

librobot-api Reference Manual

Generated by Doxygen 1.3.2

Wed Aug 20 15:56:44 2003

Contents

1	librobot-api Compound Index	1
1.1	librobot-api Compound List	1
2	librobot-api File Index	3
2.1	librobot-api File List	3
3	librobot-api Page Index	5
3.1	librobot-api Related Pages	5
4	librobot-api Class Documentation	7
4.1	bhv_connection_t Struct Reference	7
4.2	bhv_data_t Struct Reference	8
4.3	bhv_handle_t Struct Reference	9
4.4	bhv_t Struct Reference	10
4.5	devfs_set_t Struct Reference	11
4.6	freq_req_val_t Struct Reference	12
4.7	mc_lib_t Struct Reference	13
4.8	robot_handle_t Struct Reference	14
4.9	robot_net_msg_t Struct Reference	16
4.10	seq_msgbuf_t Struct Reference	17
5	librobot-api File Documentation	19
5.1	robot.h File Reference	19
5.2	robot/behavior.h File Reference	20
5.3	robot/constants.h File Reference	26
5.4	robot/devfs.h File Reference	29
5.5	robot/handle.h File Reference	38
5.6	robot/hwid.h File Reference	40
5.7	robot/mclib.h File Reference	44
5.8	robot/motors.h File Reference	46

5.9	robot/net.h File Reference	49
5.10	robot/sensors.h File Reference	52
5.11	robot/seq.h File Reference	56
5.12	robot/sys.h File Reference	63
5.13	robot/time.h File Reference	66
5.14	robot/types.h File Reference	68
5.15	robot/util.h File Reference	70
6	librobot-api Page Documentation	73
6.1	Compilation Instructions	73
6.2	Librobot Extra Documentation	75
6.3	General Robot Software Specifications	76
6.4	Todo List	79

Chapter 1

librobot-api Compound Index

1.1 librobot-api Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

bhv_connection_t	7
bhv_data_t	8
bhv_handle_t	9
bhv_t	10
devfs_set_t	11
freq_req_val_t	12
mc_lib_t	13
robot_handle_t	14
robot_net_msg_t	16
seq_msgbuf_t	17

Chapter 2

librobot-api File Index

2.1 librobot-api File List

Here is a list of all files with brief descriptions:

robot.h (Global librobot header file (all high-level robot programs should include this))	19
robot/ behavior.h (Definitions to be used in the creation of new reactive behaviors) . .	20
robot/ constants.h (Global constants for the robot, should be available to any and all robot software)	26
robot/ devfs.h (Device-related definitions for the /dev/robot device tree)	29
robot/ handle.h (Robot_handle_t and associated functionality)	38
robot/ hwid.h (Id numbers for hardware devices talking through serial)	40
robot/ mclib.h (Definitions related to loading a motor control shared library)	44
robot/ motors.h (Send commands to the motors from high-level software, and read sensor data related to the motors)	46
robot/ net.h (Methods for setting up network control of robots)	49
robot/ sensors.h (Send frequency commands to sensors and get sensor values (for sonar, ir, bump sensors))	52
robot/ seq.h (Remote control of reactive behaviors (through the sequencer))	56
robot/ sys.h (Robot system initialization and shutdown)	63
robot/ time.h (Precision timing methods for librobot)	66
robot/ types.h (Global types and data sizes for robot software. Mostly for lower-level stuff)	68
robot/ util.h (Miscellaneous utilities useful both internally to librobot and to external applications using it)	70

Chapter 3

librobot-api Page Index

3.1 librobot-api Related Pages

Here is a list of all related documentation pages:

Compilation Instructions	73
Librobot Extra Documentation	75
General Robot Software Specifications	76
Todo List	79

Chapter 4

librobot-api Class Documentation

4.1 bhv_connection_t Struct Reference

```
#include <seq.h>
```

Public Attributes

- `bhv_handle_t * from`
- `bhv_handle_t * to`
- `uint8_t from_key`
- `uint8_t to_key`

4.1.1 Member Data Documentation

4.1.1.1 `bhv_handle_t* bhv_connection_t::from`

behavior sending output

4.1.1.2 `uint8_t bhv_connection_t::from_key`

output key to send

4.1.1.3 `bhv_handle_t* bhv_connection_t::to`

behavior receiving input

4.1.1.4 `uint8_t bhv_connection_t::to_key`

input key to send to

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

- `robot/seq.h`

4.2 bhv_data_t Struct Reference

```
#include <seq.h>
```

Public Attributes

- `uint8_t key`
- `uint8_t data [ROBOT_MAX_BHV_DATA_SIZE-1]`

4.2.1 Member Data Documentation

4.2.1.1 `uint8_t bhv_data_t::data[ROBOT_MAX_BHV_DATA_SIZE - 1]`

4.2.1.2 `uint8_t bhv_data_t::key`

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

- `robot/seq.h`

4.3 bhv_handle_t Struct Reference

```
#include <seq.h>
```

Public Attributes

- pid_t pid
- int qid

4.3.1 Detailed Description

behavior handle

4.3.2 Member Data Documentation

4.3.2.1 pid_t bhv_handle_t::pid

behavior process id

4.3.2.2 int bhv_handle_t::qid

msg queue id for behavior

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

- robot/seq.h

4.4 bhv_t Struct Reference

```
#include <behavior.h>
```

Public Attributes

- `bhv_cleanup_func_t` `cleanup`
- `bhv_inhibit_func_t` `on_inhibit`
- `bhv_uninhibit_func_t` `on_uninhibit`
- `bhv_main_func_t` `main`
- `uint32_t` `flags`

4.4.1 Detailed Description

filled in, for the most part, by the behavior itself in order to tell the sequencer how to communicate with it

4.4.2 Member Data Documentation

4.4.2.1 `bhv_cleanup_func_t` `bhv_t::cleanup`

4.4.2.2 `uint32_t` `bhv_t::flags`

flags for initialization and execution; can use `bhv_flags_t` flags as well as any flags sent to `robot_init`

4.4.2.3 `bhv_main_func_t` `bhv_t::main`

4.4.2.4 `bhv_inhibit_func_t` `bhv_t::on_inhibit`

4.4.2.5 `bhv_uninhibit_func_t` `bhv_t::on_uninhibit`

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

- `robot/behavior.h`

4.5 devfs_set_t Struct Reference

```
#include <devfs.h>
```

Public Attributes

- uint8_t **hwid**
- int **stream**
- int **change**
- int **current**
- int **ctl**

4.5.1 Detailed Description

file descriptors for the /dev/robot stuff

4.5.2 Member Data Documentation

4.5.2.1 int devfs_set_t::change

4.5.2.2 int devfs_set_t::ctl

file descriptors

4.5.2.3 int devfs_set_t::current

4.5.2.4 uint8_t devfs_set_t::hwid

hardware id ([hwid.h](#))

4.5.2.5 int devfs_set_t::stream

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

- robot/[devfs.h](#)

4.6 freq_req_val_t Struct Reference

```
#include <types.h>
```

Public Attributes

- `owner_val_t` owner
- `freq_val_t` freq

4.6.1 Member Data Documentation

4.6.1.1 `freq_val_t` `freq_req_val_t::freq`

4.6.1.2 `owner_val_t` `freq_req_val_t::owner`

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

- `robot/types.h`

4.7 mc_lib_t Struct Reference

```
#include <mclib.h>
```

Public Attributes

- void * handle
- mc_init_func_t mc_init
- mc_shutdown_func_t mc_shutdown
- mc_start_frame_func_t mc_start_frame
- mc_set_velocity_func_t mc_set_velocity
- mc_get_velocity_func_t mc_get_velocity
- mc_set_odometry_func_t mc_set_odometry
- mc_do_control_func_t mc_do_control

4.7.1 Detailed Description

structure containing pointers to all library functions

4.7.2 Member Data Documentation

4.7.2.1 void* mc_lib_t::handle

dlopen handle

4.7.2.2 mc_do_control_func_t mc_lib_t::mc_do_control

4.7.2.3 mc_get_velocity_func_t mc_lib_t::mc_get_velocity

4.7.2.4 mc_init_func_t mc_lib_t::mc_init

4.7.2.5 mc_set_odometry_func_t mc_lib_t::mc_set_odometry

4.7.2.6 mc_set_velocity_func_t mc_lib_t::mc_set_velocity

4.7.2.7 mc_shutdown_func_t mc_lib_t::mc_shutdown

4.7.2.8 mc_start_frame_func_t mc_lib_t::mc_start_frame

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

- robot/mclib.h

4.8 robot_handle_t Struct Reference

```
#include <handle.h>
```

Public Attributes

- `devfs_set_t devfs_fds` [NUM_DEVICE_DIRS]
- `robot_id_t id`
- `char name` [ROBOT_MAX_NAME_LEN]
- `uint8_t init_complete`
- `uint32_t flags`
- `uint32_t devfs_flags`
- `int sock`
- `sockaddr_in sockaddr`
- `uint32_t timeout`

4.8.1 Detailed Description

handle for identifying which robot to talk to, and how to do the talking

4.8.2 Member Data Documentation

4.8.2.1 `devfs_set_t robot_handle_t::devfs_fds`[NUM_DEVICE_DIRS]

A list of structures with file descriptor information that is used to read from and write to each /dev/robot entry. Every hardware id has an associated stream, change and current entry, and optionally a ctl entry, in the /dev/robot filesystem.

When communicating with a robot over the network, all file descriptors will be the same as the network socket (sock).

4.8.2.2 `uint32_t robot_handle_t::devfs_flags`

passed to devfs_init

4.8.2.3 `uint32_t robot_handle_t::flags`

passed to robot_init

4.8.2.4 `robot_id_t robot_handle_t::id`

unique robot id number

4.8.2.5 `uint8_t robot_handle_t::init_complete`

has robot_init been called successfully?

4.8.2.6 char robot_handle_t::name[ROBOT_MAX_NAME_LEN]

robot name

4.8.2.7 int robot_handle_t::sock

network socket if necessary

4.8.2.8 struct sockaddr_in robot_handle_t::sockaddr**4.8.2.9 uint32_t robot_handle_t::timeout**

network timeout

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

- robot/handle.h

4.9 robot_net_msg_t Struct Reference

```
#include <net.h>
```

Public Attributes

- `uint8_t hwid`
- `uint8_t devfs_type`
- `uint32_t mid`
- `uint32_t count`
- `uint8_t * buf`

4.9.1 Member Data Documentation

4.9.1.1 `uint8_t* robot_net_msg_t::buf`

data buffer for message

4.9.1.2 `uint32_t robot_net_msg_t::count`

size of data in buf (bytes)

4.9.1.3 `uint8_t robot_net_msg_t::devfs_type`

device type (`devfs.h`)

4.9.1.4 `uint8_t robot_net_msg_t::hwid`

hardware id from `hwid.h`

4.9.1.5 `uint32_t robot_net_msg_t::mid`

managed internally

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

- `robot/net.h`

4.10 seq_msgbuf_t Struct Reference

```
#include <seq.h>
```

Public Attributes

- long **mtype**
- uint8_t **cmd**
- uint8_t **data** [ROBOT_MAX_BHV_DATA_SIZE]

4.10.1 Detailed Description

message queue buffer

4.10.2 Member Data Documentation

4.10.2.1 uint8_t seq_msgbuf_t::cmd

command to send (see enum)

4.10.2.2 uint8_t seq_msgbuf_t::data[ROBOT_MAX_BHV_DATA_SIZE]

4.10.2.3 long seq_msgbuf_t::mtype

should always be pid of sender

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

- robot/seq.h

Chapter 5

librobot-api File Documentation

5.1 robot.h File Reference

Global librobot header file (all high-level robot programs should include this).

```
#include <robot/config.h>
#include <robot/constants.h>
#include <robot/types.h>
#include <robot/handle.h>
#include <robot/sys.h>
#include <robot/time.h>
#include <robot/mclib.h>
#include <robot/hwid.h>
#include <robot/devfs.h>
#include <robot/net.h>
#include <robot/motors.h>
#include <robot/sensors.h>
#include <robot/util.h>
#include <robot/seq.h>
```

5.1.1 Detailed Description

Global librobot header file (all high-level robot programs should include this).

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

robot.h,v 1.15 2003/08/01 19:39:26 beevek Exp

5.2 robot/behavior.h File Reference

Definitions to be used in the creation of new reactive behaviors.

```
#include <robot/types.h>
#include <robot/seq.h>
#include <stdio.h>
```

Compounds

- struct **bhv_t**

Defines

- #define **bhv_printf(s...)**
Print to sequencer's logfile with some nice formatting (our name, pid).
- #define **bhv_perror(s)**
Print error message to sequencer's logfile with some nice formatting (our name, pid).

Typedefs

- typedef void(* **bhv_cleanup_func_t**)(void)
Called on behavior cleanup.
- typedef int(* **bhv_data_func_t**)(uint8_t key, const void *data, int size)
Called whenever new data has arrived from an outside process for one of the inputs of the behavior.
- typedef void(* **bhv_inhibit_func_t**)(void)
Called whenever the behavior is about to be inhibited.
- typedef void(* **bhv_uninhibit_func_t**)(void)
Called whenever the behavior is about to be uninhibited.
- typedef int(* **bhv_main_func_t**)(void)
The main function of a behavior.

Enumerations

- enum **bhv_flags_t** { **NO_ROBOT_INIT** = (1 << 31), **QUEUE_WHEN_INHIBITED** = (1 << 30) }

Functions

- int **bhv_init** (bhv_t *bhv)
Entry point into the behavior.
- int **bhv_add_input** (uint8_t key, bhv_data_func_t ondata)
Add an input key to accept input from.
- int **bhv_output** (uint8_t key, const void *data, int size)
Send output to all behaviors that we are connected to through the specified output key.
- int **bhv_input_pop** (uint8_t key, void *buf, int size)
Extract input data from an input key's queue.
- void **bhv_shutdown** (int exitval)
Shut down gracefully. Never call exit! Always call this instead!
- void **bhv_block_forever** (void)
Block forever (handling EINTR).

Variables

- bhv_handle_t **bhv_self**
- const char * **bhv_name**

5.2.1 Detailed Description

Definitions to be used in the creation of new reactive behaviors.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

behavior.h,v 1.7 2003/08/01 16:45:19 beevek Exp

Other programs should not include this file. It is part of libbehavior and should only be used by behaviors themselves in order to specify their interface.

5.2.2 Define Documentation

5.2.2.1 #define bhv_perror(s)

Value:

```
do { \
    fprintf(stderr, "<%s (%d)> Error: ", bhv_name, bhv_self.pid); \
    perror(s); \
    fflush(stderr); \
} while(0);
```

Print error message to sequencer's logfile with some nice formatting (our name, pid).

5.2.2.2 #define bhv_printf(s...)

Value:

```
do { \
    printf("<%s (%d)> ", bhv_name, bhv_self.pid); \
    printf(s); \
    fflush(stdout); \
} while(0);
```

Print to sequencer's logfile with some nice formatting (our name, pid).

5.2.3 Typedef Documentation

5.2.3.1 typedef void(* bhv_cleanup_func_t)(void)

Called on behavior cleanup.

Should perform any necessary de-initialization of the behavior.

5.2.3.2 typedef int(* bhv_data_func_t)(uint8_t key, const void *data, int size)

Called whenever new data has arrived from an outside process for one of the inputs of the behavior.

Should handle the data in whatever manner is necessary.

Use `bhv_add_input` to associate one of these functions with data arriving for a specific input key.

