robots-all Reference Manual

Generated by Doxygen 1.3.2

## Contents

1 robots-all Namespace Index ............................................. 1
   1.1 robots-all Namespace List ...................................... 1

2 robots-all Compound Index .......................................... 3
   2.1 robots-all Compound List ...................................... 3

3 robots-all File Index .................................................. 5
   3.1 robots-all File List ............................................ 5

4 robots-all Page Index .................................................. 9
   4.1 robots-all Related Pages ...................................... 9

5 robots-all Namespace Documentation ................................ 11
   5.1 dilt Namespace Reference .................................... 11
   5.2 std Namespace Reference .................................... 12

6 robots-all Class Documentation ..................................... 13
   6.1 bhv\_connection\_\_ struct Reference .......................... 13
   6.2 bhv\_data\_\_ struct Reference ................................ 15
   6.3 bhv\_handle\_\_ struct Reference ............................. 16
   6.4 bhv\_\_ struct Reference ..................................... 17
   6.5 BumpSensor Class Reference .................................. 18
   6.6 data\_len\_\_ struct Reference ................................ 19
   6.7 devfs\_set\_\_ struct Reference ................................ 20
   6.8 DiscoveryDialog Class Reference ............................ 21
   6.9 freq\_req\_val\_\_ struct Reference ............................ 22
   6.10 InfoDialog Class Reference ................................. 23
   6.11 IpInputDialog Class Reference ............................... 24
   6.12 IRDialog Class Reference ................................. 25
   6.13 log\_writer\_\_ Class Reference ............................. 26
6.14 mc_lib¶ Struct Reference ........................................... 28
6.15 MomCanvasView Class Reference ..................................... 29
6.16 MomWindow Class Reference .......................................... 30
6.17 mpProblem Class Reference .......................................... 32
6.18 Obstacle Class Reference ............................................. 33
6.19 Point Class Reference ................................................ 34
6.20 PointTheta Class Reference .......................................... 35
6.21 robot¶ Dev ¶ Struct Reference ........................................ 36
6.22 robot¶ Info ¶ Struct Reference ....................................... 38
6.23 robot¶ handle ¶ Struct Reference .................................... 39
6.24 robot¶ net¶ msg ¶ Struct Reference ................................ 41
6.25 RobotInfo Class Reference ........................................... 42
6.26 SensorInfo Class Reference .......................................... 43
6.27 seq¶ msgbuf ¶ Struct Reference .................................... 44
6.28 serial¶ cmd ¶ Struct Reference ...................................... 45
6.29 SimRobot Class Reference ........................................... 46
6.30 SonarDialog Class Reference ....................................... 56
6.31 World Class Reference ................................................ 57

7 robots-all File Documentation .......................................... 59
  7.1 arch¶ bhv¶ test¶ c File Reference .................................. 59
  7.2 arch¶ bhv¶ testsub¶ c File Reference ................................ 61
  7.3 arch¶ libbehavior¶ bhv¶ main¶ c File Reference ....................... 63
  7.4 arch¶ seq¶ bhvct1.cpp File Reference .............................. 68
  7.5 arch¶ seq¶ sequencer.cpp File Reference ........................... 72
  7.6 driver¶ devices¶ c File Reference .................................. 79
  7.7 driver¶ robotdrv¶ c File Reference ................................ 81
  7.8 driver¶ robotdrv.h File Reference .................................. 83
  7.9 driver¶ syscalls¶ c File Reference .................................. 85
  7.10 include¶ robot.h File Reference .................................... 88
  7.11 include¶ robot¶ behavior.h File Reference ........................ 90
  7.12 include¶ robot¶ constants.h File Reference ....................... 96
  7.13 include¶ robot¶ devfs.h File Reference ............................ 100
  7.14 include¶ robot¶ handle.h File Reference .......................... 109
  7.15 include¶ robot¶ hwid.h File Reference ............................ 112
  7.16 include¶ robot¶ mcilib.h File Reference ........................... 117
  7.17 include¶ robot¶ motors.h File Reference .......................... 119
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.18</td>
<td>include/robot/net.h File Reference</td>
<td>123</td>
</tr>
<tr>
<td>7.19</td>
<td>include/robot/sensors.h File Reference</td>
<td>126</td>
</tr>
<tr>
<td>7.20</td>
<td>include/robot/seq.h File Reference</td>
<td>130</td>
</tr>
<tr>
<td>7.21</td>
<td>include/robot/sys.h File Reference</td>
<td>138</td>
</tr>
<tr>
<td>7.22</td>
<td>include/robot/time.h File Reference</td>
<td>141</td>
</tr>
<tr>
<td>7.23</td>
<td>include/robot/types.h File Reference</td>
<td>144</td>
</tr>
<tr>
<td>7.24</td>
<td>include/robot/util.h File Reference</td>
<td>148</td>
</tr>
<tr>
<td>7.25</td>
<td>interp/data_length.c File Reference</td>
<td>151</td>
</tr>
<tr>
<td>7.26</td>
<td>interp/freq.c File Reference</td>
<td>153</td>
</tr>
<tr>
<td>7.27</td>
<td>interp/interp.c File Reference</td>
<td>156</td>
</tr>
<tr>
<td>7.28</td>
<td>interp/interp.h File Reference</td>
<td>160</td>
</tr>
<tr>
<td>7.29</td>
<td>interp/sensors.c File Reference</td>
<td>165</td>
</tr>
<tr>
<td>7.30</td>
<td>librobot/sensors.c File Reference</td>
<td>168</td>
</tr>
<tr>
<td>7.31</td>
<td>interp/serial.c File Reference</td>
<td>173</td>
</tr>
<tr>
<td>7.32</td>
<td>librobot/devfs.c File Reference</td>
<td>176</td>
</tr>
<tr>
<td>7.33</td>
<td>librobot/devfs_local.c File Reference</td>
<td>182</td>
</tr>
<tr>
<td>7.34</td>
<td>librobot/devfs_net.c File Reference</td>
<td>185</td>
</tr>
<tr>
<td>7.35</td>
<td>librobot/handle.c File Reference</td>
<td>188</td>
</tr>
<tr>
<td>7.36</td>
<td>librobot/mclib.c File Reference</td>
<td>190</td>
</tr>
<tr>
<td>7.37</td>
<td>librobot/motors.c File Reference</td>
<td>192</td>
</tr>
<tr>
<td>7.38</td>
<td>librobot/net.c File Reference</td>
<td>196</td>
</tr>
<tr>
<td>7.39</td>
<td>librobot/seq.c File Reference</td>
<td>202</td>
</tr>
<tr>
<td>7.40</td>
<td>librobot/sys.c File Reference</td>
<td>210</td>
</tr>
<tr>
<td>7.41</td>
<td>librobot/time.c File Reference</td>
<td>214</td>
</tr>
<tr>
<td>7.42</td>
<td>librobot/util.c File Reference</td>
<td>216</td>
</tr>
<tr>
<td>7.43</td>
<td>mc/mch File Reference</td>
<td>218</td>
</tr>
<tr>
<td>7.44</td>
<td>mc/mc_common.c File Reference</td>
<td>223</td>
</tr>
<tr>
<td>7.45</td>
<td>mc/mc_pid.c File Reference</td>
<td>227</td>
</tr>
<tr>
<td>7.46</td>
<td>misc/logger/logger.cpp File Reference</td>
<td>230</td>
</tr>
<tr>
<td>7.47</td>
<td>misc/logger/logger.h File Reference</td>
<td>234</td>
</tr>
<tr>
<td>7.48</td>
<td>misc/logger/logtool.cpp File Reference</td>
<td>235</td>
</tr>
<tr>
<td>7.49</td>
<td>misc/logger/parse_hwid_list.cpp File Reference</td>
<td>239</td>
</tr>
<tr>
<td>7.50</td>
<td>misc/logger/reader.cpp File Reference</td>
<td>240</td>
</tr>
<tr>
<td>7.51</td>
<td>misc/logger/writer.cpp File Reference</td>
<td>241</td>
</tr>
<tr>
<td>7.52</td>
<td>misc/logger/writer.h File Reference</td>
<td>242</td>
</tr>
<tr>
<td>7.53</td>
<td>misc/netrobot/netrobots.cpp File Reference</td>
<td>243</td>
</tr>
</tbody>
</table>
7.54 misc/netrobot/nrtool.cpp File Reference ........................................ 248
7.55 misc/simulator/bumpsensor.h File Reference .................................. 253
7.56 misc/simulator/robot_simulator.cpp File Reference .......................... 254
7.57 misc/simulator/sensorinfo.h File Reference .................................. 261
7.58 misc/simulator/simrobot.cpp File Reference .................................. 262
7.59 misc/simulator/simrobot.h File Reference .................................. 264
7.60 misc/simulator/world.cpp File Reference ..................................... 265
7.61 misc/simulator/world.h File Reference ..................................... 266
7.62 misc/tests/behavior.cpp File Reference ..................................... 267
7.63 misc/tests/drive_in_circle.cpp File Reference ................................. 268
7.64 misc/tests/forward.cpp File Reference ..................................... 269
7.65 misc/tests/freq.cpp File Reference ........................................ 270
7.66 misc/tests/lock.cpp File Reference ........................................ 271
7.67 misc/tests/sonar.cpp File Reference ........................................ 272
7.68 misc/tests/stop.cpp File Reference ........................................ 273
7.69 misc/tests/stop_sensors.cpp File Reference .................................. 274
7.70 misc/tests/time.cpp File Reference ........................................ 275
7.71 pda/mom/discoverydialog.cpp File Reference ................................. 276
7.72 pda/mom/discoverydialog.h File Reference .................................. 277
7.73 pda/mom/infodialog.cpp File Reference ..................................... 278
7.74 pda/mom/infodialog.h File Reference ..................................... 279
7.75 pda/mom/iptputdialog.cpp File Reference .................................. 280
7.76 pda/mom/iptputdialog.h File Reference .................................. 281
7.77 pda/mom/irdialog.cpp File Reference ..................................... 282
7.78 pda/mom/irdialog.h File Reference ..................................... 283
7.79 pda/mom/mom.cpp File Reference ........................................ 284
7.80 pda/mom/momcanvasview.cpp File Reference .................................. 285
7.81 pda/mom/momcanvasview.h File Reference .................................. 286
7.82 pda/mom/momwindow.cpp File Reference .................................. 287
7.83 pda/mom/momwindow.h File Reference .................................. 289
7.84 pda/mom/robotinfo.cpp File Reference ..................................... 290
7.85 pda/mom/robotinfo.h File Reference ..................................... 291
7.86 pda/mom/simulator_constants.h File Reference ............................ 292
7.87 pda/mom/sonardialog.cpp File Reference .................................. 293
7.88 pda/mom/sonardialog.h File Reference .................................. 294

