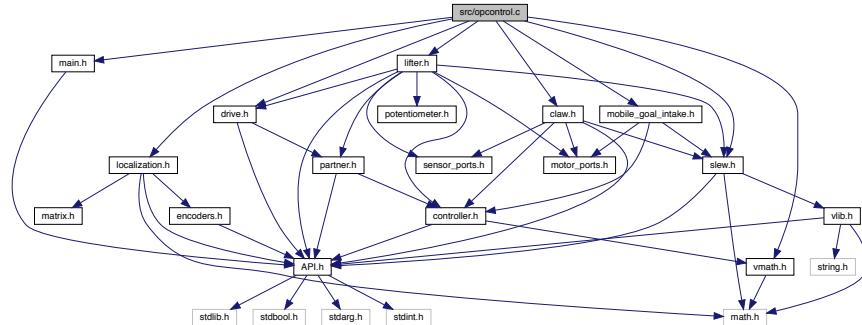
File for operator control code.

```

#include "main.h"
#include "slew.h"
#include "drive.h"
#include "lifter.h"
#include "localization.h"
#include "claw.h"
#include "mobile_goal_intake.h"
#include "vmath.h"

```

Include dependency graph for opcontrol.c:



Functions

- void **operatorControl()**

5.77.1 Detailed Description

File for operator control code.

This file should contain the user **operatorControl()** (p. 361) 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.

Definition in file **opcontrol.c**.

5.77.2 Function Documentation

5.77.2.1 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()** (p. 39) or **taskDelayUntil()** (p. 82) 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 40 of file **opcontrol.c**.

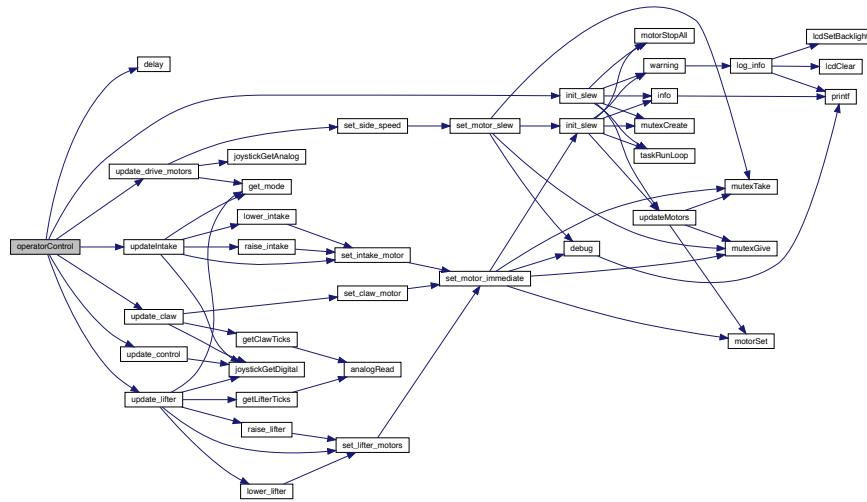
References **delay()**, **init_slew()**, **update_claw()**, **update_control()**, **update_drive_motors()**, **update_lifter()**, and **updateIntake()**.

```

00040
00041     init_slew();
00042     delay(10);
00043     while (1) {
00044         update_drive_motors();
00045         update_lifter();
00046         update_claw();
00047         updateIntake();
00048         update_control();
00049         delay(25);
00050     }
00051 }

```

Here is the call graph for this function:



5.78 opcontrol.c

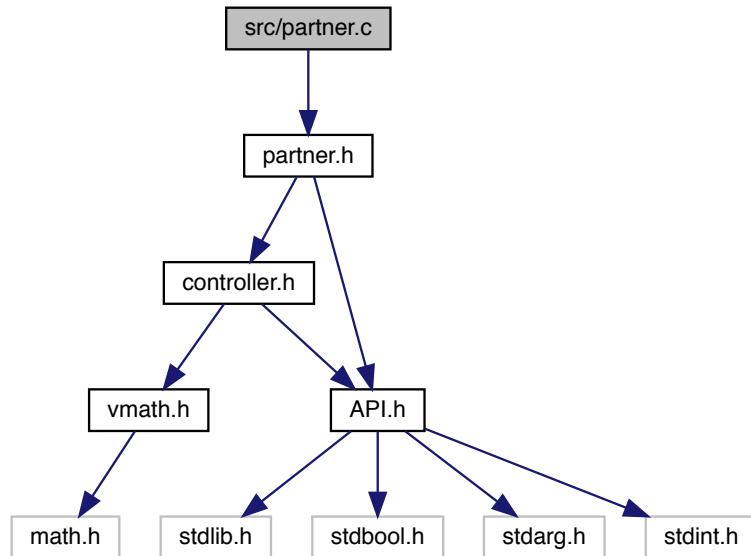
```

00001
00013 #include "main.h"
00014 #include "slew.h"
00015 #include "drive.h"
00016 #include "lifter.h"
00017 #include "localization.h"
00018 #include "claw.h"
00019 #include "mobile_goal_intake.h"
00020 #include "vmath.h"
00021 #include "lifter.h"
00022
00040 void operatorControl() {
00041     init_slew();
00042     delay(10);
00043     while (1) {
00044         update_drive_motors();
00045         update_lifter();
00046         update_claw();
00047         updateIntake();
00048         update_control();
00049         delay(25);
00050     }
00051 }

```

5.79 src/partner.c File Reference

```
#include "partner.h"
Include dependency graph for partner.c:
```



Functions

- enum **CONTROL_MODE** `get_mode ()`
- void `update_control ()`

Variables

- static enum **CONTROL_MODE** `mode = MAIN_CONTROLLER_MODE`

5.79.1 Function Documentation

5.79.1.1 get_mode()

```
enum CONTROLL_MODE get_mode ( )
```

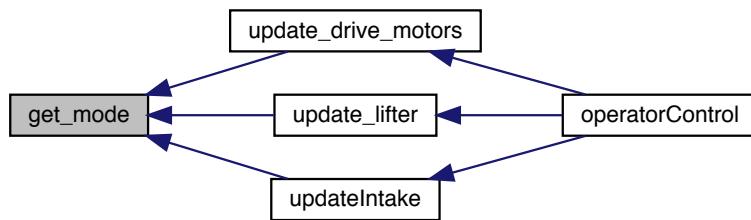
Definition at line **5** of file **partner.c**.