Parameters:

- key* Input key of data
- data* Pointer to data buffer
- size* Size of data buffer (bytes)

Returns:

- < 0 on failure, >= 0 on success

5.2.3.3 typedef void(* bhv_inhibit_func_t)(void)

Called whenever the behavior is about to be inhibited.

Note that the behavior is not responsible for inhibiting itself, this is taken care of by `libbehavior` after this function has been called.

5.2.3.4 typedef int(* bhv_main_func_t)(void)

The main function of a behavior.

This function should perform whatever behavior-related actions are necessary. In most cases it should loop forever. It does not need to worry about starting and stopping when the behavior is inhibited, this is done automatically by `libbehavior`.

If this function is executing, the behavior can assume it is not currently inhibited.

IMPORTANT: this function must be interruptible. In other words, it should check failed function calls to see whether `errno` is `EINTR`, and handle this situation gracefully. This is because the main function is interrupted by signals when data arrives or the behavior is inhibited.

Returns:

< 0 on failure, >= 0 on success.

5.2.3.5 typedef void(* bhv_uninhibit_func_t)(void)

Called whenever the behavior is about to be uninhibited.

Note that the behavior is not responsible for uninhibiting itself, this is taken care of by `libbehavior` after this function has been called.

5.2.4 Enumeration Type Documentation

5.2.4.1 enum bhv_flags_t

`bhv_t` flags

Enumeration values:

`NO_ROBOT_INIT` `robot_{init,shutdown}` not called

`QUEUE_WHEN_INHIBITED` queue incoming data when paused, and call `ondata` when unpaused for each queued item (if `ondata` is set)

5.2.5 Function Documentation

5.2.5.1 int bhv_add_input (uint8_t *key*, bhv_data_func_t *ondata*)

Add an input key to accept input from.

When data is sent to this behavior with a matching input key, call the `ondata` function with the data. Note that any number of `ondata` functions can be called for a single key. However, data will only be added to the internal queue once, no matter how many times you've called `bhv_add_input` with `ondata` as null.

Parameters:

key Input key to accept

ondata Function to call when data arrives for this input key. If null, the data will be stored in an internal queue and the behavior can access it by calling `bhv_input_pop` with the correct key.

Returns:

< 0 on failure, >= 0 on success

See also:

`bhv_input_pop`

5.2.5.2 void bhv_block_forever (void)

Block forever (handling EINTR).

Useful if you need to define a main behavior function that does a few things initially but then just wants to sleep quietly forever while data handlers are called.

Warning:

Never returns!

5.2.5.3 int bhv_init (bhv_t * bhv)

Entry point into the behavior.

All behaviors must define this function. It should fill in the values of bhv as appropriate, and perform any other initialization (such as starting sub-behaviors, etc).

Parameters:

bhv Pointer to a **bhv_t** struct to be initialized by the behavior

Returns:

< 0 on failure, >= 0 on success

5.2.5.4 int bhv_input_pop (uint8_t key, void * buf, int size)

Extract input data from an input key's queue.

If data has been sent to our input labeled key, and if no ondata function was specified for key in the call to `bhv_add_input`, the data is placed in a queue for that input and is accessible through this function.

Parameters:

key Input key to check for queued data

buf Buffer in which to place data if found

size Size of buffer (in bytes); if waiting data is larger than size, an error is generated and ERANGE is set

Returns:

< 0 on failure, or the size of the data placed in buf on success; if no items are on the specified queue, -1 is returned and ENODATA is set

5.2.5.5 int bhv_output (uint8_t key, const void * data, int size)

Send output to all behaviors that we are connected to through the specified output key.

Searches for key in the internal connection table. Any behaviors that are registered to receive input from an output from this behavior with the matching key will be sent data.

Parameters:

key Output key to send to

data Data buffer to send

size Length of data (in bytes)

See also:

`seq_connect`

5.2.5.6 void bhv_shutdown (int *exitval*)

Shut down gracefully. Never call exit! Always call this instead!

Parameters:

exitval Value passed to exit

5.2.6 Variable Documentation

5.2.6.1 const char* bhv_name

the name of this behavior

5.2.6.2 bhv_handle_t bhv_self

handle referring to ourself

5.3 robot/constants.h File Reference

Global constants for the robot, should be available to any and all robot software.

Defines

- #define **ROBOT_NUM_SONAR** 10
- #define **ROBOT_NUM_IR** 10
- #define **ROBOT_NUM_BUMP** 6
- #define **ROBOT_WHEEL_RADIUS** 0.076
- #define **ROBOT_WHEEL_RADIUS_L** ROBOT_WHEEL_RADIUS
- #define **ROBOT_WHEEL_RADIUS_R** ROBOT_WHEEL_RADIUS
- #define **ROBOT_AXLE_WIDTH** 0.37
- #define **ROBOT_ENCODER_STEPS_PER_REV** 4096
- #define **ROBOT_SONAR_MULTIPLIER** 274.4e-6
- #define **ROBOT_IR_MULTIPLIER** 0
- #define **ROBOT_HW_TTY** "/dev/tts/1"
- #define **ROBOT_THRESH_ODOM_X** 0.01
- #define **ROBOT_THRESH_ODOM_Y** 0.01
- #define **ROBOT_THRESH_ODOM_THETA** 0.01
- #define **ROBOT_THRESH_VEL_V** 0.001
- #define **ROBOT_THRESH_VEL_W** 0.001
- #define **ROBOT_THRESH_SONAR** 0.01
- #define **ROBOT_THRESH_IR** 0.01
- #define **ROBOT_THRESH_TRANSLATE** 0.03
- #define **ROBOT_THRESH_ROTATE** 0.03
- #define **ROBOT_DEFAULT_VEL_V** 0.2
- #define **ROBOT_DEFAULT_VEL_W** 0.4
- #define **ROBOT_MAX_VEL_V** 1.0
- #define **ROBOT_MAX_VEL_W** 3.14
- #define **ROBOT_DEFAULT_SONAR_FREQ** 4.0 /*! hertz */
- #define **ROBOT_DEFAULT_IR_FREQ** 4.0 /*! hertz */
- #define **ROBOT_MAX_NAME_LEN** 128
- #define **ROBOT_DEFAULT_NET_TIMEOUT** 2000
- #define **ROBOT_DEFAULT_NET_PORT** 10798

5.3.1 Detailed Description

Global constants for the robot, should be available to any and all robot software.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

constants.h,v 1.19 2003/07/24 20:53:13 beevek Exp

Todo

Most of these values need to be appropriately set.

5.3.2 Define Documentation

5.3.2.1 `#define ROBOT_AXLE_WIDTH 0.37`

5.3.2.2 `#define ROBOT_DEFAULT_IR_FREQ 4.0 /*! hertz */`

5.3.2.3 `#define ROBOT_DEFAULT_NET_PORT 10798`

5.3.2.4 `#define ROBOT_DEFAULT_NET_TIMEOUT 2000`

milliseconds

5.3.2.5 `#define ROBOT_DEFAULT_SONAR_FREQ 4.0 /*! hertz */`

5.3.2.6 `#define ROBOT_DEFAULT_VEL_V 0.2`

m/s

5.3.2.7 `#define ROBOT_DEFAULT_VEL_W 0.4`

m/s

5.3.2.8 `#define ROBOT_ENCODER_STEPS_PER_REV 4096`

number of encoder steps in one wheel revolution

5.3.2.9 `#define ROBOT_HW_TTY "/dev/tts/1"`

5.3.2.10 `#define ROBOT_IR_MULTIPLIER 0`

5.3.2.11 `#define ROBOT_MAX_NAME_LEN 128`

5.3.2.12 `#define ROBOT_MAX_VEL_V 1.0`

m/s

5.3.2.13 `#define ROBOT_MAX_VEL_W 3.14`

m/s

5.3.2.14 `#define ROBOT_NUM_BUMP 6`

must be ≤ 8

5.3.2.15 `#define ROBOT_NUM_IR 10`

must be ≤ 16

5.3.2.16 `#define ROBOT_NUM_SONAR 10`

must be ≤ 16

5.3.2.17 `#define ROBOT_SONAR_MULTIPLIER 274.4e-6`

multiply `sonar_time_val_t`, `ir_voltage_val_t` by these to get dist in meters

5.3.2.18 `#define ROBOT_THRESH_IR 0.01`

hertz

5.3.2.19 `#define ROBOT_THRESH_ODOM_THETA 0.01`

meters

5.3.2.20 `#define ROBOT_THRESH_ODOM_X 0.01`

meters

5.3.2.21 `#define ROBOT_THRESH_ODOM_Y 0.01`

meters

5.3.2.22 `#define ROBOT_THRESH_ROTATE 0.03`

5.3.2.23 `#define ROBOT_THRESH_SONAR 0.01`

hertz

5.3.2.24 `#define ROBOT_THRESH_TRANSLATE 0.03`

5.3.2.25 `#define ROBOT_THRESH_VEL_V 0.001`

m/s

5.3.2.26 `#define ROBOT_THRESH_VEL_W 0.001`

m/s

5.3.2.27 `#define ROBOT_WHEEL_RADIUS 0.076`

5.3.2.28 `#define ROBOT_WHEEL_RADIUS_L ROBOT_WHEEL_RADIUS`

5.3.2.29 `#define ROBOT_WHEEL_RADIUS_R ROBOT_WHEEL_RADIUS`

5.4 robot/devfs.h File Reference

Device-related definitions for the /dev/robot device tree.

```
#include <robot/constants.h>
#include <robot/types.h>
#include <robot/hwid.h>
```

Compounds

- struct `devfs_set_t`

Defines

- #define `NUM_DEVICE_DIRS` (7 + `ROBOT_NUM_SONAR` + `ROBOT_NUM_IR` + `ROBOT_NUM_BUMP`)
- #define `NUM_DEVICES` (26 + (`ROBOT_NUM_SONAR` * 4) + (`ROBOT_NUM_IR` * 4) + (`ROBOT_NUM_BUMP` * 3))
- #define `DEVFS_IOCLOCKCTL` 0xc301
- #define `DEVFS_IOCUNLOCKCTL` 0xc302
- #define `DEVFS_IOCGETLOCKOWNER` 0xc303
- #define `DATA_SIZE_ODOM` (3 * `sizeof(odom_val_t)`)
- #define `CTL_SIZE_ODOM` (3 * `sizeof(odom_val_t)`)
- #define `DATA_SIZE_VEL` (2 * `sizeof(vel_val_t)`)
- #define `CTL_SIZE_VEL` (2 * `sizeof(vel_val_t)`)
- #define `DATA_SIZE_ENCODER` (2 * `sizeof(encoder_val_t)`)
- #define `CTL_SIZE_ENCODER` 0
- #define `DATA_SIZE_PWM` (2 * `sizeof(pwm_val_t)`)
- #define `CTL_SIZE_PWM` 0
- #define `DATA_SIZE_SONAR` (`sizeof(sonar_val_t)`)
- #define `CTL_SIZE_SONAR` (`sizeof(freq_req_val_t)`)
- #define `DATA_SIZE_IR` (`sizeof(ir_val_t)`)
- #define `CTL_SIZE_IR` (`sizeof(freq_req_val_t)`)
- #define `DATA_SIZE_BUMP` (`sizeof(bump_val_t)`)
- #define `CTL_SIZE_BUMP` 0
- #define `DEVFS_ROOT_STR` "robot"
- #define `DEVFS_ODOM_STR` "robot/odom"
- #define `DEVFS_VEL_STR` "robot/vel"
- #define `DEVFS_ENCODER_STR` "robot/encoder"
- #define `DEVFS_PWM_STR` "robot/pwm"
- #define `DEVFS_SONAR_STR` "robot/sonar"
- #define `DEVFS_IR_STR` "robot/ir"
- #define `DEVFS_BUMP_STR` "robot/bump"
- #define `DEV_TYPE_STREAM_STR` "stream"
- #define `DEV_TYPE_CHANGE_STR` "change"
- #define `DEV_TYPE_CURRENT_STR` "current"
- #define `DEV_TYPE_CTL_STR` "ctl"

Enumerations

- enum `devfs_dir_type_t` {
`DEVFS_ODOM = 0`, `DEVFS_VEL`, `DEVFS_ENCODER`, `DEVFS_PWM`,
`DEVFS_SONAR`, `DEVFS_SONAR_SINGLE`, `DEVFS_IR`, `DEVFS_IR_SINGLE`,
`DEVFS_BUMP`, `DEVFS_BUMP_SINGLE` }
- enum `devfs_type_t` { `DEV_TYPE_STREAM = 0`, `DEV_TYPE_CHANGE`, `DEV_TYPE_CURRENT`, `DEV_TYPE_CTL` }
- enum { `DEVFS_SERVER = (1 << 0)`, `DEVFS_CLIENT = (1 << 1)`, `DEVFS_RDONLY = (1 << 2)` }
- enum {
`DEVFS_PRIO_SUPER = 0x00000000`, `DEVFS_PRIO_CRITICAL = 0x0000000f`,
`DEVFS_PRIO_HIGH = 0x000000ff`, `DEVFS_PRIO_NORMAL = 0x0000ffff`,
`DEVFS_PRIO_LOW = 0x00ffffff` }

Functions

- int `devfs_init` (int flags)
Initialize the devfs subsystem.
- void `devfs_cleanup` (void)
De-initialize the devfs subsystem.
- `devfs_set_t * devfs_find` (uint8_t hwid)
Find the `devfs_set_t` for the specified hardware id in the current robot's `devfs_fds` list.
- int `devfs_get_data_size` (uint8_t hwid, `devfs_type_t` type)
Figure out the size of the data to read or write from the `/dev/robot` entry matching hwid and type.
- int `devfs_write` (`devfs_set_t` *set, const void *buf, int count)
Write the data in buf to any of set's open file descriptors that are opened for writing.
- int `devfs_read` (`devfs_set_t` *set, `devfs_type_t` type, void *buf, int count)
Read count bytes into buf from set's file descriptor matching the specified `devfs_type_t`.
- int `devfs_wait_for_change` (uint8_t *hwids, int count)
Wait until new data arrives on the change device for one of the hardware ids in hwids.
- int `devfs_lock_ctl` (`devfs_set_t` *set, uint32_t prio)
Lock the ctl entry for a hardware device.
- int `devfs_unlock_ctl` (`devfs_set_t` *set, uint32_t prio)
Unlock the ctl entry for a hardware device.
- int `devfs_get_lock_owner` (`devfs_set_t` *set)
Get the pid of the owner of the current lock on a devfs ctl entry, if one is set.

5.4.1 Detailed Description

Device-related definitions for the /dev/robot device tree.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

devfs.h, v 1.11 2003/08/01 15:30:43 beevek Exp

5.4.2 Define Documentation

5.4.2.1 `#define CTL_SIZE_BUMP 0`

5.4.2.2 `#define CTL_SIZE_ENCODER 0`

5.4.2.3 `#define CTL_SIZE_IR (sizeof(freq_req_val_t))`

5.4.2.4 `#define CTL_SIZE_ODOM (3 * sizeof(odom_val_t))`

5.4.2.5 `#define CTL_SIZE_PWM 0`

5.4.2.6 `#define CTL_SIZE_SONAR (sizeof(freq_req_val_t))`

5.4.2.7 `#define CTL_SIZE_VEL (2 * sizeof(vel_val_t))`

5.4.2.8 `#define DATA_SIZE_BUMP (sizeof(bump_val_t))`

5.4.2.9 `#define DATA_SIZE_ENCODER (2 * sizeof(encoder_val_t))`

5.4.2.10 `#define DATA_SIZE_IR (sizeof(ir_val_t))`

5.4.2.11 `#define DATA_SIZE_ODOM (3 * sizeof(odom_val_t))`

5.4.2.12 `#define DATA_SIZE_PWM (2 * sizeof(pwm_val_t))`

5.4.2.13 `#define DATA_SIZE_SONAR (sizeof(sonar_val_t))`

5.4.2.14 `#define DATA_SIZE_VEL (2 * sizeof(vel_val_t))`

5.4.2.15 `#define DEV_TYPE_CHANGE_STR "change"`

5.4.2.16 `#define DEV_TYPE_CTL_STR "ctl"`

5.4.2.17 `#define DEV_TYPE_CURRENT_STR "current"`

5.4.2.18 `#define DEV_TYPE_STREAM_STR "stream"`

5.4.2.19 `#define DEVFS_BUMP_STR "robot/bump"`

5.4.2.20 `#define DEVFS_ENCODER_STR "robot/encoder"`

5.4.2.21 `#define DEVFS_IOCGETLOCKOWNER 0xc303`

5.4.2.22 `#define DEVFS_IOCLOCKCTL 0xc301`

ioctl's for /dev/robot driver. Not really the proper way to do these but simplifies things significantly.