8 robots-all Page Documentation .................................................. 295
8.1 Libbehavior Extra Documentation ........................................... 295
8.2 Sequencer Internals Documentation ........................................... 297
8.3 Compilation Instructions .......................................................... 299
8.4 Librobot Extra Documentation .................................................... 301
8.5 General Robot Software Specifications ........................................ 302
8.6 Low Level Data Flow ................................................................. 305
8.7 Motor Controller Interface ......................................................... 308
8.8 Data Logging ............................................................................. 310
8.9 Simulator Interface .................................................................... 312
8.10 Todo List ..................................................................................... 316
Chapter 1

robots-all Namespace Index

1.1 robots-all Namespace List

Here is a list of all namespaces with brief descriptions:

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dolt</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>std</td>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>
Chapter 2

robots-all Compound Index

2.1 robots-all Compound List

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

- `bhw_connection_t` .................................................. 13
- `bhw_data_t` .......................................................... 15
- `bhw_handle_t` ....................................................... 16
- `bhw_t` ................................................................. 17
- `BumpSensor` ........................................................ 18
- `data_len_t` .......................................................... 19
- `devfs_set_t` .......................................................... 20
- `DiscoveryDialog` ................................................... 21
- `freq_req_val_t` ..................................................... 22
- `InfoDialog` .......................................................... 23
- `IpInputDialog` ....................................................... 24
- `IRDiallog` ............................................................ 25
- `log_writer_t` ......................................................... 26
- `mc_lib_t` ............................................................. 28
- `MomCanvasView` .................................................... 29
- `MomWindow` ........................................................ 30
- `mpProblem` .......................................................... 32
- `Obstacle` ............................................................. 33
- `Point` ................................................................. 34
- `PointTheta` ........................................................... 35
- `robot_dev_t` .......................................................... 36
- `robot_info_t` ........................................................ 38
- `robot_handle_t` ..................................................... 39
- `robot_net_msg_t` .................................................... 41
- `RobotInfo` ........................................................... 42
- `SensorInfo` ........................................................... 43
- `seq_msgbuf_t` ....................................................... 44
- `serial_cmd_t` ........................................................ 45
- `SimRobot` (Used to simulate a configurable differential drive robot) .................................. 46
- `SonarDialog` .......................................................... 56
- `World` ................................................................. 57
Chapter 3

robots-all File Index

3.1 robots-all File List

Here is a list of all files with brief descriptions:

- arch/bhv/test.c (Test behavior) .......................................................... 59
- arch/bhv/testsub.c (Test behavior for loading sub-behaviors) .............. 61
- arch/libbehavior/bhv_main.c (Provides a main program to behaviors, and handles communication with controlling programs (and the sequencer)) .......... 63
- arch/seq/bhvctl.cpp (Start, stop and get status of reactive behaviors) ... 68
- arch/seq/sequencer.cpp ....................................................................... 72
- driver/devices.c (Device info for all the different robot devices) .......... 79
- driver/robotdrv.c (Robot kernel driver for /dev/robot) ....................... 81
- driver/robotdrv.h (Robot driver global include file) .......................... 83
- driver/syscalls.c (System calls (common to all the device types)) ....... 85
- include/robot.h (Global librobot header file (all high-level robot programs should include this)) .................................................. 88
- include/robot/behavior.h (Definitions to be used in the creation of new reactive behaviors) ...................................................... 90
- include/robot/constants.h (Global constants for the robot, should be available to any and all robot software) ..................................... 96
- include/robot/devfs.h (Device-related definitions for the /dev/robot device tree) . 100
- include/robot/handle.h (Robot_handle and associated functionality) ........ 109
- include/robot/hwid.h (Id numbers for hardware devices talking through serial) .... 112
- include/robot/mclib.h (Definitions related to loading a motor control shared library) . 117
- include/robot/motors.h (Send commands to the motors from high-level software, and read sensor data related to the motors) ...................... 119
- include/robot/net.h (Methods for setting up network control of robots) ...... 123
- include/robot/sensors.h (Send frequency commands to sensors and get sensor values (for sonar, ir, bump sensors)) ......................... 126
- include/robot/seq.h (Remote control of reactive behaviors (through the sequencer)) . 130
- include/robot/sys.h (Robot system initialization and shutdown) ............ 138
- include/robot/time.h (Precision timing methods for librobot) ................. 141
- include/robot/types.h (Global types and data sizes for robot software. Mostly for lower-level stuff) ................................................. 144
- include/robot/util.h (Miscellaneous utilities useful both internally to librobot and to external applications using it) .......................... 148
interp/data_lengths.c (Constants indicating the length of data to be sent/received from each type of device (through serial)) .......................... 151
interp/freq.c (Manage sensor "firing frequencies" according to requests by process) 153
interp/interp.c (Interpreter: low-level communication interface between microcontrollers and higher-level software) ............................................. 156
interp/interp.h (Interpreter definitions) ........................................ 160
interp/sensors.c (Sensor data management) .................................... 165
interp/serial.c (Serial communication with microcontrollers controlling sensor devices and motors) ...................................................... 173
librobot/devfs.c (/dev/robot management and communication) ............. 176
librobot/devfs.local.c (Devfs.* functions for talking to a local /dev/robot filesystem) 182
librobot/devfs_net.c (Devfs.* functions for talking to a remote /dev/robot (through netrobot), via a network) .................................. 185
librobot/handle.c (Robot_handle.* related methods, etc) ..................... 188
librobot/milb.c (Load/unload motor control library and initialize and de-initialize it) 190
librobot/motors.c (Implementation of methods from motors.h) .............. 192
librobot/net.c (Methods for setting up network control of robots and sending and receiving data packets) ........................................ 196
librobot/sensors.c (Implementation of methods from sensors.h) ........... 168
librobot/seq.c (Communication with the sequencer) .......................... 202
librobot/sys.c (System initialization and shutdown) ........................... 210
librobot/time.c (Us-precision timing functions for the robot. Implements stuff from time.h) .................................................. 214
librobot/util.c (Utility functions) ............................................. 216
mc/mc.h (Motor control shared library interface and internals) .............. 218
mc/mc_common.c (Implements the functions defined in mc.h that are common to all motor controllers) ............................................ 223
mc/mc_pid.c (Implements the mc_ioctl and mc_init functions for a PID type controller) ...................................................... 227
misc/logger/logger.cpp (Log data from /dev/robot entries, extensively) .... 230
misc/logger/logger.h (Interface for reading from robot log files) ............ 234
misc/logger/logtool.cpp (Tool for printing information from robot log files) 235
misc/logger/parse_hwid_list.cpp (Parse a string of the form a,b,c,d) ........ 239
misc/logger/reader.cpp (Implementation of the log_reader.* class) .......... 240
misc/logger/writer.cpp (Implementation of log_writer.* class) ............. 241
misc/logger/writer.h (Interface for log writer class (only used internally)) ... 242
misc/netrobot/netrootcpp (Remote-control server to be run on the robot) ..... 243
misc/netrobot/nrtool.cpp (A utility for controlling robots over the network) 248
misc/simulator/bumpsensor.h ................................................. 253
misc/simulator/robot_simulator.cpp ......................................... 254
misc/simulator/sensorinfo.h .................................................. 261
misc/simulator/simrobot.cpp ................................................. 262
misc/simulator/simrobot.h (This file defines the Simrobot class) ............ 264
misc/simulator/world.h ....................................................... 265
misc/simulator/world.cpp .................................................... 266
misc/tests/behavior.cpp (Test of the behavioral subsystem) .................. 267
misc/tests/drive_in_circle.cpp (Simple test to make the robot drive in a circle) 268
misc/tests/forward.cpp (Simple test to make the robot drive forward) ....... 269
misc/tests/freq.cpp (Test setting sensor frequencies) ........................ 270
misc/tests/lock.cpp (Test the locking of hardware device control) ........... 271
misc/tests/sonar.cpp (Test sonar sensors) .................................... 272
misc/tests/stop.cpp (Make the robot’s motors stop) ........................... 273
misc/tests/stop_sensors.cpp (Make the robot’s sensors stop firing) .......... 274
misc/tests/time.cpp (Test robot_time us.* ) ................................ 275

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
<table>
<thead>
<tr>
<th>Directory</th>
<th>File Name</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>pda/mom/</td>
<td>discoverydialog.cpp</td>
<td>276</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>discoverydialog.h</td>
<td>277</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>infodialog.cpp</td>
<td>278</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>infodialog.h</td>
<td>279</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>ipinputdialog.cpp</td>
<td>280</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>ipinputdialog.h</td>
<td>281</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>irdialog.cpp</td>
<td>282</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>irdialog.h</td>
<td>283</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>mom.cpp</td>
<td>284</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>momcanvasview.cpp</td>
<td>285</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>momcanvasview.h</td>
<td>286</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>momwindow.cpp</td>
<td>287</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>momwindow.h</td>
<td>289</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>robotinfo.cpp</td>
<td>290</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>robotinfo.h</td>
<td>291</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>simulator_constants.h</td>
<td>292</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>sonardialog.cpp</td>
<td>293</td>
</tr>
<tr>
<td>pda/mom/</td>
<td>sonardialog.h</td>
<td>294</td>
</tr>
</tbody>
</table>
Chapter 4

robots-all Page Index

4.1 robots-all Related Pages

Here is a list of all related documentation pages:

- Libbehavior Extra Documentation ........................................ 295
- Sequencer Internals Documentation ....................................... 297
- Compilation Instructions ...................................................... 299
- Librobot Extra Documentation ........................................... 301
- General Robot Software Specifications .................................. 302
- Low Level Data Flow .......................................................... 305
- Motor Controller Interface ................................................. 308
- Data Logging ........................................................................ 310
- Simulator Interface .............................................................. 312
- Todo List .............................................................................. 316
Chapter 5

robots-all Namespace
Documentation

5.1 dolt Namespace Reference
5.2 std Namespace Reference
Chapter 6

robots-all Class Documentation

6.1  bhv_connection_t Struct Reference

#include <seq.h>
Collaboration diagram for bhv_connection_t:

```
+ bhv_handle_t
   ^
  | from
  |  |
  v  to
```

Public Attributes

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

6.1.1  Member Data Documentation

6.1.1.1  bhv.handle_t* bhv.connection.t::from
behavior sending output

6.1.1.2  uint8_t bhv.connection.t::from_key
output key to send

6.1.1.3  bhv.handle.t* bhv.connection.t::to
behavior receiving input
6.1.1.4  `uint8_t blv_connection_t::to_key`

input key to send to

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

- `include/robot/seq.h`
6.2  bhv_data_t Struct Reference

#include <seq.h>

Public Attributes

- uint8_t key
- uint8_t data [ROBOT_MAX_BHV_DATA_SIZE - 1]

6.2.1  Member Data Documentation

6.2.1.1  uint8_t bhv_data_t::data[ROBOT_MAX_BHV_DATA_SIZE - 1]