References **mode**.

Referenced by **update_drive_motors()**, **update_lifter()**, and **updateIntake()**.

```
00005     {
00006     return mode;
00007 }
```

Here is the caller graph for this function:



5.79.1.2 update_control()

```
void update_control ( )
```

Definition at line **9** of file **partner.c**.

References **JOY_LEFT**, **JOY_RIGHT**, **joystickGetDigital()**, **MAIN_CONTROLLER_MODE**, **mode**, **PARTNER**, and **PARTNER_CONTROLLER_MODE**.

Referenced by **operatorControl()**.

```
00009     {
00010     if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
00011         mode = MAIN_CONTROLLER_MODE;
00012     } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
00013         mode = PARTNER_CONTROLLER_MODE;
00014     }
00015 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.79.2 Variable Documentation

5.79.2.1 mode

```
enum CONTROL_MODE mode = MAIN_CONTROLLER_MODE [static]
```

Definition at line 3 of file **partner.c**.

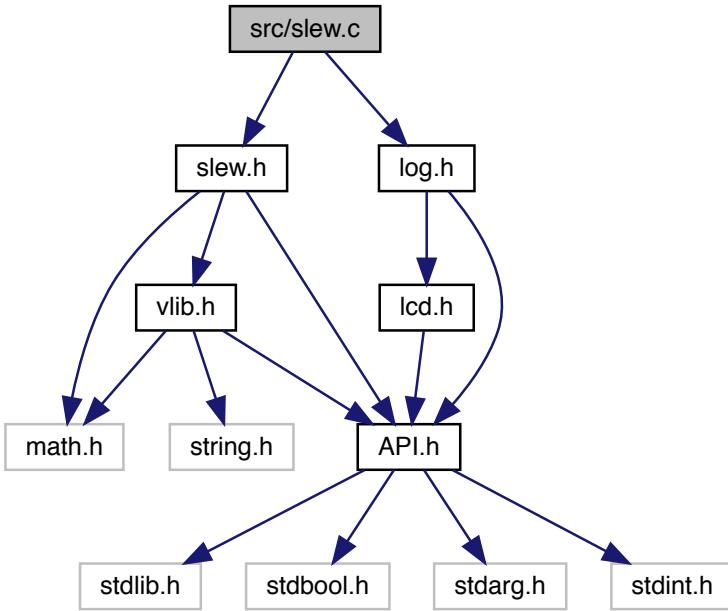
Referenced by **get_mode()**, and **update_control()**.

5.80 partner.c

```
00001 #include "partner.h"
00002
00003 static enum CONTROL_MODE mode = MAIN_CONTROLLER_MODE;
00004
00005 enum CONTROL_MODE get_mode() {
00006     return mode;
00007 }
00008
00009 void update_control() {
00010     if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
00011         mode = MAIN_CONTROLLER_MODE;
00012     } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
00013         mode = PARTNER_CONTROLLER_MODE;
00014     }
00015 }
```

5.81 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_immediate** (int motor, int speed)
- 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
- static int **motors_curr_speeds** [10]
- static int **motors_set_speeds** [10]
- static **TaskHandle** **slew** = NULL
- static **Mutex** **speeds_mutex**

5.81.1 Function Documentation

5.81.1.1 deinitSlew()

```
void deinitSlew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

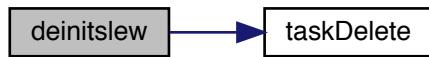
Definition at line **43** of file **slew.c**.

References **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **slew**, and **taskDelete()**.

Referenced by **autonomous()**.

```
00043     {
00044     taskDelete(slew);
00045     memset(motors_set_speeds, 0, sizeof(int) * 10);
00046     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00047     initialized = false;
00048 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.81.1.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

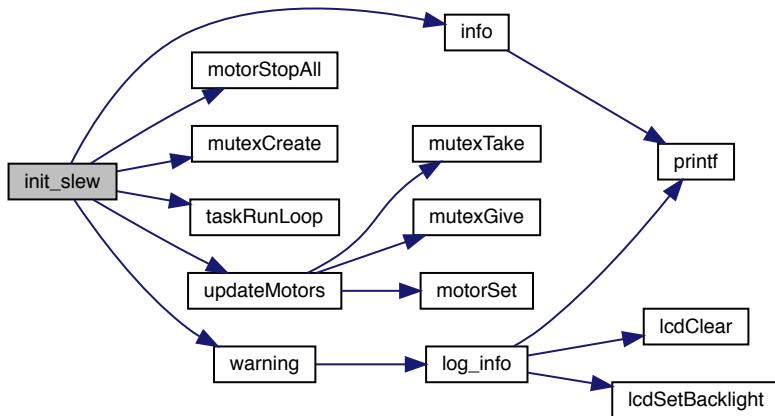
Definition at line **30** of file **slew.c**.

References **info()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **motorStopAll()**, **mutexCreate()**, **slew**, **speeds_mutex**, **taskRunLoop()**, **updateMotors()**, and **warning()**.

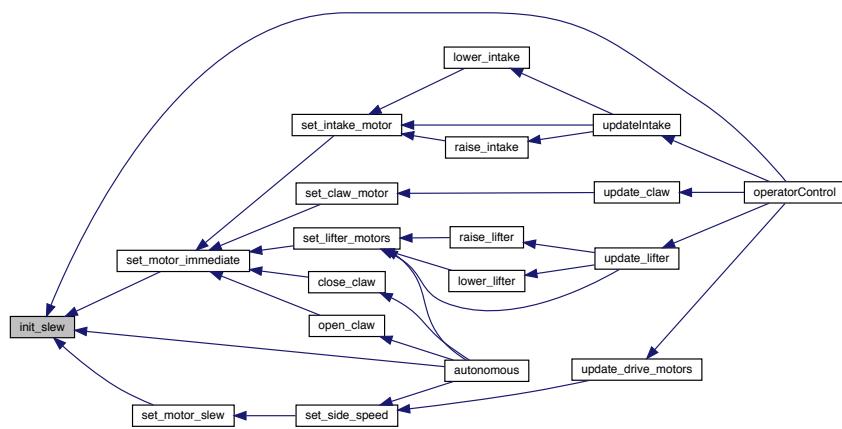
Referenced by **autonomous()**, **operatorControl()**, **set_motor_immediate()**, and **set_motor_slew()**.