5.4.2.23 `#define DEVFS_IOCUNLOCKCTL 0xc302`

5.4.2.24 `#define DEVFS_IR_STR "robot/ir"`

5.4.2.25 `#define DEVFS_ODOM_STR "robot/odom"`

5.4.2.26 `#define DEVFS_PWM_STR "robot/pwm"`

5.4.2.27 `#define DEVFS_ROOT_STR "robot"`

5.4.2.28 `#define DEVFS_SONAR_STR "robot/sonar"`

5.4.2.29 `#define DEVFS_VEL_STR "robot/vel"`

5.4.2.30 `#define NUM_DEVICE_DIRS (7 + ROBOT_NUM_SONAR +
ROBOT_NUM_IR + ROBOT_NUM_BUMP)`

Warning:

if new devices are added this MUST BE CHANGED

5.4.2.31 `#define NUM_DEVICES (26 + (ROBOT_NUM_SONAR * 4) +
(ROBOT_NUM_IR * 4) + (ROBOT_NUM_BUMP * 3))`

Warning:

if new devices are added this MUST BE CHANGED

5.4.3 Enumeration Type Documentation

5.4.3.1 anonymous enum

devfs initialization options

Enumeration values:

`DEVFS_SERVER`

`DEVFS_CLIENT`

`DEVFS_RDONLY`

5.4.3.2 anonymous enum

priorities for `devfs_lock_ctl` and `devfs_unlock_ctl`

Enumeration values:

`DEVFS_PRIO_SUPER`

`DEVFS_PRIO_CRITICAL`

`DEVFS_PRIO_HIGH`

`DEVFS_PRIO_NORMAL` always use this!

`DEVFS_PRIO_LOW`

5.4.3.3 enum devfs_dir_type_t

some numbers to identify device "directory types" by

Enumeration values:

DEVFS_ODOM

DEVFS_VEL

DEVFS_ENCODER

DEVFS_PWM

DEVFS_SONAR e.g. /dev/robot/sonar

DEVFS_SONAR_SINGLE e.g. /dev/robot/sonar/0

DEVFS_IR

DEVFS_IR_SINGLE

DEVFS_BUMP

DEVFS_BUMP_SINGLE

5.4.3.4 enum devfs_type_t

device file types

Enumeration values:

DEV_TYPE_STREAM /dev/robot/.../stream

DEV_TYPE_CHANGE /dev/robot/.../change

DEV_TYPE_CURRENT /dev/robot/.../current

DEV_TYPE_CTL /dev/robot/.../ctl

5.4.4 Function Documentation

5.4.4.1 void devfs_cleanup (void)

De-initialize the devfs subsystem.

5.4.4.2 devfs_set_t* devfs_find (uint8_t *hwid*)

Find the **devfs_set_t** for the specified hardware id in the current robot's **devfs_fds** list.

Parameters:

hwid A hardware id from **hwid.h**

Returns:

A pointer to the **devfs_set_t** matching *hwid* from the current robot's **devfs_fds** list

5.4.4.3 int devfs_get_data_size (uint8_t *hwid*, devfs_type_t *type*)

Figure out the size of the data to read or write from the /dev/robot entry matching *hwid* and *type*.

Parameters:

hwid A hardware id from `hwid.h`

type The type of device

Returns:

Data size for the device, or zero on failure (no matching device exists)

5.4.4.4 int devfs_get_lock_owner (devfs_set_t * *set*)

Get the pid of the owner of the current lock on a devfs ctl entry, if one is set.

Parameters:

set The `devfs_set_t` from the current robot's `devfs_fds` list to get the lock owner of

Returns:

< 0 on failure or if no lock is set, and the pid of the process owning the lock otherwise

5.4.4.5 int devfs_init (int *flags*)

Initialize the devfs subsystem.

Parameters:

flags A bitfield of flags to control the initialization

Returns:

< 0 on failure, >= 0 on success

5.4.4.6 int devfs_lock_ctl (devfs_set_t * *set*, uint32_t *prio*)

Lock the ctl entry for a hardware device.

If the priority is higher than that for the current owner of a lock on the device, or if no lock currently exists, the requester is granted exclusive write access to the device and only a higher-priority lock or unlock request will be granted.

Parameters:

set The `devfs_set_t` from the current robot's `devfs_fds` list to set the lock on

prio The priority of the lock; the lower this value, the higher the lock priority. In general, `DEVFS_PRIO_NORMAL` should be used.

Returns:

< 0 on failure, >= 0 on success

5.4.4.7 `int devfs_read (devfs_set_t * set, devfs_type_t type, void * buf, int count)`

Read count bytes into buf from set's file descriptor matching the specified devfs_type_t.

You can only read data of exactly the correct size from a /dev/robot device, otherwise the read will fail.

Parameters:

set The `devfs_set_t` from the current robot's `devfs_fds` list to read from

type The device type from the set to read from

buf Buffer in which the data will be placed

count Number of bytes to read

Returns:

< 0 on failure, >= 0 on success

5.4.4.8 `int devfs_unlock_ctl (devfs_set_t * set, uint32_t prio)`

Unlock the ctl entry for a hardware device.

If the priority is higher than that for the current owner of a lock on the device, or the requester is the owner of the lock, then the lock is removed and any process can write to the device.

Parameters:

set The `devfs_set_t` from the current robot's `devfs_fds` list to unlock

prio The priority of the lock; the lower this value, the higher the lock priority. In general, `DEVFS_PRIO_NORMAL` should be used.

Returns:

< 0 on failure, >= 0 on success

5.4.4.9 `int devfs_wait_for_change (uint8_t * hwids, int count)`

Wait until new data arrives on the change device for one of the hardware ids in hwids.

This function will block until the /dev/robot/.../change entry for one of the specified hwids has data waiting to be read. Note that if you have never read data from the change device before, data will be waiting.

Parameters:

hwids An array of hardware ids from `hwid.h`

count The size of the hwids array (number of hwids)

Returns:

< 0 on failure, >= 0 on success

5.4.4.10 int devfs_write (devfs_set_t * *set*, const void * *buf*, int *count*)

Write the data in *buf* to any of *set*'s open file descriptors that are opened for writing.

You can only write data of exactly the correct size to a */dev/robot* device, otherwise the write will fail.

Parameters:

set The `devfs_set_t` from the current robot's `devfs_fds` list to write to

buf Buffer containing the data to write

count Number of bytes to write

Returns:

< 0 on failure, >= 0 on success

5.5 robot/handle.h File Reference

robot_handle_t and associated functionality

```
#include <robot/types.h>
#include <robot/devfs.h>
#include <netinet/in.h>
```

Compounds

- struct **robot_handle_t**

Defines

- #define **handle_is_net**(h) ((h) → sock > 0 || (h) → sockaddr.sin_addr.s_addr)
- #define **handle_is_loc**(h) (!handle_is_net(h))
- #define **handle_is_connected**(h) (handle_is_net(h) && (h) → init_complete)

Functions

- int **robot_set_handle** (**robot_handle_t** *handle)
Set the current handle for all robot operations.

Variables

- **robot_handle_t** *cur_robot

5.5.1 Detailed Description

robot_handle_t and associated functionality

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

handle.h,v 1.4 2003/07/18 01:13:11 beevek Exp

robot_handle_t is used primarily for communicating with robots via the network. In particular, it allows one program to control multiple robots by switching between handles. In general, any program that runs directly on the robot will never need to worry about handles at all. In fact, any program that is meant to control only one robot (even over the network) will never have to use this functionality. Only programs that talk to multiple robots will make use of it.

5.5.2 Define Documentation

5.5.2.1 #define handle_is_connected(h) (handle_is_net(h) && (h) → init_complete)

Nonzero if the handle represents a robot somewhere else on the network, and we have successfully connected to it and initialized it. The h parameter must be a **robot_handle_t** *.

5.5.2.2 #define handle_is_loc(h) (!handle_is_net(h))

Nonzero if the handle represents a robot on this machine, zero if it is somewhere else on the network. The h parameter must be a **robot_handle_t** *.

5.5.2.3 #define handle_is_net(h) ((h) → sock > 0 || (h) → sockaddr.sin_addr.s_addr)

Nonzero if the handle represents a robot somewhere else on the network, zero if it is on this machine. The h parameter must be a **robot_handle_t** *.

5.5.3 Function Documentation

5.5.3.1 int robot_set_handle (robot_handle_t * handle)

Set the current handle for all robot operations.

Passing null will make all operations act on the local computer (i.e., the program must be running on the robot itself).

Parameters:

handle A valid **robot_handle_t**, or null for local robot

Returns:

< 0 on failure, >= 0 on success

5.5.4 Variable Documentation

5.5.4.1 robot_handle_t* cur_robot

global handle telling api who to talk to (sys.c)

5.6 robot/hwid.h File Reference

Id numbers for hardware devices talking through serial.

```
#include <robot/constants.h>
```

Defines

- #define **HW_IS_MC**(d) (d == HW_MC)
- #define **HW_IS_MOTORS**(d) (d == HW_MOTORS)
- #define **HW_IS_VEL**(d) (d == HW_VEL)
- #define **HW_IS_ODOM**(d) (d == HW_ODOM)
- #define **HW_IS_ENCODER**(d) (d == HW_ENCODER)
- #define **HW_IS_PWM**(d) (d == HW_PWM)
- #define **HW_IS_ALL_SONAR**(d) (d == HW_ALL_SONAR)
- #define **HW_IS_ALL_IR**(d) (d == HW_ALL_IR)
- #define **HW_IS_ALL_BUMP**(d) (d == HW_ALL_BUMP)
- #define **HW_IS_SONAR**(d) (d >= HW_SONAR01 && d <= HW_SONAR01 + ROBOT_NUM_SONAR)
- #define **HW_IS_IR**(d) (d >= HW_IR01 && d <= HW_IR01 + ROBOT_NUM_IR)
- #define **HW_IS_BUMP**(d) (d >= HW_BUMP01 && d <= HW_BUMP01 + ROBOT_NUM_BUMP)
- #define **HW_IS_VIRTUAL**(d)

Enumerations

- enum {
 - HW_MC** = 0x00, **HW_MOTORS** = 0x01, **HW_VEL** = 0x02, **HW_ODOM** = 0x03,
 - HW_ENCODER** = 0x04, **HW_PWM** = 0x05, **HW_ALL_SONAR** = 0x06, **HW_ALL_IR** = 0x07,
 - HW_ALL_BUMP** = 0x08, **HW_SONAR01** = 0x10, **HW_SONAR02** = 0x11, **HW_SONAR03** = 0x12,
 - HW_SONAR04** = 0x13, **HW_SONAR05** = 0x14, **HW_SONAR06** = 0x15, **HW_SONAR07** = 0x16,
 - HW_SONAR08** = 0x17, **HW_SONAR09** = 0x18, **HW_SONAR10** = 0x19, **HW_SONAR11** = 0x1a,
 - HW_SONAR12** = 0x1b, **HW_SONAR13** = 0x1c, **HW_SONAR14** = 0x1d, **HW_SONAR15** = 0x1e,
 - HW_SONAR16** = 0x1f, **HW_IR01** = 0x20, **HW_IR02** = 0x21, **HW_IR03** = 0x22,
 - HW_IR04** = 0x23, **HW_IR05** = 0x24, **HW_IR06** = 0x25, **HW_IR07** = 0x26,
 - HW_IR08** = 0x27, **HW_IR09** = 0x28, **HW_IR10** = 0x29, **HW_IR11** = 0x2a,
 - HW_IR12** = 0x2b, **HW_IR13** = 0x2c, **HW_IR14** = 0x2d, **HW_IR15** = 0x2e,
 - HW_IR16** = 0x2f, **HW_BUMP01** = 0x30, **HW_BUMP02** = 0x31, **HW_BUMP03** = 0x32,
 - HW_BUMP04** = 0x33, **HW_BUMP05** = 0x34, **HW_BUMP06** = 0x35, **HW_BUMP07** = 0x36,
 - HW_BUMP08** = 0x37, **HW_MIN** = 0x00, **HW_MAX** = 0x38 }

Hardware device ids.

5.6.1 Detailed Description

Id numbers for hardware devices talking through serial.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

hwid.h, v 1.7 2003/07/18 15:27:22 beevek Exp

Also used to identify hardware internally throughout librobot.

Defined here as well are a number of macros useful for determining the nature of a hardware id. These should be self-explanatory for the most part.

5.6.2 Define Documentation

5.6.2.1 `#define HW_IS_ALL_BUMP(d) (d == HW_ALL_BUMP)`

5.6.2.2 `#define HW_IS_ALL_IR(d) (d == HW_ALL_IR)`

5.6.2.3 `#define HW_IS_ALL_SONAR(d) (d == HW_ALL_SONAR)`

5.6.2.4 `#define HW_IS_BUMP(d) (d >= HW_BUMP01 && d <= HW_BUMP01 + ROBOT_NUM_BUMP)`

5.6.2.5 `#define HW_IS_ENCODER(d) (d == HW_ENCODER)`

5.6.2.6 `#define HW_IS_IR(d) (d >= HW_IR01 && d <= HW_IR01 + ROBOT_NUM_IR)`

5.6.2.7 `#define HW_IS_MC(d) (d == HW_MC)`

5.6.2.8 `#define HW_IS_MOTORS(d) (d == HW_MOTORS)`

5.6.2.9 `#define HW_IS_ODOM(d) (d == HW_ODOM)`

5.6.2.10 `#define HW_IS_PWM(d) (d == HW_PWM)`

5.6.2.11 `#define HW_IS_SONAR(d) (d >= HW_SONAR01 && d <= HW_SONAR01 + ROBOT_NUM_SONAR)`

5.6.2.12 `#define HW_IS_VEL(d) (d == HW_VEL)`

5.6.2.13 `#define HW_IS_VIRTUAL(d)`

Value:

```
(HW_IS_VEL(d) || HW_IS_ODOM(d) \
    || HW_IS_ENCODER(d) || HW_IS_PWM(d) \
    || HW_IS_ALL_SONAR(d) || HW_IS_ALL_IR(d) \
    || HW_IS_BUMP(d))
```

Returns:

Nonzero only if d has NO matching real hardware device (i.e. it is completely handled in software).

5.6.3 Enumeration Type Documentation

5.6.3.1 anonymous enum

Hardware device ids.

These numbers have two uses: they act as device identifiers when talking to the microcontroller controlling the hardware, and they also are used throughout the system to identify the hardware for dev/robot entries. Thus, every devfs entry has a corresponding hardware device id, even if there is no "real" corresponding hardware device. These virtual HW devices are those that are marked "not sent from serial" below.