6.2.1.2  uint8_t bhv_data_t::key

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

- include/robot/seq.h
6.3  bhv_handle_t Struct Reference

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

Public Attributes

- `pid_t pid`
- `int qid`

6.3.1  Detailed Description

behavior handle

6.3.2  Member Data Documentation

6.3.2.1  `pid_t bhv_handle_t::pid`

behavior process id

6.3.2.2  `int bhv_handle_t::qid`

msg queue id for behavior

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

- include/robot/seq.h
6.4 \texttt{bhv_t} Struct Reference

#include \texttt{<behavior.h>}

Public Attributes

- \texttt{bhv\_cleanup\_func\_t cleanup}
- \texttt{bhv\_inhibit\_func\_t on\_inhibit}
- \texttt{bhv\_uninhibit\_func\_t on\_uninhibit}
- \texttt{bhv\_main\_func\_t main}
- \texttt{uint32\_t flags}

6.4.1 Detailed Description

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

6.4.2 Member Data Documentation

6.4.2.1 \texttt{bhv\_cleanup\_func\_t bhv\_t::cleanup}

6.4.2.2 \texttt{uint32\_t bhv\_t::flags}

flags for initialization and execution; can use \texttt{bhv\_flags\_t flags} as well as any flags sent to \texttt{robot\_init}

6.4.2.3 \texttt{bhv\_main\_func\_t bhv\_t::main}

6.4.2.4 \texttt{bhv\_inhibit\_func\_t bhv\_t::on\_inhibit}

6.4.2.5 \texttt{bhv\_uninhibit\_func\_t bhv\_t::on\_uninhibit}

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

- \texttt{include/robot/behavior.h}
6.5 BumpSensor Class Reference

#include <bumpsensor.h>

Public Member Functions

- BumpSensor()
- BumpSensor(dolt::Polygon p, float x, float y, float theta)
  - float x()
  - float y()
  - float t()

6.5.1 Constructor & Destructor Documentation

6.5.1.1 BumpSensor::BumpSensor() [inline]

6.5.1.2 BumpSensor::BumpSensor(dolt::Polygon p, float x, float y, float theta) [inline]

6.5.2 Member Function Documentation

6.5.2.1 float BumpSensor::t() [inline]

6.5.2.2 float BumpSensor::x() [inline]

6.5.2.3 float BumpSensor::y() [inline]

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

- misc/simulator/bumpsensor.h
6.6 data_len_t Struct Reference

#include <interp.h>

Public Attributes

- uint8_t from_dev
- uint8_t to_dev

6.6.1 Detailed Description

specifies the length of data from/to each hw device see data_lengths.c

6.6.2 Member Data Documentation

6.6.2.1 uint8_t data_len_t::from_dev

6.6.2.2 uint8_t data_len_t::to_dev

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

- interp/interp.h
6.7  devfs_set_t Struct Reference

#include <devfs.h>

Public Attributes

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

6.7.1 Detailed Description

file descriptors for the /dev/robot stuff

6.7.2 Member Data Documentation

6.7.2.1 int devfs_set_t::change

6.7.2.2 int devfs_set_t::ctl

file descriptors

6.7.2.3 int devfs_set_t::current

6.7.2.4 uint8_t devfs_set_t::hwid

hardware id (hwid.h)

6.7.2.5 int devfs_set_t::stream

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

- include/robot/devfs.h
6.8 DiscoveryDialog Class Reference

#include <discoverydialog.h>

Collaboration diagram for DiscoveryDialog:

```
#include <discoverydialog.h>

Collaboration diagram for DiscoveryDialog:
```

Public Slots

- void discover ()
- void selectionChanged (int)

Public Member Functions

- DiscoveryDialog (QWidget *parent=0, const char *name=0, bool modal=FALSE, WFlags f=0)
- ~DiscoveryDialog ()
- robot_handle_t * getSelection ()

6.8.1 Constructor & Destructor Documentation

6.8.1.1 DiscoveryDialog::DiscoveryDialog (QWidget * parent = 0, const char * name = 0, bool modal = FALSE, WFlags f = 0)

6.8.1.2 DiscoveryDialog::~DiscoveryDialog ()

6.8.2 Member Function Documentation

6.8.2.1 void DiscoveryDialog::discover () [slot]

6.8.2.2 robot_handle_t * DiscoveryDialog::getSelection ()

6.8.2.3 void DiscoveryDialog::selectionChanged (int) [slot]

The documentation for this class was generated from the following files:

- pda/mom/discoverydialog.h
- pda/mom/discoverydialog.cpp

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
6.9  freq_req_val_t Struct Reference

#include <types.h>

Public Attributes

- owner_val_t owner
- freq_val_t freq

6.9.1  Member Data Documentation

6.9.1.1  freq_val_t freq_req_val_t::freq

6.9.1.2  owner_val_t freq_req_val_t::owner

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

- include/robot/types.h
6.10  InfoDialog Class Reference

#include <infodialog.h>

Public Member Functions

- InfoDialog (RobotInfo *robot, robot_handle_t *handle, QWidget *parent=0, const char *name=0, bool modal=false, WFlags f=0)
- ~InfoDialog ()

6.10.1  Constructor & Destructor Documentation

6.10.1.1  InfoDialog::InfoDialog (RobotInfo *robot, robot_handle_t *handle, QWidget *parent = 0, const char *name = 0, bool modal = false, WFlags f = 0)

6.10.1.2  InfoDialog::~InfoDialog ()

The documentation for this class was generated from the following files:

- pda/mom/infodialog.h
- pda/mom/infodialog.cpp
6.11 IpInputDialog Class Reference

#include <ipinputdialog.h>

Public Member Functions

- IpInputDialog (QWidget *parent=0, const char *name=0, bool modal=FALSE, WFlags f=0)
- ~IpInputDialog ()
- const char * get_ip ()
- const char * get_port ()

Protected Slots

- void accept ()

6.11.1 Constructor & Destructor Documentation

6.11.1.1 IpInputDialog::IpInputDialog (QWidget * parent = 0, const char * name = 0, bool modal = FALSE, WFlags f = 0)

6.11.1.2 IpInputDialog::~IpInputDialog ()

6.11.2 Member Function Documentation

6.11.2.1 void IpInputDialog::accept () [protected, slot]

6.11.2.2 const char * IpInputDialog::get_ip ()

6.11.2.3 const char * IpInputDialog::get_port ()

The documentation for this class was generated from the following files:

- pda/mom/ipinputdialog.h
- pda/mom/ipinputdialog.cpp
6.12 IRDialog Class Reference

#include <irdialog.h>

Collaboration diagram for IRDialog:

Public Slots

- void getInfo ()

Public Member Functions

- IRDialog (RobotInfo *r, robot_handle_t *h, QWidget *parent=0, const char *name=0, bool modal=TRUE, WFlags f=0)
- ~IRDialog ()

6.12.1 Constructor & Destructor Documentation

6.12.1.1 IRDialog::IRDialog (RobotInfo *r, robot_handle_t *h, QWidget *parent = 0, const char * name = 0, bool modal = TRUE, WFlags f = 0)

6.12.1.2 IRDialog::~IRDialog ()

6.12.2 Member Function Documentation

6.12.2.1 void IRDialog::getInfo () [slot]

The documentation for this class was generated from the following files:

- pda/mom/irdialog.h
- pda/mom/irdialog.cpp
6.13 log_writer_t Class Reference

#include <writer.h>

Public Member Functions

- log_writer_t (uint32_t rsz=ROBOT_DEFAULT_LOG_ROTATE_SIZE)
  Constructor.

- ~log_writer_t ()
  Destructor.