```
00030             {
00031     if(initialized) {
00032         warning("Trying to init already init slew");
00033     }
00034     memset(motors_set_speeds, 0, sizeof(int) * 10);
00035     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00036     motorStopAll();
00037     info("Did Init Slew");
00038     speeds_mutex = mutexCreate();
00039     slew = taskRunLoop(updateMotors, 100);
00040     initialized = true;
00041 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.81.1.3 set_motor_immediate()

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

Definition at line **60** of file **slew.c**.

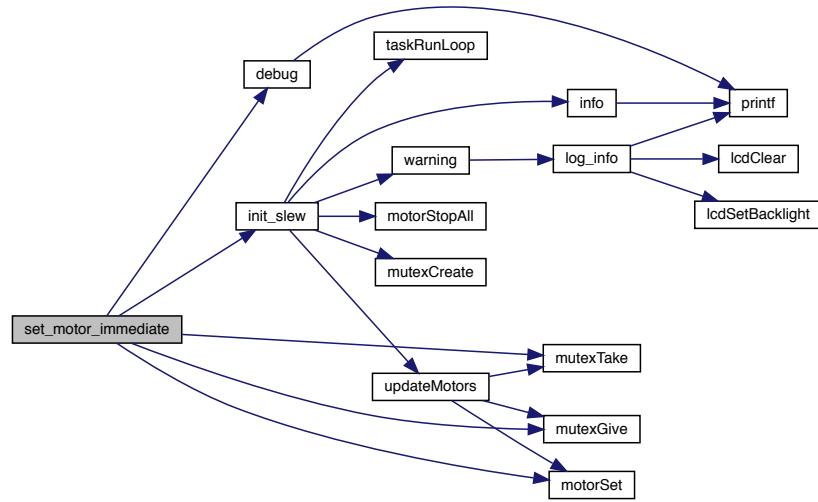
References **debug()**, **init_slew()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **motorSet()**, **mutexGive()**, **mutexTake()**, and **speeds_mutex**.

Referenced by **close_claw()**, **open_claw()**, **set_claw_motor()**, **set_intake_motor()**, and **set_lifter_motors()**.

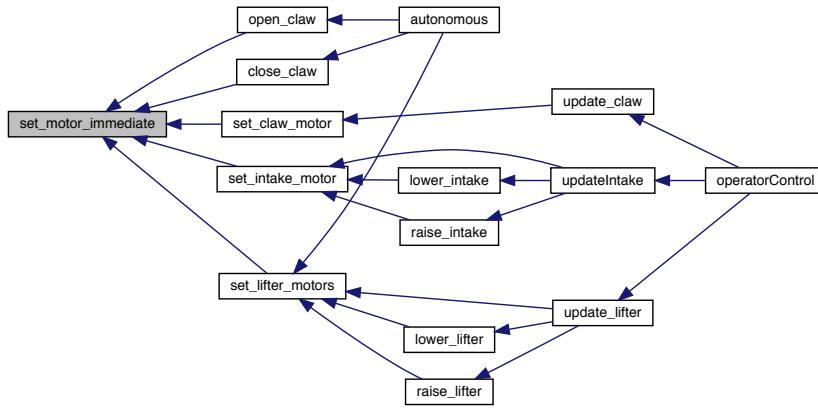
```

00060
00061     if(!initialized) {
00062         debug("Slew Not Initialized! Initializing");
00063         init_slew();
00064     }
00065     motorSet(motor, speed);
00066     mutexTake(speeds_mutex, 10);
00067     motors_curr_speeds[motor-1] = speed;
00068     motors_set_speeds[motor-1] = speed;
00069     mutexGive(speeds_mutex);
00070 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.81.1.4 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 **50** of file **slew.c**.

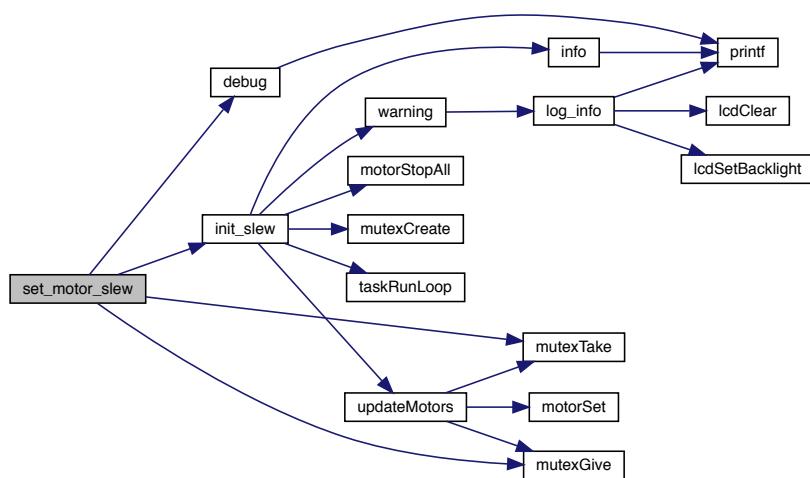
References **debug()**, **init_slew()**, **initialized**, **motors_set_speeds**, **mutexGive()**, **mutexTake()**, and **speeds_mutex**.

Referenced by **set_side_speed()**.

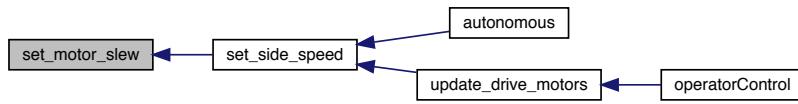
```