Enumeration values:

HW_MC

HW_MOTORS

HW_VEL not sent from serial

HW_ODOM not sent from serial

HW_ENCODER not sent from serial

HW_PWM not sent from serial

HW_ALL_SONAR not sent from serial

HW_ALL_IR not sent from serial

HW_ALL_BUMP

HW_SONAR01

HW_SONAR02

HW_SONAR03

HW_SONAR04

HW_SONAR05

HW_SONAR06

HW_SONAR07

HW_SONAR08

HW_SONAR09

HW_SONAR10

HW_SONAR11

HW_SONAR12

HW_SONAR13

HW_SONAR14

HW_SONAR15

HW_SONAR16

HW_IR01

HW_IR02

HW_IR03

HW_IR04
HW_IR05
HW_IR06
HW_IR07
HW_IR08
HW_IR09
HW_IR10
HW_IR11
HW_IR12
HW_IR13
HW_IR14
HW_IR15
HW_IR16
HW_BUMP01 none of these are ever sent from serial
HW_BUMP02
HW_BUMP03
HW_BUMP04
HW_BUMP05
HW_BUMP06
HW_BUMP07
HW_BUMP08
HW_MIN
HW_MAX one greater than max

5.7 robot/mclib.h File Reference

Definitions related to loading a motor control shared library.

```
#include <robot/types.h>
```

Compounds

- struct `mc_lib_t`

Typedefs

- typedef int(* `mc_init_func_t`)(uint32_t)
- typedef void(* `mc_shutdown_func_t`)(void)
- typedef int(* `mc_start_frame_func_t`)(encoder_val_t, encoder_val_t)
- typedef int(* `mc_set_velocity_func_t`)(vel_val_t, vel_val_t)
- typedef void(* `mc_get_velocity_func_t`)(vel_val_t *, vel_val_t *)
- typedef void(* `mc_set_odometry_func_t`)(odom_val_t *, odom_val_t *, odom_val_t *)
- typedef int(* `mc_do_control_func_t`)(pwm_val_t *, pwm_val_t *)

Functions

- int `mc_lib_load` (const char *lib_file, `mc_lib_t` *lib)
Load a motor control shared library and initialize lib to point to its functions.
- void `mc_lib_unload` (`mc_lib_t` *lib)
Unload a motor control shared library.

5.7.1 Detailed Description

Definitions related to loading a motor control shared library.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

`mclib.h`, v 1.4 2003/07/18 01:13:11 beevek Exp

This is mostly used just by the interpreter; in general most user programs will not have to worry about this stuff.

Here we specify typedefs for functions that must be provided by any motor control library. For more information, see doc/motor_control_interface.txt and mc/mc.h.

5.7.2 Typedef Documentation

5.7.2.1 typedef int(* mc_do_control_func_t)(pwm_val_t *, pwm_val_t *)

5.7.2.2 typedef void(* mc_get_velocity_func_t)(vel_val_t *, vel_val_t *)

5.7.2.3 typedef int(* mc_init_func_t)(uint32_t)

5.7.2.4 typedef void(* mc_set_odometry_func_t)(odom_val_t *, odom_val_t *, odom_val_t *)

5.7.2.5 typedef int(* mc_set_velocity_func_t)(vel_val_t, vel_val_t)

5.7.2.6 typedef void(* mc_shutdown_func_t)(void)

5.7.2.7 typedef int(* mc_start_frame_func_t)(encoder_val_t, encoder_val_t)

5.7.3 Function Documentation

5.7.3.1 int mc_lib_load (const char * *lib_file*, mc_lib_t * *lib*)

Load a motor control shared library and initialize lib to point to its functions.

Parameters:

lib_file Path to the library shared object file

lib mc_lib_t to fill in with pointers to the library's functions

Returns:

< 0 on failure, >= 0 on success

5.7.3.2 void mc_lib_unload (mc_lib_t * *lib*)

Unload a motor control shared library.

Parameters:

lib An mc_lib_t previously initialized with mc_lib_load

5.8 robot/motors.h File Reference

Send commands to the motors from high-level software, and read sensor data related to the motors.

```
#include <robot/types.h>
#include <robot/constants.h>
```

Functions

- int **robot_set_velocity** (**vel_val_t** v, **vel_val_t** w)
Set translational and rotational velocity of the robot.
- int **robot_translate** (float dist, **vel_val_t** v)
Translate (straight forward or backward) dist meters at velocity v.
- int **robot_rotate** (float radians, **vel_val_t** w)
Rotate the specified distance in radians at rotational velocity w.
- int **robot_set_odometry** (**odom_val_t** x, **odom_val_t** y, **odom_val_t** theta)
Reset the robot's internal odometry counters to the specified values.
- int **robot_get_velocity** (**vel_val_t** *v, **vel_val_t** *w)
Get the robot's current translational and rotational velocities.
- int **robot_get_odometry** (**odom_val_t** *x, **odom_val_t** *y, **odom_val_t** *theta)
Get the robot's current odometry counter values.
- int **robot_lock_motors** (void)
Lock velocity control of the motors to this process.
- int **robot_unlock_motors** (void)
Unlock velocity control of the motors.

5.8.1 Detailed Description

Send commands to the motors from high-level software, and read sensor data related to the motors.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

motors.h,v 1.6 2003/08/01 15:30:43 beevek Exp

Todo

FIXME: acceleration?

5.8.2 Function Documentation

5.8.2.1 int robot_get_odometry (odom_val_t * *x*, odom_val_t * *y*, odom_val_t * *theta*)

Get the robot's current odometry counter values.

Parameters:

x Pointer to location to store x-component

y Pointer to location to store y-component

theta Pointer to location to store theta-component (rotation)

Returns:

< 0 on failure, >= 0 on success

5.8.2.2 int robot_get_velocity (vel_val_t * *v*, vel_val_t * *w*)

Get the robot's current translational and rotational velocities.

Parameters:

v Pointer to location to store the robot's translational velocity (meters/sec)

w Pointer to location to store the robot's rotational velocity (radians/sec)

Returns:

< 0 on failure, >= 0 otherwise

5.8.2.3 int robot_lock_motors (void)

Lock velocity control of the motors to this process.

When a process has locked control of the motors, no other process can control them (unless it has a higher priority – e.g. the emergency-stop behavior).

Returns:

< 0 on failure, >= 0 on success

See also:

robot_unlock_motors

5.8.2.4 int robot_rotate (float *radians*, vel_val_t *w*)

Rotate the specified distance in radians at rotational velocity *w*.

This function quits if it detects that the robot is for some reason turning improperly.

Parameters:

radians Distance (in radians) to rotate; positive for counterclockwise rotation, negative for clockwise rotation

w Rotational velocity (radians/sec)

Returns:

< 0 on failure, >= 0 otherwise

5.8.2.5 int robot_set_odometry (odom_val_t *x*, odom_val_t *y*, odom_val_t *theta*)

Reset the robot's internal odometry counters to the specified values.

Parameters:

- x* x-component odometry value
- y* y-component odometry value
- theta* theta-component (rotation) odometry value

Returns:

< 0 on failure, >= 0 otherwise

5.8.2.6 int robot_set_velocity (vel_val_t *v*, vel_val_t *w*)

Set translational and rotational velocity of the robot.

Parameters:

- v* Translational velocity (in meters/second)
- w* Rotational velocity (in radians/second)

Returns:

< 0 on failure, >= 0 otherwise

5.8.2.7 int robot_translate (float *dist*, vel_val_t *v*)

Translate (straight forward or backward) *dist* meters at velocity *v*.

If for some reason the robot moves farther away from its goal during the course of this call, it fails and sets *errno* to ESPIPE (Illegal seek :)

Parameters:

- dist* Distance (in meters) to translate; positive to go forward, negative to go backward
- v* Translational velocity (meters/sec)

Returns:

< 0 on failure, >= 0 otherwise

5.8.2.8 int robot_unlock_motors (void)

Unlock velocity control of the motors.

In general, only call this after locking control of the motors with `robot_lock_motors`. If you do not call this before your program quits, `robot_shutdown` will take care of it for you.

Returns:

< 0 on failure, >= 0 on success

See also:

`robot_lock_motors`

5.9 robot/net.h File Reference

Methods for setting up network control of robots.

```
#include <robot/types.h>
#include <robot/handle.h>
```

Compounds

- struct **robot_net_msg_t**

Enumerations

- enum {
MSG_PING = 0xffffffff, MSG_INIT = 0x01, MSG_SHUTDOWN = 0x02, MSG_WAIT_FOR_CHANGE = 0x03,
MSG_LOCK_CTL = 0xff, MSG_UNLOCK_CTL = 0xfe, MSG_GET_LOCK_OWNER = 0xfd, MSG_ERROR = 0x00 }

Functions

- int **robot_net_set_timeout** (**robot_handle_t** *handle, int32_t ms)
Set the network timeout for the specified handle, in milliseconds.
- int **robot_net_find** (**robot_handle_t** **handles, uint32_t timeout_ms)
Search for network-controllable robots on the local subnet.
- const char * **robot_get_ip_str** (const **robot_handle_t** *handle)
Get a string containing the ip address of the robot pointed to by handle.

5.9.1 Detailed Description

Methods for setting up network control of robots.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

net.h,v 1.7 2003/07/31 22:30:03 beevek Exp

In order to control a robot via the network, the netrobotd program must be running on it!

5.9.2 Enumeration Type Documentation

5.9.2.1 anonymous enum

special "devfs_types" that we send with hwid 0 to control netrobotd

Enumeration values:

MSG_PING udp only
 MSG_INIT
 MSG_SHUTDOWN
 MSG_WAIT_FOR_CHANGE
 MSG_LOCK_CTL
 MSG_UNLOCK_CTL
 MSG_GET_LOCK_OWNER
 MSG_ERROR

5.9.3 Function Documentation**5.9.3.1 const char* robot_get_ip_str (const robot_handle_t * handle)**

Get a string containing the ip address of the robot pointed to by handle.

Parameters:

handle A pointer to a network-controllable **robot_handle_t**

Returns:

Pointer to a STATICALLY allocated string containing the ip address. Do not attempt to deallocate this string. Note that it will be overwritten by subsequent calls.

5.9.3.2 int robot_net_find (robot_handle_t ** handles, uint32_t timeout_ms)

Search for network-controllable robots on the local subnet.

handles will be allocated and must be properly freed by the caller.

Note that after this call, `robot_init` must still be called with each handle to actually begin talking to the robot.

Parameters:

handles Pointer to a pointer that will be set the the memory location of an array of `robot_handle_t`'s that have been initialized with network information for discovered robots

timeout_ms The time to wait for robots to report after sending a discovery broadcast

Returns:

Number of discovered robots, zero if none are found; < 0 on failure.

5.9.3.3 int robot_net_set_timeout (robot_handle_t * handle, int32_t ms)

Set the network timeout for the specified handle, in milliseconds.

This will affect all read and write operations, as well as connection attempts.

Parameters:

handle Pointer to a robot handle

ms Time, in milliseconds, of timeout. If less than zero, the default timeout is used. If equal to zero, there is no timeout (operations will block forever). If greater than zero, exactly this value is used.

Returns:

< 0 on failure, >= 0 on success

5.10 robot/sensors.h File Reference

Send frequency commands to sensors and get sensor values (for sonar, ir, bump sensors).

```
#include <robot/types.h>
#include <robot/constants.h>
#include <robot/hwid.h>
```

Functions

- **int robot_force_reset_all_sensors** (void)
Demand that all sensors stop firing regardless of what other processes have requested.
- **int robot_set_sensor_freq** (uint8_t hwid, **freq_val_t** freq)
Generic method to set any sonar or ir sensor's frequency.
- **int robot_set_sonar_freq** (uint8_t hwid, **freq_val_t** freq)
Set sonar firing frequency for the specified sonar hardware id.
- **int robot_set_all_sonar_freq** (**freq_val_t** freq)
Set all sonar firing frequencies to the same value.
- **int robot_set_ir_freq** (uint8_t hwid, **freq_val_t** freq)
Set infrared firing frequency for the specified ir hardware id.
- **int robot_set_all_ir_freq** (**freq_val_t** freq)
Set all infrared firing frequencies to the same value.
- **sonar_val_t robot_get_sonar** (uint8_t hwid)
Get current range value from a single sonar.
- **int robot_get_all_sonar** (**sonar_val_t** ranges[ROBOT_NUM_SONAR])
Place all current sonar ranges in a buffer.
- **ir_val_t robot_get_ir** (uint8_t hwid)
Get current range value from a single ir sensor.
- **int robot_get_all_ir** (**ir_val_t** ranges[ROBOT_NUM_IR])
Place all current infrared ranges in a buffer.
- **bump_val_t robot_get_bump** (uint8_t hwid)
Get current bump toggle from a single bump sensor.
- **int robot_get_all_bump** (**bump_val_t** toggles[ROBOT_NUM_BUMP])
Place all current bump toggle values in a buffer.

5.10.1 Detailed Description

Send frequency commands to sensors and get sensor values (for sonar, ir, bump sensors).

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

sensors.h, v 1.4 2003/07/18 01:13:12 beevek Exp

Setting the frequency at which a sonar or ir sensor fires does not guarantee that the sensor will fire at exactly that frequency. Instead, it guarantees that the sensor will fire at AT LEAST that frequency. If the requested frequency is not possible, the sensor will fire as quickly as it is able.

To set the frequency of sonar number 3, for example, do: `robot_set_sonar_freq(HW_SONAR03, frequency);`

5.10.2 Function Documentation

5.10.2.1 `int robot_force_reset_all_sensors (void)`

Demand that all sensors stop firing regardless of what other processes have requested.

Returns:

< 0 on failure, >= 0 on success

5.10.2.2 `int robot_get_all_bump (bump_val_t toggles[ROBOT_NUM_BUMP])`

Place all current bump toggle values in a buffer.

Parameters:

toggles Array in which toggle values will be placed

Returns:

< 0 on failure, >= 0 otherwise

5.10.2.3 `int robot_get_all_ir (ir_val_t ranges[ROBOT_NUM_IR])`

Place all current infrared ranges in a buffer.

Parameters:

ranges Array in which range values will be placed

Returns:

< 0 on failure, >= 0 otherwise

5.10.2.4 `int robot_get_all_sonar (sonar_val_t ranges[ROBOT_NUM_SONAR])`

Place all current sonar ranges in a buffer.

Parameters:

ranges Array in which range values will be placed

Returns:

< 0 on failure, >= 0 otherwise

5.10.2.5 `bump_val_t robot_get_bump (uint8_t hwid)`

Get current bump toggle from a single bump sensor.

Parameters:

hwid A bump sensor hardware id from `hwid.h`

Returns:

Toggle value; 0 for off, nonzero for on

5.10.2.6 `ir_val_t robot_get_ir (uint8_t hwid)`

Get current range value from a single ir sensor.

Parameters:

hwid An ir hardware id from `hwid.h`

Returns:

An ir range; zero generally indicates a failure

5.10.2.7 `sonar_val_t robot_get_sonar (uint8_t hwid)`

Get current range value from a single sonar.

Parameters:

hwid A sonar hardware id from `hwid.h`

Returns:

A sonar range; zero generally indicates a failure

5.10.2.8 `int robot_set_all_ir_freq (freq_val_t freq)`

Set all infrared firing frequencies to the same value.

See also:

`robot_set_sensor_freq`

5.10.2.9 int robot_set_all_sonar_freq (freq_val_t *freq*)

Set all sonar firing frequencies to the same value.