- bool open (const char *file)
  Open a log file for appending.

- void close ()
  Close file if open.

- uint32_t size ()
- bool write (log_entry_t *entry)
  Write a log entry to the log file.

6.13.1 Constructor & Destructor Documentation

6.13.1.1 log_writer_t::log_writer_t (uint32_t rsz = ROBOT_DEFAULT_LOG_ROTATE_SIZE)

Constructor.
Does not open any files!

See also:
  open

6.13.1.2 log_writer_t::~log_writer_t ()

Destructor.
Closes file if open.

See also:
  close

6.13.2 Member Function Documentation

6.13.2.1 void log_writer_t::close ()

Close file if open.
6.13.2.2 bool log_writer_t::open (const char * file)
Open a log file for appending.

Parameters:
  *file* Path of log file to open

Returns:
  true on success, false on failure or if file is currently locked by another writer

6.13.2.3 uint32_t log_writer_t::size ()

Returns:
  Size, in bytes, of the log file; 0 if no file is open.

6.13.2.4 bool log_writer_t::write (log_entry_t * entry)
Write a log entry to the log file.

Parameters:
  *entry* Pointer to a log entry to write

Returns:
  true on success, false on failure

The documentation for this class was generated from the following files:

- misc/logger/writer.h
- misc/logger/writer.cpp
6.14 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

6.14.1 Detailed Description

structure containing pointers to all library functions

6.14.2 Member Data Documentation

6.14.2.1 void* mc_lib_t::handle

dlopen handle

6.14.2.2 mc_do_control_func_t mc_lib_t::mc_do_control

6.14.2.3 mc_get_velocity_func_t mc_lib_t::mc_get_velocity

6.14.2.4 mc_init_func_t mc_lib_t::mc_init

6.14.2.5 mc_set_odometry_func_t mc_lib_t::mc_set_odometry

6.14.2.6 mc_set_velocity_func_t mc_lib_t::mc_set_velocity

6.14.2.7 mc_shutdown_func_t mc_lib_t::mc_shutdown

6.14.2.8 mc_start_frame_func_t mc_lib_t::mc_start_frame

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

- include/robot/mclib.h
6.15 MomCanvasView Class Reference

#include <momcanvasview.h>

Public Slots

- void contentsMouseMoveEvent (QMouseEvent *event)
- void contentsMouseReleaseEvent (QMouseEvent *event)

Signals

- void positionChanged (int, int)

Public Member Functions

- MomCanvasView (QCanvas *c, QWidget *parent)
- ~MomCanvasView ()

Static Public Attributes

- const int xOffset = 118
- const int yOffset = 160

6.15.1 Constructor & Destructor Documentation

6.15.1.1 MomCanvasView::MomCanvasView (QCanvas * c, QWidget * parent)

6.15.2 Member Function Documentation

6.15.2.1 void MomCanvasView::contentsMouseMoveEvent (QMouseEvent * event) [slot]

6.15.2.2 void MomCanvasView::contentsMouseReleaseEvent (QMouseEvent * event) [slot]

6.15.2.3 void MomCanvasView::positionChanged (int, int) [signal]

6.15.3 Member Data Documentation

6.15.3.1 const int MomCanvasView::xOffset = 118 [static]

6.15.3.2 const int MomCanvasView::yOffset = 160 [static]

The documentation for this class was generated from the following files:

- pda/mom/momcanvasview.h
- pda/mom/momcanvasview.cpp

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
6.16 MomWindow Class Reference

#include <momwindow.h>

Collaboration diagram for MomWindow:

Public Slots

- void about ()
- void updateStatus (int x, int y)
- void updateScreen (int x, int y)
- void ipWindow ()
- void discoveryWindow ()
- void disconnect ()
- void moveRobot (int x, int y)
- void haltRobot ()
- void robotStatus ()
- void sonarStatus ()
- void infraredStatus ()

Public Member Functions

- MomWindow ()
- RobotInfo & getRInfo ()
- robot_handle_t * getHandle ()
- virtual ~MomWindow ()
6.16.1 Constructor & Destructor Documentation

6.16.1.1 MomWindow::MomWindow ()

6.16.1.2 MomWindow::~MomWindow () [virtual]

6.16.2 Member Function Documentation

6.16.2.1 void MomWindow::about () [slot]

6.16.2.2 void MomWindow::disconnect () [slot]

6.16.2.3 void MomWindow::discoveryWindow () [slot]

6.16.2.4 robot_handle_t* MomWindow::getHandle () [inline]

6.16.2.5 RobotInfo& MomWindow::getRInfo () [inline]

6.16.2.6 void MomWindow::haltRobot () [slot]

6.16.2.7 void MomWindow::infraredStatus () [slot]

6.16.2.8 void MomWindow::ipWindow () [slot]

6.16.2.9 void MomWindow::moveRobot (int x, int y) [slot]

6.16.2.10 void MomWindow::robotStatus () [slot]

6.16.2.11 void MomWindow::sonarStatus () [slot]

6.16.2.12 void MomWindow::updateScreen (int x, int y) [slot]

6.16.2.13 void MomWindow::updateStatus (int x, int y) [slot]

The documentation for this class was generated from the following files:

- pda/mom/momwindow.h
- pda/mom/momwindow.cpp
6.17 mpProblem Class Reference

#include <world.h>

Collaboration diagram for mpProblem:

```
        World
           |
           w
           |
        mpProblem
```

**Public Member Functions**

- **mpProblem** (const char *fname, bool verbose=false)
- SimRobot * getNextRobot ()
- log_reader_t * getNextLog ()

**Public Attributes**

- char problemName [MAX_PROBLEM_NAME_LEN]
- World * w

6.17.1 Constructor & Destructor Documentation

6.17.1.1 mpProblem::mpProblem (const char * fname, bool verbose = false)

6.17.2 Member Function Documentation

6.17.2.1 log_reader_t * mpProblem::getNextLog ()

6.17.2.2 SimRobot * mpProblem::getNextRobot ()

6.17.3 Member Data Documentation

6.17.3.1 char mpProblem::problemName[MAX_PROBLEM_NAME_LEN]

6.17.3.2 World* mpProblem::w

The documentation for this class was generated from the following files:

- misc/simulator/world.h
- misc/simulator/world.cpp
6.18 Obstacle Class Reference

#include <world.h>

Public Member Functions

- Obstacle ()

Public Attributes

- string name

6.18.1 Constructor & Destructor Documentation

6.18.1.1 Obstacle::Obstacle () [inline]

6.18.2 Member Data Documentation

6.18.2.1 string Obstacle::name

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

- misc/simulator/world.h
6.19 Point Class Reference

#include <robotinfo.h>

Public Attributes

- float x
- float y

6.19.1 Member Data Documentation

6.19.1.1 float Point::x

6.19.1.2 float Point::y

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

- pda/mom/robotinfo.h
6.20 PointTheta Class Reference

#include <robotinfo.h>

Public Attributes

- float x
- float y
- float t

6.20.1 Member Data Documentation

6.20.1.1 float PointTheta::t

6.20.1.2 float PointTheta::x

6.20.1.3 float PointTheta::y

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

- pda/mom/robotinfo.h
6.21 robot_dev_t Struct Reference

#include <robotdrv.h>

Public Attributes

- uint8_t type
- uint8_t data_size
- devfs_handle_t handle
- uint8_t wcnt
- uint8_t rcnt
- semaphore sem
- uint8_t * data
- uint32_t dcnt
- wait_queue_head_t readq
- pid_t lock_owner
- uint32_t lock_prio