00050
00051     if(!initialized) {
00052         debug("Slew Not Initialized! Initializing");
00053         init_slew();
00054     }
00055     mutexTake(speeds_mutex, 10);
00056     motors_set_speeds[motor-1] = speed;
00057     mutexGive(speeds_mutex);
00058 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.81.1.5 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 13 of file **slew.c**.

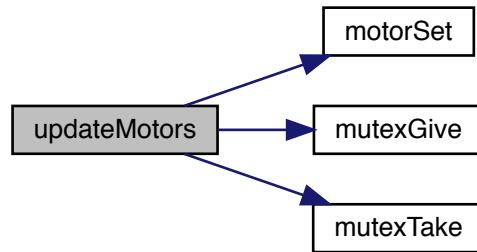
References **motors_curr_speeds**, **motors_set_speeds**, **motorSet()**, **mutexGive()**, **mutexTake()**, and **speeds_mutex**.

Referenced by **init_slew()**.

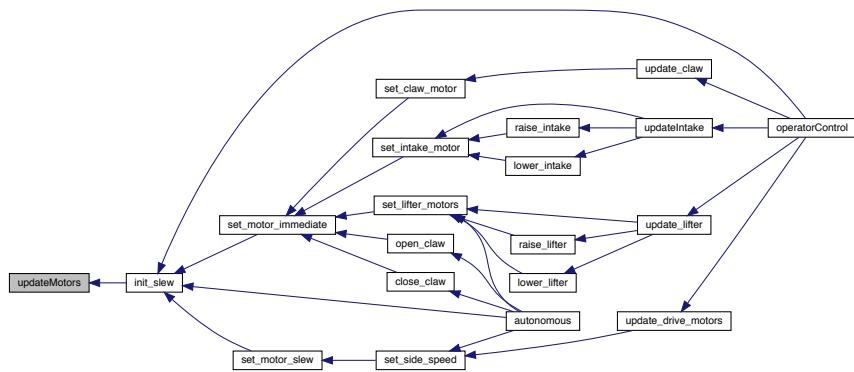
```

00013     {
00014     //Take back half approach
00015     //Not linear but equal to setSpeed(1-(1/2)^x)
00016     for(unsigned int i = 0; i < 9; i++) {
00017         if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
00018         mutexTake(speeds_mutex, 10);
00019         int set_speed = (motors_set_speeds[i]);
00020         int curr_speed = motors_curr_speeds[i];
00021         mutexGive(speeds_mutex);
00022         int diff = set_speed - curr_speed;
00023         int offset = diff;
00024         int n = curr_speed + offset;
00025         motors_curr_speeds[i] = n;
00026         motorSet(i+1, n);
00027     }
00028 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.81.2 Variable Documentation

5.81.2.1 initialized

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

Definition at line 11 of file **slew.c**.

Referenced by `deinitSlew()`, `init_slew()`, `set_motor_immediate()`, and `set_motor_slew()`.

5.81.2.2 motors_curr_speeds

```
int motors_curr_speeds[10] [static]
```

Definition at line **7** of file **slew.c**.

Referenced by **deinitSlew()**, **init_slew()**, **set_motor_immediate()**, and **updateMotors()**.

5.81.2.3 motors_set_speeds

```
int motors_set_speeds[10] [static]
```

Definition at line **6** of file **slew.c**.

Referenced by **deinitSlew()**, **init_slew()**, **set_motor_immediate()**, **set_motor_slew()**, and **updateMotors()**.

5.81.2.4 slew

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

Definition at line **9** of file **slew.c**.

Referenced by **deinitSlew()**, and **init_slew()**.

5.81.2.5 speeds_mutex

```
Mutex speeds_mutex [static]
```

Definition at line **4** of file **slew.c**.

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

5.82 slew.c