See also:

`robot_set_sensor_freq`

5.10.2.10 int robot_set_ir_freq (uint8_t *hwid*, freq_val_t *freq*)

Set infrared firing frequency for the specified ir hardware id.

See also:

`robot_set_sensor_freq`

5.10.2.11 int robot_set_sensor_freq (uint8_t *hwid*, freq_val_t *freq*)

Generic method to set any sonar or ir sensor's frequency.

Parameters:

hwid A sonar or ir hardware id from `hwid.h`

freq A frequency in hertz

Returns:

< 0 on failure, >= 0 on success

5.10.2.12 int robot_set_sonar_freq (uint8_t *hwid*, freq_val_t *freq*)

Set sonar firing frequency for the specified sonar hardware id.

See also:

`robot_set_sensor_freq`

5.11 robot/seq.h File Reference

Remote control of reactive behaviors (through the sequencer).

```
#include <robot/types.h>
```

Compounds

- struct **bhv_connection_t**
- struct **bhv_data_t**
- struct **bhv_handle_t**
- struct **seq_msgbuf_t**

Defines

- #define **ROBOT_MAX_BHV_DATA_SIZE** 512
- #define **ROBOT_SEQ_QUEUE_KEY** 1
- #define **MSGBUF_BASE_SZ** (sizeof(uint8_t))

Enumerations

- enum {
SEQ_LOAD = 0x00, **SEQ_ATTACH** = 0x01, **SEQ_UNLOAD** = 0x02, **SEQ_ERR** = 0x03,
SEQ_LIST = 0x04, **BHV_INIT** = 0x10, **BHV_INHIBIT** = 0x11, **BHV_UNINHIBIT** = 0x12,
BHV_CONNECT = 0x13, **BHV_DISCONNECT** = 0x14, **BHV_DATA** = 0x15 }

Functions

- **bhv_handle_t * seq_load** (const char *name)
Load a reactive behavior and initialize it.
- **bhv_handle_t * seq_load_args** (const char *name, const char *args)
Load a reactive behavior and pass it commandline arguments.
- **bhv_handle_t * seq_load_net** (const char *name, const char *ip, uint16_t port)
Simple wrapper for seq_load_args that connects the behavior to a remote netrobotd.
- **bhv_handle_t * seq_attach** (const char *name)
Get the handle of an already-running behavior.
- **int seq_unload** (bhv_handle_t *bhv)
Unload a reactive behavior.
- **int seq_unload_all** (void)
Unload all loaded behaviors.

- int **seq_inhibit** (**bhv_handle_t** *bhv)
Inhibit ("pause") a behavior.
- int **seq_inhibit_all** (void)
Inhibit all loaded behaviors.
- int **seq_uninhibit** (**bhv_handle_t** *bhv)
Uninhibit ("unpause") a behavior.
- int **seq_uninhibit_all** (void)
Uninhibit all loaded behaviors.
- int **seq_send** (**bhv_handle_t** *bhv, uint8_t key, const void *data, int size)
Send (per-behavior) data directly to a behavior (without actually "connecting" to one of its inputs).
- int **seq_get** (uint8_t key, **bhv_data_t** *buf)
Wait for a behavior to send data to us, with input key key.
- int **seq_connect** (const **bhv_connection_t** *conn)
Connect an output of one behavior to an input of another.
- int **seq_disconnect** (const **bhv_connection_t** *conn)
Disconnect an output of one behavior from an input of another.
- int **seq_get_my_handle** (**bhv_handle_t** *handle)
Get a bhv_handle_t representing the calling process, even if it isn't a behavior.

5.11.1 Detailed Description

Remote control of reactive behaviors (through the sequencer).

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

seq.h,v 1.10 2003/08/20 19:43:51 beevek Exp

5.11.2 Define Documentation

5.11.2.1 `#define MSGBUF_BASE_SZ (sizeof(uint8_t))`

5.11.2.2 `#define ROBOT_MAX_BHV_DATA_SIZE 512`

maximum size of data that can be read from or written to a behavior

5.11.2.3 `#define ROBOT_SEQ_QUEUE_KEY 1`

message queue key for sequencer

5.11.3 Enumeration Type Documentation

5.11.3.1 anonymous enum

commands to send as part of a `seq_msgbuf_t`

Enumeration values:

`SEQ_LOAD`

`SEQ_ATTACH`

`SEQ_UNLOAD`

`SEQ_ERR`

`SEQ_LIST` internal use only

`BHV_INIT`

`BHV_INHIBIT`

`BHV_UNINHIBIT`

`BHV_CONNECT`

`BHV_DISCONNECT`

`BHV_DATA`

5.11.4 Function Documentation

5.11.4.1 `bhv_handle_t*` `seq_attach` (`const char *` *name*)

Get the handle of an already-running behavior.

Looks for the oldest already-running behavior with the specified name. If the caller has sufficient permissions, returns a handle to the behavior.

Sets EACCES if caller does not have permission to control the previously loaded behavior.

Parameters:

name Name of the behavior to attach to

Returns:

Pointer to a behavior handle for the behavior, or null on failure

See also:

`seq_load`

5.11.4.2 `int` `seq_connect` (`const bhv_connection_t *` *conn*)

Connect an output of one behavior to an input of another.

If the specified output and input are valid, all data sent to the specified output by the behavior `conn->from` will be forwarded to the specified input of `conn->to`.

Parameters:

conn Pointer to a structure specifying the connection information

Returns:

< 0 on failure, >= 0 on success

5.11.4.3 int seq_disconnect (const bhv_connection_t * conn)

Disconnect an output of one behavior from an input of another.

If the specified output and input are valid, the behavior conn->from will no longer forward data from the specified output to the input of conn->to.

Parameters:

conn Pointer to a structure specifying the connection information

Returns:

< 0 on failure, >= 0 on success

5.11.4.4 int seq_get (uint8_t key, bhv_data_t * buf)

Wait for a behavior to send data to us, with input key key.

Used to get data in a non-behavior, from a behavior's output. This function will block until data arrives for the specified key. Note that no data will ever arrive unless seq_connect is called to connect a behavior to the caller (use seq_get_my_handle when setting this up).

Warning:

IMPORANT: Behaviors should not use this function, only external programs controlling behaviors!

Parameters:

key Input key of data

buf Data input buffer

Returns:

< 0 on failure, >= 0 on success

5.11.4.5 int seq_get_my_handle (bhv_handle_t * handle)

Get a bhv_handle_t representing the calling process, even if it isn't a behavior.

Useful for setting up a connection with a real behavior's outputs (see seq_get).

Parameters:

handle Pointer to a bhv_handle_t to be filled in

Returns:

< 0 on failure, >= 0 on success

5.11.4.6 int seq_inhibit (bhv_handle_t * bhv)

Inhibit ("pause") a behavior.

Tells a behavior to stop processing until it is told to "uninhibit" itself. When a behavior is inhibited, any data sent to its inputs is either queued or discarded, according to options set by the behavior itself.

Parameters:

bhv Behavior handle from `seq_load`

Returns:

< 0 on failure, >= 0 otherwise

See also:

`seq_uninhibit`

5.11.4.7 int seq_inhibit_all (void)

Inhibit all loaded behaviors.

Returns:

< 0 on failure, >= 0 otherwise

See also:

`seq_inhibit`

5.11.4.8 bhv_handle_t* seq_load (const char * name)

Load a reactive behavior and initialize it.

Sets the following errno's on failure:

EINVAL: unable to communicate with the sequencer

EBUSY: program has loaded the maximum number of allowed behaviors

ERANGE: the data being sent to the sequencer is too large (i.e. the cwd + name are too long)

If the calling program has performed a `robot_init` with a remote netrobotd, `seq_load` will connect the behavior to the same remote robot.

Parameters:

name Name of the behavior to load. The sequencer searches its internal path for a behavior of this name. If not found, the current working directory of the process that called `seq_load` is also searched. If still not found, the function fails.

Returns:

Pointer to a behavior handle for the behavior, or null on failure.

See also:

`seq_unload` `seq_attach` `seq_load_args`

5.11.4.9 bhv_handle_t* seq_load_args (const char * name, const char * args)

Load a reactive behavior and pass it commandline arguments.

Same as `seq_load`, but passes commandline arguments to the behavior. Note that all behaviors that use `libbehavior` will treat the first commandline argument as an ip address/dns name, and the second as a port, used to connect to a netrobotd. Using this to load a behavior causes the `-i` and `-o` arguments to the sequencer to NOT be sent to the behavior. Generally this function should not be used, unless you have a VERY GOOD REASON for passing arguments to your behavior (rather than sending them as an "input").

See also:

`seq_load`

5.11.4.10 `bhv_handle_t* seq_load_net (const char * name, const char * ip, uint16_t port)`

Simple wrapper for `seq_load_args` that connects the behavior to a remote netrobotd.

When you use this instead of `seq_load`, the `-i` and `-o` arguments to the sequencer are not passed to the behavior. This function can be used to connect the behavior to a different ip/port than the calling program is connected to.

Parameters:

name Name of the behavior to load (see `seq_load`)

ip IP address for the behavior to connect to

port Remote port for the behavior to connect to

See also:

`seq_load` `seq_load_args`

5.11.4.11 `int seq_send (bhv_handle_t * bhv, uint8_t key, const void * data, int size)`

Send (per-behavior) data directly to a behavior (without actually "connecting" to one of its inputs).

Generally this is some sort of structure defined in a header file for the specific behavior. The user is responsible for making sure this is the right kind of data to send to the behavior.

Sets ERANGE if the size of the data is larger than allowed.

Parameters:

bhv Behavior handle from `seq_load`

key Input key of *bhv* to send to

data Pointer to data buffer

size Size (in bytes) of data buffer

Returns:

< 0 on failure, >= 0 on success

5.11.4.12 `int seq_uninhibit (bhv_handle_t * bhv)`

Uninhibit ("unpause") a behavior.

Tells a behavior it may begin processing again if it is currently paused.

Parameters:

bhv Behavior handle from `seq_load`

Returns:

< 0 on failure, >= 0 otherwise

See also:

`seq_inhibit`

5.11.4.13 int seq_uninhibit_all (void)

Uninhibit all loaded behaviors.

Returns:

< 0 on failure, >= 0 otherwise

See also:

seq_uninhibit

5.11.4.14 int seq_unload (bhv_handle_t * bhv)

Unload a reactive behavior.

Basically just decrements a usage count for the behavior. When this count reaches zero, the behavior is completely unloaded from the system. Note that it is not strictly necessary to call this since usage counts for behaviors are automatically decremented when the calling program exits.

Parameters:

bhv Behavior handle from seq_load

Returns:

< 0 on failure, >= 0 otherwise

See also:

seq_load

5.11.4.15 int seq_unload_all (void)

Unload all loaded behaviors.

Returns:

< 0 on failure, >= 0 otherwise

See also:

seq_unload

5.12 robot/sys.h File Reference

Robot system initialization and shutdown.

```
#include <robot/types.h>
#include <robot/handle.h>
```

Defines

- #define **robot_init_local**(flags) robot_init(flags, 0, 0, 0)
Initialize a local robot. Purely for convenience.

Enumerations

- enum **robot_init_flags_t** {
 RI_NO_SET_FREQS = (1 << 0), **RI_NO_HANDLE_SIGS** = (1 << 1), **RI_STOP_ON_QUIT** = (1 << 2), **RI_DEVFS_READ_ALL** = (1 << 3),
 RI_USE_SEQUENCER = (1 << 4), **RI_NO_DEVFS** = (1 << 5), **RI_DEFAULT** = **RI_STOP_ON_QUIT** }

Functions

- int **robot_init** (uint32_t flags, **robot_handle_t** *handle, const char *ip, uint16_t port)
Initialize the robot for use either locally or over a network.
- void **robot_shutdown** (void)
Shut down the robot currently in use.
- **robot_id_t** **robot_get_id** (void)
Get the unique id number of the current robot.
- const char * **robot_get_name** (void)
Get the name of the current robot.

5.12.1 Detailed Description

Robot system initialization and shutdown.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

sys.h,v 1.9 2003/07/29 18:03:58 beevek Exp

5.12.2 Define Documentation

5.12.2.1 `#define robot_init_local(flags) robot_init(flags, 0, 0, 0)`

Initialize a local robot. Purely for convenience.

This just calls `robot_init` with null handle, ip and port arguments (i.e., it will only succeed if the program is running on the robot itself).

See also:

`robot_init`

5.12.3 Enumeration Type Documentation

5.12.3.1 `enum robot_init_flags_t`

flags to be passed to `robot_init`

Enumeration values:

`RI_NO_SET_FREQS` don't request default sonar/ir freqs

`RI_NO_HANDLE_SIGS` don't handle signals to shut down cleanly

`RI_STOP_ON_QUIT` set velocity to zero in `robot_shutdown`

`RI_DEVFS_READ_ALL` open all devfs entries with `O_RDONLY`

`RI_USE_SEQUENCER` enable loading/unloading of reactive behaviors

`RI_NO_DEVFS` do not initialize devfs entries

`RI_DEFAULT` default flags; most programs should use this

5.12.4 Function Documentation

5.12.4.1 `robot_id_t robot_get_id (void)`

Get the unique id number of the current robot.

Returns:

The robot's id number, or 0 on failure

5.12.4.2 `const char* robot_get_name (void)`

Get the name of the current robot.

Returns:

A pointer to a string containing the name of the current robot, or null on failure. Do not deallocate this string yourself, it is internal.

5.12.4.3 int robot_init (uint32_t *flags*, robot_handle_t * *handle*, const char * *ip*, uint16_t *port*)

Initialize the robot for use either locally or over a network.

This function performs the following tasks:

If an ip address has been specified, or if ip information in the robot handle has already been filled in, initialize network communications

Set the current robot handle for all librobot functions to act on to be handle

If RLNO_HANDLE_SIGS is not set, register signal handler for cleanly shutting down the robot on receipt of SIGINT, SIGQUIT, SIGTERM or SIGSEGV

Initialize the devfs subsystem (if RLNO_DEVFS is not set)

Initialize sensors; if RLNO_SET_FREQS is not set, request that all sensors fire at the default frequency

If RL_USE_SEQUENCER is set, initialize sequencer IPC

Parameters:

flags Initialization flags (robot_init_flags_t)

handle A handle to use in the initialization. This is only necessary if your program will be controlling more than one robot. The handle's values will be filled in by robot_init.

ip An ip address string telling us where to connect if the robot is to be controlled over the network. This should be null if the robot will be controlled locally, or if handle's network information has already been filled in elsewhere (such as robot_net_find).

port The remote port to connect to if ip is non-null (ignored otherwise). Usually, this should be ROBOT_DEFAULT_NET_PORT from **constants.h**.

5.12.4.4 void robot_shutdown (void)

Shut down the robot currently in use.

Uses the internal pointer to the current robot handle to decide which robot to shut down.

5.13 robot/time.h File Reference

Precision timing methods for librobot.

```
#include <robot/types.h>
```

Defines

- `#define USEC_PER_SEC 1000000`
- `#define USEC_PER_MSEC 1000`
- `#define MSEC_PER_SEC 1000`
- `#define robot_time_to_float_sec(rt) ((float)((double)(rt) / (double)USEC_PER_SEC))`
- `#define robot_time_to_float_ms(rt) ((float)((double)(rt) / (double)USEC_PER_MSEC))`

Functions

- `int robot_get_time (robot_time_us_t *time)`
Get the current time since the Unix epoch in microseconds.
- `int robot_sleep (const robot_time_us_t *time)`
Sleep for the specified time (in microseconds).
- `int robot_alarm (const robot_time_us_t *time)`
Set an alarm to go off after the specified time (in microseconds). Similar to alarm().