6.21.1 Detailed Description

our representation of a /dev/robot entry

6.21.2 Member Data Documentation

6.21.2.1 uint8_t* robot_dev_t::data

current data

6.21.2.2 uint8_t robot_dev_t::data_size

size of data

6.21.2.3 uint32_t robot_dev_t::dcnt

number of writes (wrap ok)

6.21.2.4 devfs_handle_t robot_dev_t::handle

devfs handle

6.21.2.5 pid_t robot_dev_t::lock_owner

owner of current lock (if any)

6.21.2.6 uint32_t robot_dev_t::lock_prio

priority of current lock
6.21.2.7 uint8_t robot_dev_t::rcnt
num current writers/readers

6.21.2.8 wait_queue_head_t robot_dev_t::readq
blocking/polling

6.21.2.9 struct semaphore robot_dev_t::sem

6.21.2.10 uint8_t robot_dev_t::type
device type

6.21.2.11 uint8_t robot_dev_t::wcnt

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

- driver/robotdrv.h
6.22 robot_finfo_t Struct Reference

#include <robotdrv.h>

Collaboration diagram for robot_finfo_t:

```plaintext
collaboration diagram
```

Public Attributes

- `uint32_t dcnt`
- `robot_dev_t * dev`

6.22.1 Detailed Description

file information for each open file

6.22.2 Member Data Documentation

6.22.2.1 `uint32_t robot_finfo_t::dcnt`

the dcnt of dev when i last read

6.22.2.2 `robot_dev_t* robot_finfo_t::dev`

the device i have open

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

- `driver/robotdrv.h`
6.23  robot_handle_t Struct Reference

#include <handle.h>

Collaboration diagram for robot_handle_t:

```
+----------------+    +-----------------+
| devfs_set_t    |    | devfs_fds        |
+----------------+    +-----------------+
|                 |    | robot_handle_t  |
```

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

6.23.1 Detailed Description

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

6.23.2 Member Data Documentation

6.23.2.1 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).

6.23.2.2 uint32_t robot_handle_t::devfs_flags

passed to devfs_init

6.23.2.3 uint32_t robot_handle_t::flags

passed to robot_init
6.23.2.4  

robot_id_t robot_handle_t::id

unique robot id number

6.23.2.5  

uint8_t robot_handle.t::init_complete

has robot_init been called successfully?

6.23.2.6  

char robot_handle.t::name[ROBOT_MAX_NAME_LEN]

robot name

6.23.2.7  

int robot_handle.t::sock

network socket if necessary

6.23.2.8  

struct sockaddr.in robot_handle.t::sockaddr

6.23.2.9  

uint32_t robot_handle.t::timeout

network timeout

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

- include/robot/handle.h
6.24 \texttt{robot\_net\_msg\_t} Struct Reference

\#include <net.h>

Public Attributes

- \texttt{uint8\_t hwid}
- \texttt{uint8\_t devfs\_type}
- \texttt{uint32\_t mid}
- \texttt{uint32\_t count}
- \texttt{uint8\_t * buf}

6.24.1 Member Data Documentation

6.24.1.1 \texttt{uint8\_t* robot\_net\_msg\_t::buf}

data buffer for message

6.24.1.2 \texttt{uint32\_t robot\_net\_msg\_t::count}

size of data in buf (bytes)

6.24.1.3 \texttt{uint8\_t robot\_net\_msg\_t::devfs\_type}

device type (devfs.h)

6.24.1.4 \texttt{uint8\_t robot\_net\_msg\_t::hwid}

hardware id from hwid.h

6.24.1.5 \texttt{uint32\_t robot\_net\_msg\_t::mid}

managed internally

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

- \texttt{include/robot/net.h}
6.25 RobotInfo Class Reference

#include <robotinfo.h>

Collaboration diagram for RobotInfo:

```
PointTheta -- sonars / boundary
Point
```

Public Member Functions

- RobotInfo()
- ~RobotInfo()
- QCanvasPolygon* getBoundarySonar (QCanvas *c)
- QCanvasPolygon* getSonarPolygon (QCanvas *c, int sensor, float dist)
- QCanvasPolygon* getBoundaryIR (QCanvas *c)
- QCanvasLine* getIRLine (QCanvas *c, int sensor, float dist)

6.25.1 Constructor & Destructor Documentation

6.25.1.1 RobotInfo::RobotInfo ()

6.25.1.2 RobotInfo::~RobotInfo ()

6.25.2 Member Function Documentation

6.25.2.1 QCanvasPolygon* RobotInfo::getBoundaryIR (QCanvas *c)

6.25.2.2 QCanvasPolygon* RobotInfo::getBoundarySonar (QCanvas *c)

6.25.2.3 QCanvasLine* RobotInfo::getIRLine (QCanvas *c, int sensor, float dist)

6.25.2.4 QCanvasPolygon* RobotInfo::getSonarPolygon (QCanvas *c, int sensor, float dist)

The documentation for this class was generated from the following files:

- pda/mom/robotinfo.h
- pda/mom/robotinfo.cpp
6.26 SensorInfo Class Reference

#include <sensorinfo.h>

Public Member Functions

- SensorInfo ()
- SensorInfo (float x, float y, float theta)
- float t ()

6.26.1 Constructor & Destructor Documentation

6.26.1.1 SensorInfo::SensorInfo () [inline]

6.26.1.2 SensorInfo::SensorInfo (float x, float y, float theta) [inline]

6.26.2 Member Function Documentation

6.26.2.1 float SensorInfo::t () [inline]

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

- misc/simulator/sensorinfo.h
6.27  seq_msgbuf_t Struct Reference

#include <seq.h>

Public Attributes

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

6.27.1  Detailed Description

message queue buffer

6.27.2  Member Data Documentation

6.27.2.1  uint8_t seq_msgbuf_t::cmd

command to send (see enum)

6.27.2.2  uint8_t seq_msgbuf_t::data[ROBOT_MAX_BHV_DATA_SIZE]

6.27.2.3  long seq_msgbuf_t::mtype

should always be pid of sender
The documentation for this struct was generated from the following file:

- include/robot/seq.h
6.28 serial_cmd_t Struct Reference

#include <interp.h>

Public Attributes

- uint8_t hwid
- uint8_t data [4]

6.28.1 Detailed Description

A command to send to or receive from a hardware device through serial.

6.28.2 Member Data Documentation

6.28.2.1 uint8_t serial_cmd_t::data[4]

Max amount is 4 bytes.