```

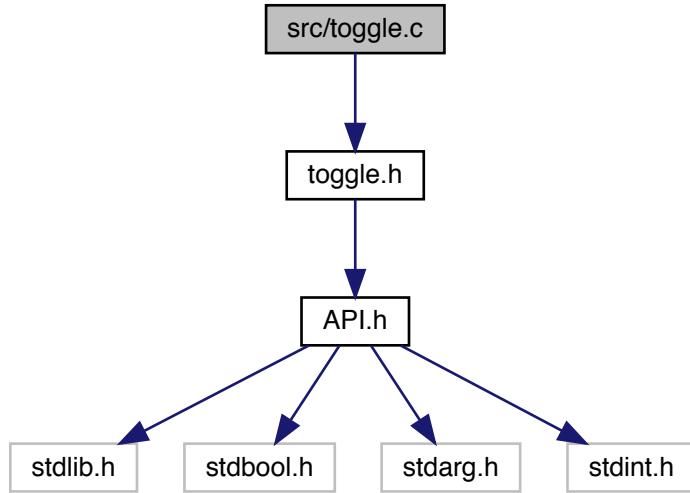
00001 #include "slew.h"
00002 #include "log.h"
00003
00004 static Mutex speeds_mutex;
00005
00006 static int motors_set_speeds[10];
00007 static int motors_curr_speeds[10];
00008
00009 static TaskHandle slew = NULL; //TaskHandle is of type void*
00010
00011 static bool initialized = false;
00012
00013 void updateMotors() {
00014     //Take back half approach
00015     //Not linear but equal to setSpeed(1-(1/2)^x)
00016     for(unsigned int i = 0; i < 9; i++) {
00017         if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
00018         mutexTake(speeds_mutex, 10);
00019         int set_speed = (motors_set_speeds[i]);
00020         int curr_speed = motors_curr_speeds[i];
00021         mutexGive(speeds_mutex);
00022         int diff = set_speed - curr_speed;
00023         int offset = diff;
00024         int n = curr_speed + offset;
00025         motors_curr_speeds[i] = n;
00026         motorSet(i+1, n);
00027     }
00028 }
00029
00030 void init_slew(){
00031     if(initialized) {
00032         warning("Trying to init already init slew");
00033     }
00034     memset(motors_set_speeds, 0, sizeof(int) * 10);
00035     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00036     motorStopAll();
00037     info("Did Init Slew");
00038     speeds_mutex = mutexCreate();
00039     slew = taskRunLoop(updateMotors, 100);
00040     initialized = true;
00041 }
00042
00043 void deinit_slew(){
00044     taskDelete(slew);
00045     memset(motors_set_speeds, 0, sizeof(int) * 10);
00046     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00047     initialized = false;
00048 }
00049
00050 void set_motor_slew(int motor, int speed){
00051     if(!initialized) {
00052         debug("Slew Not Initialized! Initializing");
00053         init_slew();
00054     }
00055     mutexTake(speeds_mutex, 10);
00056     motors_set_speeds[motor-1] = speed;
00057     mutexGive(speeds_mutex);
00058 }
00059
00060 void set_motor_immediate(int motor, int speed) {
00061     if(!initialized) {
00062         debug("Slew Not Initialized! Initializing");
00063         init_slew();
00064     }
00065     motorSet(motor, speed);
00066     mutexTake(speeds_mutex, 10);
00067     motors_curr_speeds[motor-1] = speed;
00068     motors_set_speeds[motor-1] = speed;
00069     mutexGive(speeds_mutex);
00070 }

```

5.83 src/toggle.c File Reference

```
#include "toggle.h"
```

Include dependency graph for toggle.c:



Functions

- **bool buttonGetState (button_t button)**
Returns the current status of a button (pressed or not pressed)
- **void buttonInit ()**
Initializes the buttons array.
- **bool buttonIsNewPress (button_t button)**
Detects if button is a new press from most recent check by comparing previous value to current value.

Variables

- **bool buttonPressed [27]**

5.83.1 Function Documentation

5.83.1.1 buttonGetState()

```
bool buttonGetState (
    button_t   )
```

Returns the current status of a button (pressed or not pressed)

Parameters

<i>button</i>	The button to detect from the Buttons enumeration.
---------------	--

Returns

true (pressed) or false (not pressed)

Definition at line 25 of file **toggle.c**.

References **JOY_DOWN**, **JOY_LEFT**, **JOY_RIGHT**, **JOY_UP**, **joystickGetDigital()**, **LCD_BTN_CENTER**, **LCD_BTN_LEFT**, **LCD_BTN_RIGHT**, **LCD_CENT**, **LCD_LEFT**, **LCD_RIGHT**, **IcdReadButtons()**, and **uart1**.

Referenced by **buttonIsNewPress()**.

```

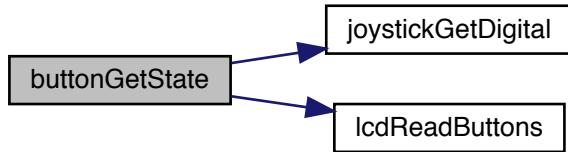
00025
00026     bool currentButton = false;
00027
00028 // Determine how to get the current button value (from what function) and where it
00029 // is, then get it.
00030 if (button < LCD_LEFT) {
00031     // button is a joystick button
00032     unsigned char joystick;
00033     unsigned char buttonGroup;
00034     unsigned char buttonLocation;
00035
00036     button_t newButton;
00037     if (button <= 11) {
00038         // button is on joystick 1
00039         joystick = 1;
00040         newButton = button;
00041     }
00042     else {
00043         // button is on joystick 2
00044         joystick = 2;
00045         // shift button down to joystick 1 buttons in order to
00046         // detect which button on joystick is queried
00047         newButton = (button_t)(button - 12);
00048     }
00049
00050     switch (newButton) {
00051     case 0:
00052         buttonGroup = 5;
00053         buttonLocation = JOY_DOWN;
00054         break;
00055     case 1:
00056         buttonGroup = 5;
00057         buttonLocation = JOY_UP;
00058         break;
00059     case 2:
00060         buttonGroup = 6;
00061         buttonLocation = JOY_DOWN;
00062         break;
00063     case 3:
00064         buttonGroup = 6;
00065         buttonLocation = JOY_UP;
00066         break;
00067     case 4:
00068         buttonGroup = 7;
00069         buttonLocation = JOY_UP;
00070         break;
00071     case 5:
00072         buttonGroup = 7;
00073         buttonLocation = JOY_LEFT;
00074         break;
00075     case 6:
00076         buttonGroup = 7;
00077         buttonLocation = JOY_RIGHT;
00078         break;
00079     case 7:

```