5.13.1 Detailed Description

Precision timing methods for librobot.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

`time.h`, v 1.4 2003/07/18 01:13:12 beevek Exp

It is useful to have very precise timing information for certain tasks on the robot. We'll do this timing using microseconds.

5.13.2 Define Documentation

5.13.2.1 `#define MSEC_PER_SEC 1000`

5.13.2.2 `#define robot_time_to_float_ms(rt) ((float)((double)(rt) / (double)USEC_PER_MSEC))`

convert robot time to a float representing milliseconds

```
5.13.2.3 #define robot_time_to_float_sec(rt) ((float)((double)(rt) /  
      (double)USEC_PER_SEC))
```

convert robot time to a float representing seconds

```
5.13.2.4 #define USEC_PER_MSEC 1000
```

```
5.13.2.5 #define USEC_PER_SEC 1000000
```

5.13.3 Function Documentation

```
5.13.3.1 int robot_alarm (const robot_time_us_t * time)
```

Set an alarm to go off after the specified time (in microseconds). Similar to alarm().

Parameters:

time Pointer to a location with the time for the alarm

Returns:

< 0 on failure, >= 0 otherwise

```
5.13.3.2 int robot_get_time (robot_time_us_t * time)
```

Get the current time since the Unix epoch in microseconds.

Parameters:

time Pointer to a location to store the current time

Returns:

< 0 on failure, >= 0 otherwise

```
5.13.3.3 int robot_sleep (const robot_time_us_t * time)
```

Sleep for the specified time (in microseconds).

Parameters:

time Pointer to a location with the time to sleep

Returns:

< 0 on failure, >= 0 otherwise

5.14 robot/types.h File Reference

Global types and data sizes for robot software. Mostly for lower-level stuff.

```
#include <inttypes.h>
#include <sys/types.h>
#include <robot/config.h>
```

Compounds

- struct `freq_req_val_t`

Defines

- `#define` `ROBOT_BYTE_ORDER` `_BIG_ENDIAN`

Typedefs

- typedef float `odom_val_t`
- typedef float `vel_val_t`
- typedef int16_t `encoder_val_t`
- typedef float `sonar_val_t`
- typedef float `ir_val_t`
- typedef uint8_t `bump_val_t`
- typedef float `freq_val_t`
- typedef int8_t `pwm_val_t`
- typedef uint16_t `sonar_time_val_t`
- typedef uint16_t `ir_voltage_val_t`
- typedef uint8_t `bump_bitfield_val_t`
- typedef uint8_t `hw_freq_val_t`
- typedef pid_t `owner_val_t`
- typedef uint64_t `robot_time_us_t`
- typedef uint32_t `robot_id_t`

5.14.1 Detailed Description

Global types and data sizes for robot software. Mostly for lower-level stuff.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

`types.h`, v 1.16 2003/07/24 18:23:51 beevek Exp

5.14.2 Define Documentation

5.14.2.1 `#define` `ROBOT_BYTE_ORDER` `_BIG_ENDIAN`

powerpc

5.14.3 Typedef Documentation

5.14.3.1 typedef uint8_t bump_bitfield_val_t

5.14.3.2 typedef uint8_t bump_val_t

5.14.3.3 typedef int16_t encoder_val_t

5.14.3.4 typedef float freq_val_t

to ctl sonar/ir

5.14.3.5 typedef uint8_t hw_freq_val_t

freq sent to hw

5.14.3.6 typedef float ir_val_t

5.14.3.7 typedef uint16_t ir_voltage_val_t

5.14.3.8 typedef float odom_val_t

5.14.3.9 typedef pid_t owner_val_t

5.14.3.10 typedef int8_t pwm_val_t

5.14.3.11 typedef uint32_t robot_id_t

5.14.3.12 typedef uint64_t robot_time_us_t

5.14.3.13 typedef uint16_t sonar_time_val_t

5.14.3.14 typedef float sonar_val_t

5.14.3.15 typedef float vel_val_t

5.15 robot/util.h File Reference

Miscellaneous utilities useful both internally to librobot and to external applications using it.

```
#include <robot/config.h>
#include <robot/types.h>
#include <math.h>
```

Defines

- #define **robot_deg2rad**(a) ((a) * M_PI / 180)
- #define **robot_rad2deg**(a) ((a) * 180 / M_PI)
- #define **robot_dprintf**(s...)
- #define **assert**(a)
- #define **robot_fix_byte_order**(buf, item_sz, n)

Functions

- **odom_val_t robot_sqdist** (**odom_val_t** x1, **odom_val_t** y1, **odom_val_t** x2, **odom_val_t** y2)
Calculate squared distance between two points.
- **odom_val_t robot_dist** (**odom_val_t** x1, **odom_val_t** y1, **odom_val_t** x2, **odom_val_t** y2)
Calculate distance between two points.

5.15.1 Detailed Description

Miscellaneous utilities useful both internally to librobot and to external applications using it.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

util.h,v 1.9 2003/07/18 18:41:58 beevek Exp

5.15.2 Define Documentation

5.15.2.1 #define **assert**(a)

5.15.2.2 #define **robot_deg2rad**(a) ((a) * M_PI / 180)

degrees to radians

5.15.2.3 `#define robot_dprintf(s...)`

5.15.2.4 `#define robot_fix_byte_order(buf, item_sz, n)`

5.15.2.5 `#define robot_rad2deg(a) ((a) * 180 / M_PI)`

radians to degrees

5.15.3 Function Documentation

5.15.3.1 `odom_val_t robot_dist (odom_val_t x1, odom_val_t y1, odom_val_t x2, odom_val_t y2)`

Calculate distance between two points.

5.15.3.2 `odom_val_t robot_sqdist (odom_val_t x1, odom_val_t y1, odom_val_t x2, odom_val_t y2)`

Calculate squared distance between two points.

Chapter 6

librobot-api Page Documentation

6.1 Compilation Instructions

```
Robot source compilation/installation howto
Kris Beevers (beevek@cs.rpi.edu)
$Id: compiling.txt,v 1.5 2003/08/01 18:17:31 beevek Exp $
```

Compilation of the software for the robot is managed by a combination of global makefiles and local ones for each software module.

The simple method for compiling all of the robot code is:

Robot control code (everything but PDA, simulator)

```
$ cd <robot source dir>
$ ./configure --prefix=<installdir>
$ make
$ make install
```

this will compile all of the robot code and install it in a directory tree whose root is <installdir> (/usr/local by default). additionally, a SysV init script will be put at /etc/init.d/robot and linked to in /etc/rc{234}.d. This script will load the robot driver and start several services. Edit options within the script to change what/how different services are run.

For debugging code to be compiled in, you must instead do

```
$ ./configure --enable-debug
```

mostly, this will just print out lots of debugging information to stdout. there are a few minor other things that it does; to see them, do 'grep -r ROBOT_DEBUG *' in the source tree.

On-PDA code

```
$ cd <robot source dir>
$ ./configure --enable-pda
$ make
```

You'll have to install this yourself for now, scripts might be provided eventually.

Simulator

```
$ cd <robot source dir>
```

```
$ ./configure --enable-sim
$ make
$ make install (FIXME not yet working)
```

Doxygen Documentation

```
$ cd <robot source dir>
$ ./configure --enable-doc
$ make
```

Now all of the documentation is in the doc/ directory; this generates both documentation for the entire tree, and a smaller set of documentation just for the librobot api.

Other configure options

Running configure with the --enable-librobot option builds only librobot itself and the tests in misc/tests. This is particularly useful for programs that are running somewhere other than the robot. Running configure with --enable-init-script=no (or just --disable-init-script) will cause the SysV init script for the robot not to be installed during a make install.

All of the modules can be compiled individually as well. For example,

```
$ cd <robot source dir>
$ ./configure
$ cd librobot
$ make
```

will build the librobot module. However, in order for any of the modules to compile, configure must be run first, in order to create the Rules.make file.

Most modules provide an uninstall target as well. Just run make uninstall in the source tree's root directory to uninstall everything recursively.

All new Makefiles should

- a) set the TOPDIR variable based on their relative paths
- b) include \$(TOPDIR)/Rules.make

See some of the already-written makefiles for examples of doing this.

6.2 Librobot Extra Documentation

Kris Beevers (beevek@cs.rpi.edu)

API documentation for librobot.

\$Id: librobot_api.txt,v 1.2 2003/08/01 18:17:31 beevek Exp \$

The librobot API is extensively documented via doxygen. To compile the doxygen documentation, you must

- * have doxygen
- * have latex if you want ps/pdf documentation (not just html)

Simply run configure with the --enable-doc option, and then make. I.e., in the source tree root directory, do:

```
$ ./configure --enable-doc
$ make
```

Now, librobot documentation is available in doc/librobot-api (html) and doc/librobot-api.{ps,pdf}.

In order to use librobot, programs need to link with it. Ideally, all robot programs should include the robot's Rules.make file and use its definitions in their own Makefile. To do so, in their Makefiles they should:

```
include $(ROBOT_INSTALL_DIR)/make/Rules.make
LDFLAGS += $(LIBROBOT_FLAGS)
LDLIBS += $(LIBROBOT)
```

This will ensure that programs use the proper libraries and flags for compiling with librobot.

In general, most programs should just use

```
#include <robot.h>
```

rather than including individual header files.

In general, librobot is not thread safe. In particular, the sequencer functionality (seq.h) makes use of static variables. It is not recommended that threaded programs make use of librobot.

6.3 General Robot Software Specifications

Kris Beevers (beevek@cs.rpi.edu)

General software specification document for the robot software, from the lowest to highest levels.

\$Id: soft_specs.txt,v 1.3 2003/08/01 18:17:31 beevek Exp \$

This document has largely been turned into an index for more focused documents that talk about different components in the robot's software system.

All of the robot's software that runs on its computer is available from the CVS repository on the CS filesystem, at /projects/ar1/CVS. Almost all of the software will compile on any Linux machine, but most of it is meant to be compiled and used on the robot itself. For information on properly building and installing the robot software, see doc/compiling.txt.

The robot's core software consists of a number of components; ordered from lowest "level" (with respect to the hardware), they are:

```
* interp      (interpreter daemon)
* mc_*        (motor controllers)
* robot.o     (/dev/robot driver)
* librobot    (software interface for user-level programs)
* netrobotd   (daemon for controlling robot over network)
* logger      (daemon for logging sensor/etc data)
* sequencer   (manages reactive layer/behaviors)
* libbehavior (used in the creation of reactive behaviors)
```

There are also a number of "peripheral" tools/programs, as follows:

```
* nrtool      (console interface for controlling robot over the
              network)
* logtool     (examine log files produced by logger)
* bhvctl      (cmdline tool for talking to sequencer)
* misc/tests/* (a number of example/test programs for librobot)
* simulator   (simulator providing a librobot api)
* mom         (pda interface to robot)
* robot.sh    (sysv init script to start core services)
```

Detailed information on the interpreter, motor controllers and robot.o is in low_level_data_flow.txt and motor_control_interface.txt.

Information on librobot and netrobotd is in librobot_api.txt.

Information on data logging provided by logger is in data_logging.txt.

Information on the sequencer is in sequencer.txt. Information on libbehavior is in libbehavior_api.txt.

Note that most of the programs (e.g. interp, netrobotd, logger, sequencer, nrtool, logtool, bhvctl) take command line options. For usage information, run them with --help.

Here is an (extremely) high-level overview of "how the system works":

Most important hardware in the system, such as motors and sensors, is controlled by a series of microcontrollers. These controllers speak (currently through a serial interface) to the main CPU, where all of the above software resides. The interpreter reads and writes from the serial device and is the sole channel for talking to the hardware. Motor control is performed inside the interpreter, via shared motor control libraries. All sensor data from the hardware is forwarded to the /dev/robot interface provided by robot.o. In addition, the interpreter reads from "ctl"-type device files and sends commands to the hardware based on the data.

The librobot library provides a simple, clean interface and for the most part just reads from and writes to the /dev/robot files

(communicating with the interpreter). However, it also transparently provides a network-based (tcp/udp) version of /dev/robot for talking to a netrobotd remotely. All netrobotd really does is forward data between the network and a local /dev/robot, and vice versa. So, a program compiled with librobot can be run either locally on the robot itself, or remotely on another computer (as long as the robot is accessible via the network, and is running netrobotd).

Worthy of note is that we have dropped the RTOS requirement that we originally had for the system. No RTOS is installed. Generally, interp is run as a high-priority process and so far this has been more than sufficient to ensure adequate control over the robot's hardware.

On top of librobot, normal user programs can run and control the robot. In addition, we provide the functionality of a "reactive" layer, in which a number of (generally small) "behaviors" run concurrently, interacting with each other and the hardware as appropriate. This is done via the "sequencer," which is basically just a launching and control point for behavior programs. In our system, every behavior is a separate process (including sub-behaviors launched by a higher-level behavior).

Eventually bindings for librobot might be provided in several languages (other than C/C++); Perl, Python, Lisp and Scheme are candidates.

And now some notes about what still needs to be done in terms of software in the system.

Low Level

- * USB; the LCD panel, on-robot UI switches, and all proprioceptive sensors (power status, internal temperature, and tilt?) will be interfaced through a USB port. the sensors and UI switches will definitely go through a microcontroller; possibly the LCD will too. in all probability, interp will be updated to talk to these devices as well.
- * Firewire/Camera; drivers and support for extracting images must be provided. it is questionable whether interp should handle this or whether it should instead be part of librobot (if it is part of librobot, image data will not be available from /dev/robot/camera, though this is probably ok).
- * Wireless ethernet; drivers for a PCMCIA ethernet card need to be installed.
- * Complete testing of PID motor controller.
- * Implementation of other motor controllers.
- * Communication "service" for inter-robot communication.

Reactive Layer

Here are a few of the behaviors we may want to implement; we currently have no real behaviors, just the facilities for implementing them:

- * Emergency stop: stop on toggling of bump sensors (any other criteria?)
- * Preemptive stop: stop when infrared sensors return some minimum value
- * Move-to: move to a specified configuration while avoiding obstacles (e.g. using a bug algorithm)
- * Wall-following: find a wall and then follow it
- * Exploration: wander according to some rules (perhaps based on data from a mapping/localization service?)

- * Follow: follow a person (probably based on vision) or another robot
- * Flee: avoid any approaching object
- * Meet: approach other robots (or people?) to within some threshold distance (perhaps while talking to the other robots?)
- * What else?

Deliberative Components

FIXME (asynchronous planning, should be able to change the goals being worked on by the sequencer at any time, i.e. interrupting whatever is currently being worked on. pass series of 'actions' to the sequencer? in this case, are what we have in the reactive layer right now the 'actions' and behaviors should be lower level?)

Other Services

FIXME (e.g. communication; data requests/responses, what else?, want to use 802.11b in adhoc mode, should handle multihop routing for us)

6.4 Todo List

File constants.h Most of these values need to be appropriately set.

File motors.h FIXME: acceleration?