6.28.2.2 uint8_t serial_cmd_t::hwid

A hardware id from hwid.h

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

- interp/interp.h
6.29 SimRobot Class Reference

The SimRobot class is used to simulate a configurable differential drive robot.

```
#include <simrobot.h>
```

Collaboration diagram for SimRobot:

```
SimRobot
  `-- odometryReference
     `-- SensorInfo
```

Public Member Functions

- **SimRobot** (const char *fname, int port, bool verbose=false)
- void **setV** (float velocity)
  
  *Set the translational velocity of the robot.*

- void **setW** (float velocity)
  
  *Set the rotational velocity of the robot.*

- void **get_all_sonar_data** (float **info, bool verbose=false)
  
  *Get sonar data from all of sonar sensors on the robot.*

- void **get_sonar_data** (float *info, int sensor, bool verbose=false)
  
  *Get sonar data from a specific sonar sensor.*

- void **get_all_ir_data** (float **info, bool verbose=false)
  
  *Get infrared data from all of IR sensors on the robot.*

- void **get_ir_data** (float *info, int sensor, bool verbose=false)
  
  *Get infrared data from a specific IR sensor.*

- void **getBumpData** (unsigned char **info)
  
  *Get bump sensor data from all of the bump sensors on the robot.*

- void **getBumpData** (unsigned char *info, int sensor)
  
  *Get bump sensor data from a specific bump sensor on the robot.*

- void **getOdometry** (float *x, float *y, float *theta)
  
  *Get odometry data for the robot.*

- void **setOdometry** (float x, float y, float theta)
  
  *Set the robot’s odometry so that the current odometry values are equal to those passed to the function.*

- void **move** (float time)
  
  *Tell the robot that a certain amount of time has passed.*
• string & **getName** ()
  *Get the name of this robot.*

• drawablePoint * **get_true_cor** ()
  *Get a visual representation of the robot’s center of rotation.*

• drawablePoint * **get_estimated_cor** ()
  *Get a visual representation of the robot’s estimated center of rotation.*

• drawableLineStrip * **getSonarScan** ()
  *Get a visual representation of the last sonar scan.*

• drawableLineStrip * **getPath** ()
  *Get a visual representation of the path that this robot has traveled.*

• objectList & **getIRBeams** ()
  *Get a visual representation of the last infrared scan.*

• objectList & **getSonarObjects** ()
  *Get a visual representation of the sonar sensors on this robot.*

• objectList & **getIRObjects** ()
  *Get a visual representation of the IR sensors on this robot.*

• objectList & **getBumpObjects** ()
  *Get a visual representation of the bump sensors on this robot.*

• float **getV** ()
  *Get the robot’s current translational velocity.*

• float **getTargetV** ()
  *Get the robot’s target translational velocity.*

• float **getW** ()
  *Get the robot’s current rotational velocity.*

• float **getTargetW** ()
  *Get the robot’s target rotational velocity.*

• float **getA** ()
  *Get the robot’s current acceleration value.*

• void **setA** (float acceleration)
  *Set the robot’s current acceleration value.*

• unsigned int **getNumSonars** ()
Get the number of sonar sensors on the robot.

- `unsigned int getNumIRs ()`
  Get the number of infrared sensors on the robot.

- `unsigned int getNumBumps ()`
  Get the number of bump sensors on the robot.

- `bool odometryChanged ()`
  Check to see if the robot's odometry has changed.

- `void set_all_sonar_frequencies (float frequency)`
  Set the frequency of all of the sonar sensors.

- `void set_sonar_frequency (int sensor, float frequency)`
  Set a sonar sensor to fire at a certain frequency.

- `void set_all_ir_frequencies (float frequency)`
  Set the frequency of all of the IR sensors.

- `void set_ir_frequency (int sensor, float frequency)`
  Set an IR sensor to fire at a certain frequency.

### 6.29.1 Detailed Description

The SimRobot class is used to simulate a configurable differential drive robot.

### 6.29.2 Constructor & Destructor Documentation

#### 6.29.2.1 SimRobot::SimRobot (const char * `fname`, int `port`, bool `verbose = false`)
6.29.3.2 void SimRobot::get_all_sonar_data (float ** info, bool verbose = false)

Get sonar data from all of sonar sensors on the robot.

This function generates simulated sonar data based on the positions and orientations of the sonar sensors on the robot. It is important to note that this function only accounts for readings that result from direct echoes back to the robot and does not account for the possibilities of multiple reflections that may occur in real life.

Parameters:

info An uninitialized pointer that will point to an array containing the sonar data after the function returns. The user is not responsible for this memory and should not try to delete it!

verbose Specifies whether or not drawable versions of the sonar cones should be generated.

6.29.3.3 drawablePoint* SimRobot::get_estimated_cor () [inline]

Get a visual representation of the robot’s estimated center of rotation.

Returns:

Returns a pointer to a drawablePoint which defines this robot’s estimated center of rotation. Useful for seeing how much odometry drift has occurred.

6.29.3.4 void SimRobot::get_ir_data (float * info, int sensor, bool verbose = false)

Get infrared data from a specific IR sensor.

This function generates simulated infrared data based on the positions and orientations of the IR sensors on the robot.

Parameters:

info A valid pointer. The IR data will be placed here when the function returns.

sensor Specifies the IR sensor to get data from.

verbose Specifies whether or not a drawable version of the IR beam should be generated.

6.29.3.5 void SimRobot::get_sonar_data (float * info, int sensor, bool verbose = false)

Get sonar data from a specific sonar sensor.

This function generates simulated sonar data based on the positions and orientations of the sonar sensors on the robot. It is important to note that this function only accounts for readings that result from direct echoes back to the robot and does not account for the possibilities of multiple reflections that may occur in real life.

Parameters:

info A valid pointer. The sonar data will be placed here when the function returns.

sensor Specifies the sonar sensor to get data from.

verbose Specifies whether or not a drawable version of the sonar cone should be generated.
6.29.3.6 \texttt{drawablePoint* SimRobot::get\_true\_cor () [inline]}

Get a visual representation of the robot’s center of rotation.

\textbf{Returns:}

- Returns a pointer to a \texttt{drawablePoint} which defines this robot’s true center of rotation.

6.29.3.7 \texttt{float SimRobot::getA () [inline]}

Get the robot’s current acceleration value.

\textbf{Returns:}

- Returns the robot’s current acceleration value.

6.29.3.8 \texttt{void SimRobot::getBumpData (unsigned char * info, int sensor)}

Get bump sensor data from a specific bump sensor on the robot.

This function generates simulated bump sensor data based on the positions and orientations of the IR sensors on the robot. A bump sensor "fires" if after a specified action the bump sensor would penetrate an object such as the world boundary or an obstacle.

\textbf{Parameters:}

- \textit{info} A valid pointer. The data from the bump sensor will be placed here when the function returns.

6.29.3.9 \texttt{void SimRobot::getBumpData (unsigned char ** info)}

Get bump sensor data from all of the bump sensors on the robot.

This function generates simulated bump sensor data based on the positions and orientations of the IR sensors on the robot. A bump sensor "fires" if after a specified action the bump sensor would penetrate an object such as the world boundary or an obstacle.

\textbf{Parameters:}

- \textit{info} An uninitialized pointer that will point to an array containing the bump sensor data after the function returns. The user is not responsible for this memory and should not try to delete it!

6.29.3.10 \texttt{objectList& SimRobot::getBumpObjects () [inline]}

Get a visual representation of the bump sensors on this robot.

\textbf{Returns:}

- Returns a reference to an \texttt{objectList} that contains \texttt{drawablePolygons}. These \texttt{drawablePolygons} define the positions and orientations of the bump sensors of the robot.
6.29.3.11 drawableLineStrip* SimRobot::getEstimatedPath () [inline]

Get a visual representation of the path that the robot thinks it has traveled.

Returns:
  Returns a pointer to a drawableLineStrip that shows the path the robot thinks has taken
during the simulation. This differs from the actual path because of odometry drift.

6.29.3.12 objectList& SimRobot::getIRBeams () [inline]

Get a visual representation of the last infrared scan.