```

00080         buttonGroup = 7;
00081         buttonLocation = JOY_DOWN;
00082         break;
00083     case 8:
00084         buttonGroup = 8;
00085         buttonLocation = JOY_UP;
00086         break;
00087     case 9:
00088         buttonGroup = 8;
00089         buttonLocation = JOY_LEFT;
00090         break;
00091     case 10:
00092         buttonGroup = 8;
00093         buttonLocation = JOY_RIGHT;
00094         break;
00095     case 11:
00096         buttonGroup = 8;
00097         buttonLocation = JOY_DOWN;
00098         break;
00099     default:
00100         break;
00101     }
00102     currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
00103 }
00104 else {
00105     // button is on LCD
00106     if (button == LCD_LEFT)
00107         currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);
00108
00109     if (button == LCD_CENT)
00110         currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);
00111
00112     if (button == LCD_RIGHT)
00113         currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
00114 }
00115 return currentButton;
00116 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.83.1.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons array.

Initializes the buttons.

Definition at line **20** of file **toggle.c**.

References **buttonPressed**.

```
00020      {
00021      for (int i = 0; i < 27; i++)
00022          buttonPressed[i] = false;
00023 }
```

5.83.1.3 buttonIsNewPress()

```
bool buttonIsNewPress (
    button_t button )
```

Detects if button is a new press from most recent check by comparing previous value to current value.

Parameters

<i>button</i>	The button to detect from the Buttons enumeration (see include/buttons.h).
---------------	--

Returns

true or false depending on if there was a change in button state.

Example code:

```
...
if(buttonIsNewPress(JOY1_8D))
    digitalWrite(1, !digitalRead(1));
...
```

Definition at line **135** of file **toggle.c**.

References **buttonGetState()**, and **buttonPressed**.

```

00135
00136     bool currentButton = buttonGetState(button);
00137
00138     if (!currentButton) // buttons is not currently pressed
00139         buttonPressed[button] = false;
00140
00141     if (currentButton && !buttonPressed[button]) {
00142         // button is currently pressed and was not detected as being pressed during last check
00143         buttonPressed[button] = true;
00144         return true;
00145     }
00146     else return false; // button is not pressed or was already detected
00147 }
```

Here is the call graph for this function:



5.83.2 Variable Documentation

5.83.2.1 buttonPressed

```
bool buttonPressed[27]
```

Represents the array of "wasPressed" for all 27 available buttons.

Definition at line **15** of file **toggle.c**.

Referenced by **buttonInit()**, and **buttonIsNewPress()**.

5.84 toggle.c

```

00001
00010 #include "toggle.h"
00011
00015 bool buttonPressed[27];
00016
00020 void buttonInit() {
00021     for (int i = 0; i < 27; i++)
00022         buttonPressed[i] = false;
00023 }
00024
00025 bool buttonGetState(button_t button) {
00026     bool currentButton = false;
00027
00028     // Determine how to get the current button value (from what function) and where it
00029     // is, then get it.
```

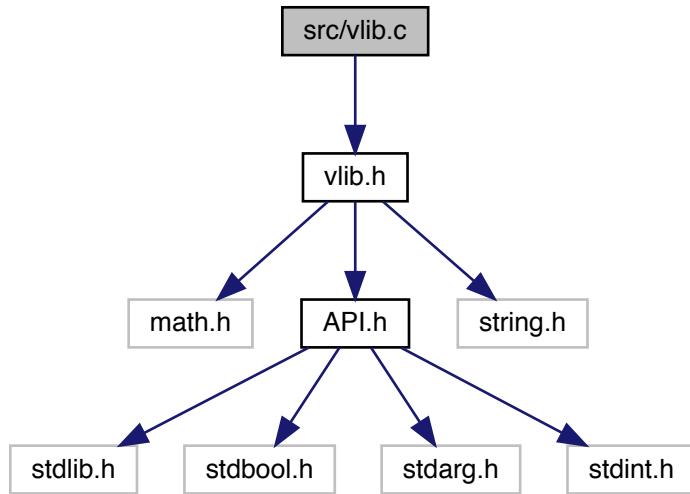
```
00030     if (button < LCD_LEFT) {
00031         // button is a joystick button
00032         unsigned char joystick;
00033         unsigned char buttonGroup;
00034         unsigned char buttonLocation;
00035
00036         button_t newButton;
00037         if (button <= 11) {
00038             // button is on joystick 1
00039             joystick = 1;
00040             newButton = button;
00041         }
00042         else {
00043             // button is on joystick 2
00044             joystick = 2;
00045             // shift button down to joystick 1 buttons in order to
00046             // detect which button on joystick is queried
00047             newButton = (button_t)(button - 12);
00048         }
00049
00050         switch (newButton) {
00051             case 0:
00052                 buttonGroup = 5;
00053                 buttonLocation = JOY_DOWN;
00054                 break;
00055             case 1:
00056                 buttonGroup = 5;
00057                 buttonLocation = JOY_UP;
00058                 break;
00059             case 2:
00060                 buttonGroup = 6;
00061                 buttonLocation = JOY_DOWN;
00062                 break;
00063             case 3:
00064                 buttonGroup = 6;
00065                 buttonLocation = JOY_UP;
00066                 break;
00067             case 4:
00068                 buttonGroup = 7;
00069                 buttonLocation = JOY_UP;
00070                 break;
00071             case 5:
00072                 buttonGroup = 7;
00073                 buttonLocation = JOY_LEFT;
00074                 break;
00075             case 6:
00076                 buttonGroup = 7;
00077                 buttonLocation = JOY_RIGHT;
00078                 break;
00079             case 7:
00080                 buttonGroup = 7;
00081                 buttonLocation = JOY_DOWN;
00082                 break;
00083             case 8:
00084                 buttonGroup = 8;
00085                 buttonLocation = JOY_UP;
00086                 break;
00087             case 9:
00088                 buttonGroup = 8;
00089                 buttonLocation = JOY_LEFT;
00090                 break;
00091             case 10:
00092                 buttonGroup = 8;
00093                 buttonLocation = JOY_RIGHT;
00094                 break;
00095             case 11:
00096                 buttonGroup = 8;
00097                 buttonLocation = JOY_DOWN;
00098                 break;
00099             default:
00100                 break;
00101         }
00102         currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
00103     }
00104     else {
00105         // button is on LCD
00106         if (button == LCD_LEFT)
00107             currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);
00108
00109         if (button == LCD_CENT)
00110             currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);
```

```

00111     if (button == LCD_RIGHT)
00112         currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
00113     }
00114     return currentButton;
00115 }
00116
00117
00118 bool buttonIsNewPress(button_t button) {
00119     bool currentButton = buttonGetState(button);
00120
00121     if (!currentButton) // buttons is not currently pressed
00122         buttonPressed[button] = false;
00123
00124     if (currentButton && !buttonPressed[button]) {
00125         // button is currently pressed and was not detected as being pressed during last check
00126         buttonPressed[button] = true;
00127         return true;
00128     }
00129     else return false; // button is not pressed or was already detected
00130 }
```