Index

- assert
 - util.h, 70
- behavior.h
 - bhv_add_input, 23
 - bhv_block_forever, 23
 - bhv_cleanup_func_t, 22
 - bhv_data_func_t, 22
 - bhv_flags_t, 23
 - bhv_inhibit_func_t, 22
 - bhv_init, 24
 - bhv_input_pop, 24
 - bhv_main_func_t, 22
 - bhv_name, 25
 - bhv_output, 24
 - bhv_perror, 21
 - bhv_printf, 21
 - bhv_self, 25
 - bhv_shutdown, 25
 - bhv_uninhibit_func_t, 23
 - NO_ROBOT_INIT, 23
 - QUEUE_WHEN_INHIBITED, 23
- bhv_add_input
 - behavior.h, 23
- bhv_block_forever
 - behavior.h, 23
- bhv_cleanup_func_t
 - behavior.h, 22
- BHV_CONNECT
 - seq.h, 58
- bhv_connection_t, 7
 - from, 7
 - from_key, 7
 - to, 7
 - to_key, 7
- BHV_DATA
 - seq.h, 58
- bhv_data_func_t
 - behavior.h, 22
- bhv_data_t, 8
 - data, 8
 - key, 8
- BHV_DISCONNECT
 - seq.h, 58
- bhv_flags_t
 - behavior.h, 23
- bhv_handle_t, 9
 - pid, 9
 - qid, 9
- BHV_INHIBIT
 - seq.h, 58
- bhv_inhibit_func_t
 - behavior.h, 22
- BHV_INIT
 - seq.h, 58
- bhv_init
 - behavior.h, 24
- bhv_input_pop
 - behavior.h, 24
- bhv_main_func_t
 - behavior.h, 22
- bhv_name
 - behavior.h, 25
- bhv_output
 - behavior.h, 24
- bhv_perror
 - behavior.h, 21
- bhv_printf
 - behavior.h, 21
- bhv_self
 - behavior.h, 25
- bhv_shutdown
 - behavior.h, 25
- bhv_t, 10
 - cleanup, 10
 - flags, 10
 - main, 10
 - on_inhibit, 10
 - on_uninhibit, 10
- BHV_UNINHIBIT
 - seq.h, 58
- bhv_uninhibit_func_t
 - behavior.h, 23
- buf
 - robot_net_msg_t, 16
- bump_bitfield_val_t
 - types.h, 69
- bump_val_t
 - types.h, 69

- change
 - devfs_set_t, 11
- cleanup
 - bhv_t, 10
- cmd
 - seq_msgbuf_t, 17
- constants.h
 - ROBOT_AXLE_WIDTH, 27
 - ROBOT_DEFAULT_IR_FREQ, 27
 - ROBOT_DEFAULT_NET_PORT, 27
 - ROBOT_DEFAULT_NET_TIMEOUT, 27
 - ROBOT_DEFAULT_SONAR_FREQ, 27
 - ROBOT_DEFAULT_VEL_V, 27
 - ROBOT_DEFAULT_VEL_W, 27
 - ROBOT_ENCODER_STEPS_PER_REV, 27
 - ROBOT_HW_TTY, 27
 - ROBOT_IR_MULTIPLIER, 27
 - ROBOT_MAX_NAME_LEN, 27
 - ROBOT_MAX_VEL_V, 27
 - ROBOT_MAX_VEL_W, 27
 - ROBOT_NUM_BUMP, 27
 - ROBOT_NUM_IR, 27
 - ROBOT_NUM_SONAR, 27
 - ROBOT_SONAR_MULTIPLIER, 28
 - ROBOT_THRESH_IR, 28
 - ROBOT_THRESH_ODOM_THETA, 28
 - ROBOT_THRESH_ODOM_X, 28
 - ROBOT_THRESH_ODOM_Y, 28
 - ROBOT_THRESH_ROTATE, 28
 - ROBOT_THRESH_SONAR, 28
 - ROBOT_THRESH_TRANSLATE, 28
 - ROBOT_THRESH_VEL_V, 28
 - ROBOT_THRESH_VEL_W, 28
 - ROBOT_WHEEL_RADIUS, 28
 - ROBOT_WHEEL_RADIUS_L, 28
 - ROBOT_WHEEL_RADIUS_R, 28
- count
 - robot_net_msg_t, 16
- ctl
 - devfs_set_t, 11
- CTL_SIZE_BUMP
 - devfs.h, 32
- CTL_SIZE_ENCODER
 - devfs.h, 32
- CTL_SIZE_IR
 - devfs.h, 32
- CTL_SIZE_ODOM
 - devfs.h, 32
- CTL_SIZE_PWM
 - devfs.h, 32
- CTL_SIZE_SONAR
 - devfs.h, 32
- CTL_SIZE_VEL
 - devfs.h, 32
- cur_robot
 - handle.h, 39
- current
 - devfs_set_t, 11
- data
 - bhv_data_t, 8
 - seq_msgbuf_t, 17
- DATA_SIZE_BUMP
 - devfs.h, 32
- DATA_SIZE_ENCODER
 - devfs.h, 32
- DATA_SIZE_IR
 - devfs.h, 32
- DATA_SIZE_ODOM
 - devfs.h, 32
- DATA_SIZE_PWM
 - devfs.h, 32
- DATA_SIZE_SONAR
 - devfs.h, 32
- DATA_SIZE_VEL
 - devfs.h, 32
- DEV_TYPE_CHANGE
 - devfs.h, 34
- DEV_TYPE_CHANGE_STR
 - devfs.h, 32
- DEV_TYPE_CTL
 - devfs.h, 34
- DEV_TYPE_CTL_STR
 - devfs.h, 32
- DEV_TYPE_CURRENT
 - devfs.h, 34
- DEV_TYPE_CURRENT_STR
 - devfs.h, 32
- DEV_TYPE_STREAM
 - devfs.h, 34
- DEV_TYPE_STREAM_STR
 - devfs.h, 32
- devfs.h
 - CTL_SIZE_BUMP, 32
 - CTL_SIZE_ENCODER, 32
 - CTL_SIZE_IR, 32
 - CTL_SIZE_ODOM, 32
 - CTL_SIZE_PWM, 32
 - CTL_SIZE_SONAR, 32
 - CTL_SIZE_VEL, 32
 - DATA_SIZE_BUMP, 32
 - DATA_SIZE_ENCODER, 32
 - DATA_SIZE_IR, 32
 - DATA_SIZE_ODOM, 32
 - DATA_SIZE_PWM, 32
 - DATA_SIZE_SONAR, 32

- DATA_SIZE_VEL, 32
- DEV_TYPE_CHANGE, 34
- DEV_TYPE_CHANGE_STR, 32
- DEV_TYPE_CTL, 34
- DEV_TYPE_CTL_STR, 32
- DEV_TYPE_CURRENT, 34
- DEV_TYPE_CURRENT_STR, 32
- DEV_TYPE_STREAM, 34
- DEV_TYPE_STREAM_STR, 32
- DEVFS_BUMP, 34
- DEVFS_BUMP_SINGLE, 34
- DEVFS_BUMP_STR, 32
- devfs_cleanup, 34
- DEVFS_CLIENT, 33
- devfs_dir_type_t, 33
- DEVFS_ENCODER, 34
- DEVFS_ENCODER_STR, 32
- devfs_fds, 14
- robot_handle_t, 14
- devfs_find, 34
- devfs_flags, 14
- robot_handle_t, 14
- devfs_get_data_size, 34
- devfs_get_lock_owner, 35
- devfs_init, 35
- DEVFS_IOCTLGETLOCKOWNER, 32
- DEVFS_IOCTLLOCKCTL, 32
- DEVFS_IOCTLUNLOCKCTL, 32
- DEVFS_IR, 34
- DEVFS_IR_SINGLE, 34
- DEVFS_IR_STR, 33
- devfs_lock_ctl, 35
- DEVFS_ODOM, 34
- DEVFS_ODOM_STR, 33
- DEVFS_PRIO_CRITICAL, 33
- DEVFS_PRIO_HIGH, 33
- DEVFS_PRIO_LOW, 33
- DEVFS_PRIO_NORMAL, 33
- DEVFS_PRIO_SUPER, 33
- DEVFS_PWM, 34
- DEVFS_PWM_STR, 33
- DEVFS_RDONLY, 33
- devfs_read, 35
- DEVFS_ROOT_STR, 33
- DEVFS_SERVER, 33
- DEVFS_SONAR, 34
- DEVFS_SONAR_SINGLE, 34
- DEVFS_SONAR_STR, 33
- devfs_type_t, 34
- devfs_unlock_ctl, 36
- DEVFS_VEL, 34
- DEVFS_VEL_STR, 33
- devfs_wait_for_change, 36
- devfs_write, 36
- NUM_DEVICE_DIRS, 33
- NUM_DEVICES, 33
- DEVFS_BUMP, 34
- devfs.h, 34
- DEVFS_BUMP_SINGLE, 34
- devfs.h, 34
- DEVFS_BUMP_STR, 32
- devfs.h, 32
- devfs_cleanup, 34
- devfs.h, 34
- DEVFS_CLIENT, 33
- devfs.h, 33
- devfs_dir_type_t, 33
- devfs.h, 33
- DEVFS_ENCODER, 34
- devfs.h, 34
- DEVFS_ENCODER_STR, 32
- devfs.h, 32
- devfs_fds, 14
- robot_handle_t, 14
- devfs_find, 34
- devfs.h, 34
- devfs_flags, 14
- robot_handle_t, 14
- devfs_get_data_size, 34
- devfs.h, 34
- devfs_get_lock_owner, 35
- devfs.h, 35
- devfs_init, 35
- devfs.h, 35
- DEVFS_IOCTLGETLOCKOWNER, 32
- devfs.h, 32
- DEVFS_IOCTLLOCKCTL, 32
- devfs.h, 32
- DEVFS_IOCTLUNLOCKCTL, 32
- devfs.h, 32
- DEVFS_IR, 34
- devfs.h, 34
- DEVFS_IR_SINGLE, 34
- devfs.h, 34
- DEVFS_IR_STR, 33
- devfs.h, 33
- devfs_lock_ctl, 35
- devfs.h, 35
- DEVFS_ODOM, 34
- devfs.h, 34
- DEVFS_ODOM_STR, 33
- devfs.h, 33
- DEVFS_PRIO_CRITICAL, 33
- devfs.h, 33
- DEVFS_PRIO_HIGH, 33
- devfs.h, 33
- DEVFS_PRIO_LOW, 33
- devfs.h, 33
- DEVFS_PRIO_NORMAL, 33
- devfs.h, 33
- DEVFS_PRIO_SUPER, 33
- devfs.h, 33

- DEVFS_PWM
 - devfs.h, 34
- DEVFS_PWM_STR
 - devfs.h, 33
- DEVFS_RDONLY
 - devfs.h, 33
- devfs_read
 - devfs.h, 35
- DEVFS_ROOT_STR
 - devfs.h, 33
- DEVFS_SERVER
 - devfs.h, 33
- devfs_set_t, 11
 - change, 11
 - ctl, 11
 - current, 11
 - hwid, 11
 - stream, 11
- DEVFS_SONAR
 - devfs.h, 34
- DEVFS_SONAR_SINGLE
 - devfs.h, 34
- DEVFS_SONAR_STR
 - devfs.h, 33
- devfs_type
 - robot_net_msg_t, 16
- devfs_type_t
 - devfs.h, 34
- devfs_unlock_ctl
 - devfs.h, 36
- DEVFS_VEL
 - devfs.h, 34
- DEVFS_VEL_STR
 - devfs.h, 33
- devfs_wait_for_change
 - devfs.h, 36
- devfs_write
 - devfs.h, 36
- encoder_val_t
 - types.h, 69
- flags
 - bhv_t, 10
 - robot_handle_t, 14
- freq
 - freq_req_val_t, 12
- freq_req_val_t, 12
 - freq, 12
 - owner, 12
- freq_val_t
 - types.h, 69
- from
 - bhv_connection_t, 7
 - from_key
 - bhv_connection_t, 7
- handle
 - mc_lib_t, 13
- handle.h
 - cur_robot, 39
 - handle_is_connected, 38
 - handle_is_loc, 38
 - handle_is_net, 39
 - robot_set_handle, 39
- handle_is_connected
 - handle.h, 38
- handle_is_loc
 - handle.h, 38
- handle_is_net
 - handle.h, 39
- HW_ALL_BUMP
 - hwid.h, 42
- HW_ALL_IR
 - hwid.h, 42
- HW_ALL_SONAR
 - hwid.h, 42
- HW_BUMP01
 - hwid.h, 43
- HW_BUMP02
 - hwid.h, 43
- HW_BUMP03
 - hwid.h, 43
- HW_BUMP04
 - hwid.h, 43
- HW_BUMP05
 - hwid.h, 43
- HW_BUMP06
 - hwid.h, 43
- HW_BUMP07
 - hwid.h, 43
- HW_BUMP08
 - hwid.h, 43
- HW_ENCODER
 - hwid.h, 42
- hw_freq_val_t
 - types.h, 69
- HW_IR01
 - hwid.h, 42
- HW_IR02
 - hwid.h, 42
- HW_IR03
 - hwid.h, 42
- HW_IR04
 - hwid.h, 42
- HW_IR05
 - hwid.h, 43
- HW_IR06
 - hwid.h, 43

hwid.h, 43
 HW_IR07
 hwid.h, 43
 HW_IR08
 hwid.h, 43
 HW_IR09
 hwid.h, 43
 HW_IR10
 hwid.h, 43
 HW_IR11
 hwid.h, 43
 HW_IR12
 hwid.h, 43
 HW_IR13
 hwid.h, 43
 HW_IR14
 hwid.h, 43
 HW_IR15
 hwid.h, 43
 HW_IR16
 hwid.h, 43
 HW_IS_ALL_BUMP
 hwid.h, 41
 HW_IS_ALL_IR
 hwid.h, 41
 HW_IS_ALL_SONAR
 hwid.h, 41
 HW_IS_BUMP
 hwid.h, 41
 HW_IS_ENCODER
 hwid.h, 41
 HW_IS_IR
 hwid.h, 41
 HW_IS_MC
 hwid.h, 41
 HW_IS_MOTORS
 hwid.h, 41
 HW_IS_ODOM
 hwid.h, 41
 HW_IS_PWM
 hwid.h, 41
 HW_IS_SONAR
 hwid.h, 41
 HW_IS_VEL
 hwid.h, 41
 HW_IS_VIRTUAL
 hwid.h, 41
 HW_MAX
 hwid.h, 43
 HW_MC
 hwid.h, 42
 HW_MIN
 hwid.h, 43
 HW_MOTORS
 hwid.h, 42
 HW_ODOM
 hwid.h, 42
 HW_PWM
 hwid.h, 42
 HW_SONAR01
 hwid.h, 42
 HW_SONAR02
 hwid.h, 42
 HW_SONAR03
 hwid.h, 42
 HW_SONAR04
 hwid.h, 42
 HW_SONAR05
 hwid.h, 42
 HW_SONAR06
 hwid.h, 42
 HW_SONAR07
 hwid.h, 42
 HW_SONAR08
 hwid.h, 42
 HW_SONAR09
 hwid.h, 42
 HW_SONAR10
 hwid.h, 42
 HW_SONAR11
 hwid.h, 42
 HW_SONAR12
 hwid.h, 42
 HW_SONAR13
 hwid.h, 42
 HW_SONAR14
 hwid.h, 42
 HW_SONAR15
 hwid.h, 42
 HW_SONAR16
 hwid.h, 42
 HW_VEL
 hwid.h, 42
 hwid
 devfs_set_t, 11
 robot_net_msg_t, 16
 hwid.h
 HW_ALL_BUMP, 42
 HW_ALL_IR, 42
 HW_ALL_SONAR, 42
 HW_BUMP01, 43
 HW_BUMP02, 43
 HW_BUMP03, 43
 HW_BUMP04, 43
 HW_BUMP05, 43
 HW_BUMP06, 43
 HW_BUMP07, 43
 HW_BUMP08, 43