5.85 src/vlib.c File Reference

#include "vlib.h"
 Include dependency graph for vlib.c:



Functions

- void **ftoa_bad** (float a, char *buffer, int precision)
converts a float to string.
- int **itoa_bad** (int a, char *buffer, int digits)
converts a int to string.
- void **reverse** (char *str, int len)
reverses a string 'str' of length 'len'

5.85.1 Function Documentation

5.85.1.1 ftoa_bad()

```
void ftoa_bad (
    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

Author

Christian DeSimone

Date

9/26/2017

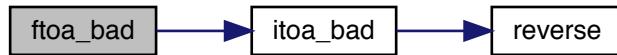
Definition at line **30** of file **vlib.c**.

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

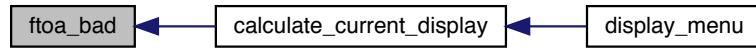
Referenced by [calculate_current_display\(\)](#).

```
00030
00031 // Extract integer part
00032 int ipart = (int)a;
00033
00034 // Extract floating part
00035 float fpart = a - (float)ipart;
00036
00037 // convert integer part to string
00038 int i = itoa_bad(ipart, buffer, 0);
00039
00040 // check for display option after point
00041 if(precision != 0) {
00042     buffer[i] = '.';
00043
00044     // Get the value of fraction part up to given num.
00045     // of points after dot. The third parameter is needed
00046     // to handle cases like 233.007
00047     fpart = fpart * pow(10, precision);
00048
00049     itoa_bad((int)fpart, buffer + i + 1, precision);
00050 }
00051 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.85.1.2 itoa_bad()

```
int itoa_bad (
    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, Christian DeSimone

Date

9/9/2017

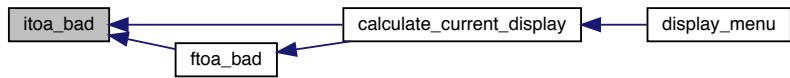
Definition at line 13 of file vlib.c.References **reverse()**.Referenced by **calculate_current_display()**, and **ftoa_bad()**.

```
00013     int i = 0;
00014     while (a) {
00015         buffer[i++] = (a%10) + '0';
00016         a = a/10;
00017     }
00018
00019 // If number of digits required is more, then
00020 // add 0s at the beginning
00021 while (i < digits)
00022     buffer[i++] = '0';
00023
00024
00025 reverse(buffer, i);
00026 buffer[i] = '\0';
00027 return i;
00028 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.85.1.3 reverse()

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

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

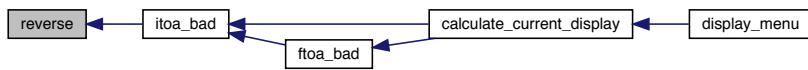
<i>str</i>	the string to reverse
<i>len</i>	the length

Definition at line **3** of file **vlib.c**.

Referenced by **itoa_bad()**.

```
00003     int i=0, j=len-1, temp;
00004     while (i<j) {
00005         temp = str[i];
00006         str[i] = str[j];
00007         str[j] = temp;
00008         i++; j--;
00009     }
00010 }
```

Here is the caller graph for this function:



5.86 vlib.c

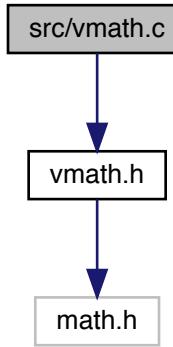
```
00001 #include "vlib.h"
00002
00003 void reverse(char *str, int len) {
00004     int i=0, j=len-1, temp;
00005     while (i<j) {
```

```
00006     temp = str[i];
00007     str[i] = str[j];
00008     str[j] = temp;
00009     i++; j--;
00010 }
00011 }
00012
00013 int itoa_bad(int a, char *buffer, int digits) {
00014     int i = 0;
00015     while (a) {
00016         buffer[i++] = (a%10) + '0';
00017         a = a/10;
00018     }
00019
00020     // If number of digits required is more, then
00021     // add 0s at the beginning
00022     while (i < digits)
00023         buffer[i++] = '0';
00024
00025     reverse(buffer, i);
00026     buffer[i] = '\0';
00027     return i;
00028 }
00029
00030 void ftoa_bad(float a, char *buffer, int precision) {
00031     // Extract integer part
00032     int ipart = (int)a;
00033
00034     // Extract floating part
00035     float fpart = a - (float)ipart;
00036
00037     // convert integer part to string
00038     int i = itoa_bad(ipart, buffer, 0);
00039
00040     // check for display option after point
00041     if(precision != 0) {
00042         buffer[i] = '.';
00043         // add dot
00044         // Get the value of fraction part up to given num.
00045         // of points after dot. The third parameter is needed
00046         // to handle cases like 233.007
00047         fpart = fpart * pow(10, precision);
00048
00049         itoa_bad((int)fpart, buffer + i + 1, precision);
00050     }
00051 }
```

5.87 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 cordinated 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.
- int **max** (int a, int b)

the min of two values
- int **min** (int a, int b)

the min of two values
- double **sind** (double angle)

sine of a angle in degrees

5.87.1 Function Documentation

5.87.1.1 cartesian_cord_to_polar()

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

Function to convert x and y 2 dimensional cartesian cordinated 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 \(p. 16\)](#)

[cord \(p. 6\)](#)

Definition at line 33 of file **vmath.c**.

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

```
00033     {  
00034     return cartesian_to_polar(cords.x, cords.y);  
00035 }
```

Here is the call graph for this function:



5.87.1.2 cartesian_to_polar()

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

Function to convert x and y 2 dimensional cartesian coordinates 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 (p. 16)

Definition at line 3 of file **vmath.c**.

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

Referenced by **cartesian_cord_to_polar()**.

```

00003                                     {
00004     float degree = 0;
00005     double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
00006
00007     if(x < 0){
00008         degree += 180.0;
00009     }
00010     else if(x > 0 && y < 0){
00011         degree += 360.0;
00012     }
00013
00014     if(x != 0 && y != 0){
00015         degree += atan((float)y / (float)x);
00016     }
00017     else if(x == 0 && y > 0){
00018         degree = 90.0;
00019     }
00020     else if(y == 0 && x < 0){
00021         degree = 180.0;
00022     }
00023     else if(x == 0 && y < 0){
00024         degree = 270.0;
00025     }
00026
00027     struct polar_cord p;
00028     p.angle = degree;
00029     p.magnitue = magnitude;
00030     return p;
00031 }
```

Here is the caller graph for this function:



5.87.1.3 max()

```
int max (
    int a,
    int b )
```

the min of two values

Parameters

<i>a</i>	the first
<i>b</i>	the second

Returns

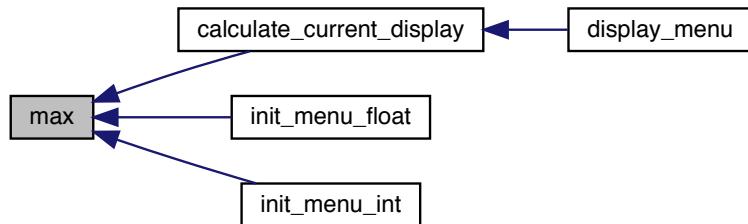
the smaller of a and b

Definition at line **48** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
00048
00049     if(a > b)  return a;
00050     return b;
00051 }
```

Here is the caller graph for this function:



5.87.1.4 min()

```
int min (
    int a,
    int b )
```

the min of two values

Parameters

<i>a</i>	the first
<i>b</i>	the second

Returns

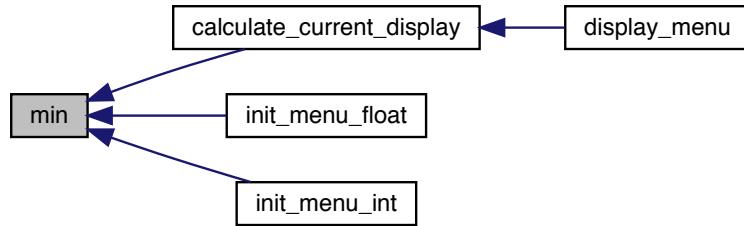
the smaller of *a* and *b*

Definition at line **42** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
00042
00043     if(a < b)  return a;
00044     return b;
00045 }
```

Here is the caller graph for this function:

**5.87.1.5 sind()**

```
double sind (
    double angle )
```

sine of a angle in degrees

Definition at line **37** of file **vmath.c**.

References **M_PI**.

```
00037
00038     double angleradians = angle * M_PI / 180.0f;
00039     return sin(angleradians);
00040 }
```

5.88 vmath.c

```
00001 #include "vmath.h"
00002
00003 struct polar_cord cartesian_to_polar(float x, float y) {
00004     float degree = 0;
00005     double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
00006
00007     if(x < 0){
00008         degree += 180.0;
00009     }
00010     else if(x > 0 && y < 0){
00011         degree += 360.0;
00012     }
00013
00014     if(x != 0 && y != 0){
00015         degree += atan((float)y / (float)x);
00016     }
00017     else if(x == 0 && y > 0){
00018         degree = 90.0;
00019     }
00020     else if(y == 0 && x < 0){
00021         degree = 180.0;
00022     }
00023     else if(x == 0 && y < 0){
00024         degree = 270.0;
00025     }
00026
00027     struct polar_cord p;
00028     p.angle = degree;
00029     p.magnitude = magnitude;
00030     return p;
00031 }
00032
00033 struct polar_cord cartesian_cord_to_polar(struct cord cords) {
00034     return cartesian_to_polar(cords.x, cords.y);
00035 }
00036
00037 double sind(double angle) {
00038     double angleradians = angle * M_PI / 180.0f;
00039     return sin(angleradians);
00040 }
00041
00042 int min(int a, int b) {
00043     if(a < b) return a;
00044     return b;
00045 }
00046
00047
00048 int max(int a, int b) {
00049     if(a > b) return a;
00050     return b;
00051 }
```