- HW_ENCODER, 42
- HW_IR01, 42
- HW_IR02, 42
- HW_IR03, 42
- HW_IR04, 42
- HW_IR05, 43
- HW_IR06, 43
- HW_IR07, 43
- HW_IR08, 43
- HW_IR09, 43
- HW_IR10, 43
- HW_IR11, 43
- HW_IR12, 43
- HW_IR13, 43
- HW_IR14, 43
- HW_IR15, 43
- HW_IR16, 43
- HW_IS_ALL_BUMP, 41
- HW_IS_ALL_IR, 41
- HW_IS_ALL_SONAR, 41
- HW_IS_BUMP, 41
- HW_IS_ENCODER, 41
- HW_IS_IR, 41
- HW_IS_MC, 41
- HW_IS_MOTORS, 41
- HW_IS_ODOM, 41
- HW_IS_PWM, 41
- HW_IS_SONAR, 41
- HW_IS_VEL, 41
- HW_IS_VIRTUAL, 41
- HW_MAX, 43
- HW_MC, 42
- HW_MIN, 43
- HW_MOTORS, 42
- HW_ODOM, 42
- HW_PWM, 42
- HW_SONAR01, 42
- HW_SONAR02, 42
- HW_SONAR03, 42
- HW_SONAR04, 42
- HW_SONAR05, 42
- HW_SONAR06, 42
- HW_SONAR07, 42
- HW_SONAR08, 42
- HW_SONAR09, 42
- HW_SONAR10, 42
- HW_SONAR11, 42
- HW_SONAR12, 42
- HW_SONAR13, 42
- HW_SONAR14, 42
- HW_SONAR15, 42
- HW_SONAR16, 42
- HW_VEL, 42
- id
 - robot_handle_t, 14
- init_complete
 - robot_handle_t, 14
- ir_val_t
 - types.h, 69
- ir_voltage_val_t
 - types.h, 69
- key
 - bhv_data_t, 8
- main
 - bhv_t, 10
- mc_do_control
 - mc.lib_t, 13
- mc_do_control_func_t
 - mclib.h, 45
- mc_get_velocity
 - mc.lib_t, 13
- mc_get_velocity_func_t
 - mclib.h, 45
- mc_init
 - mc.lib_t, 13
- mc_init_func_t
 - mclib.h, 45
- mc_lib_load
 - mclib.h, 45
- mc_lib_t, 13
 - handle, 13
 - mc_do_control, 13
 - mc_get_velocity, 13
 - mc_init, 13
 - mc_set_odometry, 13
 - mc_set_velocity, 13
 - mc_shutdown, 13
 - mc_start_frame, 13
- mc_lib_unload
 - mclib.h, 45
- mc_set_odometry
 - mc.lib_t, 13
- mc_set_odometry_func_t
 - mclib.h, 45
- mc_set_velocity
 - mc.lib_t, 13
- mc_set_velocity_func_t
 - mclib.h, 45
- mc_shutdown
 - mc.lib_t, 13
- mc_shutdown_func_t
 - mclib.h, 45
- mc_start_frame
 - mc.lib_t, 13
- mc_start_frame_func_t

- mclib.h, 45
- mclib.h
 - mc_do_control_func_t, 45
 - mc_get_velocity_func_t, 45
 - mc_init_func_t, 45
 - mc_lib_load, 45
 - mc_lib_unload, 45
 - mc_set_odometry_func_t, 45
 - mc_set_velocity_func_t, 45
 - mc_shutdown_func_t, 45
 - mc_start_frame_func_t, 45
- mid
 - robot_net_msg_t, 16
- motors.h
 - robot_get_odometry, 47
 - robot_get_velocity, 47
 - robot_lock_motors, 47
 - robot_rotate, 47
 - robot_set_odometry, 47
 - robot_set_velocity, 48
 - robot_translate, 48
 - robot_unlock_motors, 48
- MSEC_PER_SEC
 - time.h, 66
- MSG_ERROR
 - net.h, 50
- MSG_GET_LOCK_OWNER
 - net.h, 50
- MSG_INIT
 - net.h, 50
- MSG_LOCK_CTL
 - net.h, 50
- MSG_PING
 - net.h, 50
- MSG_SHUTDOWN
 - net.h, 50
- MSG_UNLOCK_CTL
 - net.h, 50
- MSG_WAIT_FOR_CHANGE
 - net.h, 50
- MSGBUF_BASE_SZ
 - seq.h, 57
- mtype
 - seq_msgbuf_t, 17
- name
 - robot_handle_t, 14
- net.h
 - MSG_ERROR, 50
 - MSG_GET_LOCK_OWNER, 50
 - MSG_INIT, 50
 - MSG_LOCK_CTL, 50
 - MSG_PING, 50
 - MSG_SHUTDOWN, 50
 - MSG_UNLOCK_CTL, 50
 - MSG_WAIT_FOR_CHANGE, 50
 - robot_get_ip_str, 50
 - robot_net_find, 50
 - robot_net_set_timeout, 50
- NO_ROBOT_INIT
 - behavior.h, 23
- NUM_DEVICE_DIRS
 - devfs.h, 33
- NUM_DEVICES
 - devfs.h, 33
- odom_val_t
 - types.h, 69
- on_inhibit
 - bhv_t, 10
- on_uninhibit
 - bhv_t, 10
- owner
 - freq_req_val_t, 12
- owner_val_t
 - types.h, 69
- pid
 - bhv_handle_t, 9
- pwm_val_t
 - types.h, 69
- qid
 - bhv_handle_t, 9
- QUEUE_WHEN_INHIBITED
 - behavior.h, 23
- RLDEFAULT
 - sys.h, 64
- RLDEVFS_READ_ALL
 - sys.h, 64
- RLNO_DEVFS
 - sys.h, 64
- RLNO_HANDLE_SIGS
 - sys.h, 64
- RLNO_SET_FREQS
 - sys.h, 64
- RLSTOP_ON_QUIT
 - sys.h, 64
- RLUSE_SEQUENCER
 - sys.h, 64
- robot.h, 19
- robot/behavior.h, 20
- robot/constants.h, 26
- robot/devfs.h, 29
- robot/handle.h, 38
- robot/hwid.h, 40
- robot/mclib.h, 44

- robot/motors.h, 46
- robot/net.h, 49
- robot/sensors.h, 52
- robot/seq.h, 56
- robot/sys.h, 63
- robot/time.h, 66
- robot/types.h, 68
- robot/util.h, 70
- robot_alarm
 - time.h, 67
- ROBOT_AXLE_WIDTH
 - constants.h, 27
- ROBOT_BYTE_ORDER
 - types.h, 68
- ROBOT_DEFAULT_IR_FREQ
 - constants.h, 27
- ROBOT_DEFAULT_NET_PORT
 - constants.h, 27
- ROBOT_DEFAULT_NET_TIMEOUT
 - constants.h, 27
- ROBOT_DEFAULT_SONAR_FREQ
 - constants.h, 27
- ROBOT_DEFAULT_VEL_V
 - constants.h, 27
- ROBOT_DEFAULT_VEL_W
 - constants.h, 27
- robot_deg2rad
 - util.h, 70
- robot_dist
 - util.h, 71
- robot_dprintf
 - util.h, 70
- ROBOT_ENCODER_STEPS_PER_REV
 - constants.h, 27
- robot_fix_byte_order
 - util.h, 71
- robot_force_reset_all_sensors
 - sensors.h, 53
- robot_get_all_bump
 - sensors.h, 53
- robot_get_all_ir
 - sensors.h, 53
- robot_get_all_sonar
 - sensors.h, 53
- robot_get_bump
 - sensors.h, 54
- robot_get_id
 - sys.h, 64
- robot_get_ip_str
 - net.h, 50
- robot_get_ir
 - sensors.h, 54
- robot_get_name
 - sys.h, 64
- robot_get_odometry
 - motors.h, 47
- robot_get_sonar
 - sensors.h, 54
- robot_get_time
 - time.h, 67
- robot_get_velocity
 - motors.h, 47
- robot_handle_t, 14
 - devfs_fds, 14
 - devfs_flags, 14
 - flags, 14
 - id, 14
 - init_complete, 14
 - name, 14
 - sock, 15
 - sockaddr, 15
 - timeout, 15
- ROBOT_HW_TTY
 - constants.h, 27
- robot_id_t
 - types.h, 69
- robot_init
 - sys.h, 64
- robot_init_flags_t
 - sys.h, 64
- robot_init_local
 - sys.h, 64
- ROBOT_IR_MULTIPLIER
 - constants.h, 27
- robot_lock_motors
 - motors.h, 47
- ROBOT_MAX_BHV_DATA_SIZE
 - seq.h, 57
- ROBOT_MAX_NAME_LEN
 - constants.h, 27
- ROBOT_MAX_VEL_V
 - constants.h, 27
- ROBOT_MAX_VEL_W
 - constants.h, 27
- robot_net_find
 - net.h, 50
- robot_net_msg_t, 16
 - buf, 16
 - count, 16
 - devfs_type, 16
 - hwid, 16
 - mid, 16
- robot_net_set_timeout
 - net.h, 50
- ROBOT_NUM_BUMP
 - constants.h, 27
- ROBOT_NUM_IR
 - constants.h, 27

- ROBOT_NUM_SONAR
 - constants.h, 27
- robot_rad2deg
 - util.h, 71
- robot_rotate
 - motors.h, 47
- ROBOT_SEQ_QUEUE_KEY
 - seq.h, 57
- robot_set_all_ir_freq
 - sensors.h, 54
- robot_set_all_sonar_freq
 - sensors.h, 54
- robot_set_handle
 - handle.h, 39
- robot_set_ir_freq
 - sensors.h, 55
- robot_set_odometry
 - motors.h, 47
- robot_set_sensor_freq
 - sensors.h, 55
- robot_set_sonar_freq
 - sensors.h, 55
- robot_set_velocity
 - motors.h, 48
- robot_shutdown
 - sys.h, 65
- robot_sleep
 - time.h, 67
- ROBOT_SONAR_MULTIPLIER
 - constants.h, 28
- robot_sqdist
 - util.h, 71
- ROBOT_THRESH_IR
 - constants.h, 28
- ROBOT_THRESH_ODOM_THETA
 - constants.h, 28
- ROBOT_THRESH_ODOM_X
 - constants.h, 28
- ROBOT_THRESH_ODOM_Y
 - constants.h, 28
- ROBOT_THRESH_ROTATE
 - constants.h, 28
- ROBOT_THRESH_SONAR
 - constants.h, 28
- ROBOT_THRESH_TRANSLATE
 - constants.h, 28
- ROBOT_THRESH_VEL_V
 - constants.h, 28
- ROBOT_THRESH_VEL_W
 - constants.h, 28
- robot_time_to_float_ms
 - time.h, 66
- robot_time_to_float_sec
 - time.h, 66
- robot_time_us_t
 - types.h, 69
- robot_translate
 - motors.h, 48
- robot_unlock_motors
 - motors.h, 48
- ROBOT_WHEEL_RADIUS
 - constants.h, 28
- ROBOT_WHEEL_RADIUS_L
 - constants.h, 28
- ROBOT_WHEEL_RADIUS_R
 - constants.h, 28
- sensors.h
 - robot_force_reset_all_sensors, 53
 - robot_get_all_bump, 53
 - robot_get_all_ir, 53
 - robot_get_all_sonar, 53
 - robot_get_bump, 54
 - robot_get_ir, 54
 - robot_get_sonar, 54
 - robot_set_all_ir_freq, 54
 - robot_set_all_sonar_freq, 54
 - robot_set_ir_freq, 55
 - robot_set_sensor_freq, 55
 - robot_set_sonar_freq, 55
- seq.h
 - BHV_CONNECT, 58
 - BHV_DATA, 58
 - BHV_DISCONNECT, 58
 - BHV_INHIBIT, 58
 - BHV_INIT, 58
 - BHV_UNINHIBIT, 58
 - MSGBUF_BASE_SZ, 57
 - ROBOT_MAX_BHV_DATA_SIZE, 57
 - ROBOT_SEQ_QUEUE_KEY, 57
 - SEQ_ATTACH, 58
 - seq_attach, 58
 - seq_connect, 58
 - seq_disconnect, 58
 - SEQ_ERR, 58
 - seq_get, 59
 - seq_get_my_handle, 59
 - seq_inhibit, 59
 - seq_inhibit_all, 60
 - SEQ_LIST, 58
 - SEQ_LOAD, 58
 - seq_load, 60
 - seq_load_args, 60
 - seq_load_net, 61
 - seq_send, 61
 - seq_uninhibit, 61
 - seq_uninhibit_all, 61
 - SEQ_UNLOAD, 58

- seq_unload, 62
- seq_unload_all, 62
- SEQ_ATTACH
 - seq.h, 58
- seq_attach
 - seq.h, 58
- seq_connect
 - seq.h, 58
- seq_disconnect
 - seq.h, 58
- SEQ_ERR
 - seq.h, 58
- seq_get
 - seq.h, 59
- seq_get_my_handle
 - seq.h, 59
- seq_inhibit
 - seq.h, 59
- seq_inhibit_all
 - seq.h, 60
- SEQ_LIST
 - seq.h, 58
- SEQ_LOAD
 - seq.h, 58
- seq_load
 - seq.h, 60
- seq_load_args
 - seq.h, 60
- seq_load_net
 - seq.h, 61
- seq_msgbuf_t, 17
 - cmd, 17
 - data, 17
 - mtype, 17
- seq_send
 - seq.h, 61
- seq_uninhibit
 - seq.h, 61
- seq_uninhibit_all
 - seq.h, 61
- SEQ_UNLOAD
 - seq.h, 58
- seq_unload
 - seq.h, 62
- seq_unload_all
 - seq.h, 62
- sock
 - robot_handle_t, 15
- sockaddr
 - robot_handle_t, 15
- sonar_time_val_t
 - types.h, 69
- sonar_val_t
 - types.h, 69
- stream
 - devfs_set_t, 11
- sys.h
 - RLDEFAULT, 64
 - RLDEVFS_READ_ALL, 64
 - RLNO_DEVFS, 64
 - RLNO_HANDLE_SIGS, 64
 - RLNO_SET_FREQS, 64
 - RLSTOP_ON_QUIT, 64
 - RLUSE_SEQUENCER, 64
 - robot_get_id, 64
 - robot_get_name, 64
 - robot_init, 64
 - robot_init_flags_t, 64
 - robot_init_local, 64
 - robot_shutdown, 65
- time.h
 - MSEC_PER_SEC, 66
 - robot_alarm, 67
 - robot_get_time, 67
 - robot_sleep, 67
 - robot_time_to_float_ms, 66
 - robot_time_to_float_sec, 66
 - USEC_PER_MSEC, 67
 - USEC_PER_SEC, 67
- timeout
 - robot_handle_t, 15
- to
 - bhv_connection_t, 7
- to_key
 - bhv_connection_t, 7
- types.h
 - bump_bitfield_val_t, 69
 - bump_val_t, 69
 - encoder_val_t, 69
 - freq_val_t, 69
 - hw_freq_val_t, 69
 - ir_val_t, 69
 - ir_voltage_val_t, 69
 - odom_val_t, 69
 - owner_val_t, 69
 - pwm_val_t, 69
 - ROBOT_BYTE_ORDER, 68
 - robot_id_t, 69
 - robot_time_us_t, 69
 - sonar_time_val_t, 69
 - sonar_val_t, 69
 - vel_val_t, 69
- USEC_PER_MSEC
 - time.h, 67
- USEC_PER_SEC
 - time.h, 67

util.h

- assert, 70
- robot_deg2rad, 70
- robot_dist, 71
- robot_dprintf, 70
- robot_fix_byte_order, 71
- robot_rad2deg, 71
- robot_sqdist, 71

vel_val_t

- types.h, 69