Returns:
  Returns a reference to an objectList that defines the infrared beams generated by the last
infrared scan. When calling getIRData, verbose must be set to true.

6.29.3.13 objectList& SimRobot::getIRObjcts () [inline]

Get a visual representation of the IR sensors on this robot.

Returns:
  Returns a reference to an objectList that contains drawablePoints. These drawablePoints
define the position of the IR sensors on the robot.

6.29.3.14 string& SimRobot::getName () [inline]

Get the name of this robot.

Returns:
  Returns a string that contains the name of the robot.

6.29.3.15 unsigned int SimRobot::getNumBumps () [inline]

Get the number of bump sensors on the robot.

Returns:
  Returns the number of bump sensors on the robot.

6.29.3.16 unsigned int SimRobot::getNumIRs () [inline]

Get the number of infrared sensors on the robot.

Returns:
  Returns the number of sonar sensors on the robot.
6.29.3.17 unsigned int SimRobot::getNumSonars () [inline]

Get the number of sonar sensors on the robot.

Returns:
Returns the number of sonar sensors on the robot.

6.29.3.18 void SimRobot::getOdometry (float * x, float * y, float * theta)

Get odometry data for the robot.

Parameters:
- \(x\) A valid pointer. The x component of the robot’s odometry will be placed here.
- \(y\) A valid pointer. The y component of the robot’s odometry will be placed here.
- \(\theta\) A valid pointer. The theta component of the robot’s odometry will be placed here.

6.29.3.19 drawableLineStrip* SimRobot::getPath () [inline]

Get a visual representation of the path that this robot has traveled.

Returns:
Returns a pointer to a drawableLineStrip that shows the path this robot has taken during the simulation.

6.29.3.20 objectList& SimRobot::getSonarObjects () [inline]

Get a visual representation of the sonar sensors on this robot.

Returns:
Returns a reference to an objectList that contains drawablePoints. These drawablePoints define the position of the sonar sensors on the robot.

6.29.3.21 drawableLineStrip* SimRobot::getSonarScan () [inline]

Get a visual representation of the last sonar scan.

Returns:
Returns a pointer to a drawableLineStrip that defines the sonar cones generated by the last sonar scan. When calling getSonarData, verbose must be set to true.

6.29.3.22 float SimRobot::getTargetV () [inline]

Get the robot’s target translational velocity.

Returns:
Returns the robot’s target translational velocity.
6.29.3.23  float SimRobot::getTargetW ()  [inline]

Get the robot’s target rotational velocity.

Returns:
Returns the robot’s target rotational velocity.

6.29.3.24  float SimRobot::getV ()  [inline]

Get the robot’s current translational velocity.

Returns:
Returns the robot’s current translational velocity.

6.29.3.25  float SimRobot::getW ()  [inline]

Get the robot’s current rotational velocity.

Returns:
Returns the robot’s current rotational velocity.

6.29.3.26  void SimRobot::move (float time)

Tell the robot that a certain amount of time has passed. Based on the given discrete time value, the actions of the robot during this time will be approximated.

Parameters:
- time  The amount of time (in seconds) that the robot should simulate its actions for.

6.29.3.27  bool SimRobot::odometryChanged ()

Check to see if the robot’s odometry has changed.

Returns:
Returns true if the robot has moved since the last time interval. Returns false if the robot has not moved since that time.

6.29.3.28  void SimRobot::set_all_ir_frequencies (float frequency)

Set the frequency of all of the ir sensors.

This function will set the frequency of the ir sensors so that frequency number of readings is taken by each sensor per second. This function staggers the firing times of the sensors evenly.

Parameters:
- frequency  The frequency in Hz.
6.29.3.29  void SimRobot::set_all_sonar_frequencies (float frequency)

Set the frequency of all of the sonar sensors. This function will set the frequency of the sonar sensors so that frequency number of readings is taken by each sensor per second. This function staggers the firing times of the sensors evenly.

Parameters:
  frequency  The frequency in Hz.

6.29.3.30  void SimRobot::set_ir_frequency (int sensor, float frequency)

Set an ir sensor to fire at a certain frequency. This function will set the frequency of a specified ir sensor so that frequency number of readings is taken by by that sensor per second. This

Parameters:
  sensor   The sensor that should have its frequency set
  frequency  The frequency in Hz.

6.29.3.31  void SimRobot::set_sonar_frequency (int sensor, float frequency)

Set a sonar sensor to fire at a certain frequency. This function will set the frequency of a specified sonar sensor so that frequency number of readings is taken by by that sensor per second. This

Parameters:
  sensor   The sensor that should have its frequency set
  frequency  The frequency in Hz.

6.29.3.32  void SimRobot::set (float acceleration)  [inline]

Set the robot’s current acceleration value.

Returns:
  Returns the robot’s current acceleration value.

6.29.3.33  void SimRobot::setOdometry (float x, float y, float theta)

Set the robot’s odometry so that the current odometry values are equal to those passed to the function.

Parameters:
  x  What the x component of the odometry should be.
  y  What the y component of the odometry should be.
  theta  What the theta component of the odometry should be.
6.29.3.34  void SimRobot::setV (float velocity)

Set the translational velocity of the robot.
This does not guarantee that the robot will immediately travel at the specified velocity. The robot
may have to accelerate to reach this value.

Parameters:
  velocity The target velocity (in meters/second) at which the robot should travel.

6.29.3.35  void SimRobot::setW (float velocity)

Set the rotational velocity of the robot.
This does not guarantee that the robot will immediately travel at the specified velocity. The robot
may have to accelerate to reach this value.

Parameters:
  velocity The target velocity (in radians/second) at which the robot should travel.

The documentation for this class was generated from the following files:

- misc/simulator/simrobot.h
- misc/simulator/simrobot.cpp
6.30 SonarDialog Class Reference

#include <sonardialog.h>

Collaboration diagram for SonarDialog:

Public Slots

- void getInfo ()

Public Member Functions

- SonarDialog (RobotInfo *r, robot_handle_t *h, QWidget *parent=0, const char *name=0, bool modal=TRUE, WFlags f=0)
- ~SonarDialog ()

6.30.1 Constructor & Destructor Documentation

6.30.1.1 SonarDialog::SonarDialog (RobotInfo * r, robot_handle_t * h, QWidget * 
parent = 0, const char * name = 0, bool modal = TRUE, WFlags f = 0)

6.30.1.2 SonarDialog::~SonarDialog ()

6.30.2 Member Function Documentation

6.30.2.1 void SonarDialog::getInfo () [slot]

The documentation for this class was generated from the following files:

- pda/mom/sonardialog.h
- pda/mom/sonardialog.cpp
6.31 World Class Reference

#include <world.h>

Public Member Functions

• World (const char *fname, bool verbose=false)

Public Attributes

• string name
• dolt::drawablePolygon * boundary
• std::list< Obstacle * > obstacles

6.31.1 Constructor & Destructor Documentation

6.31.1.1 World::World (const char *fname, bool verbose = false)

6.31.2 Member Data Documentation

6.31.2.1 dolt::drawablePolygon* World::boundary

6.31.2.2 string World::name

6.31.2.3 std::list<Obstacle*> World::obstacles

The documentation for this class was generated from the following files:

• misc/simulator/world.h
• misc/simulator/world.cpp
Chapter 7

robots-all File Documentation

7.1 arch/bhv/test.c File Reference

Test behavior.
#include <robot/behavior.h>
#include <robot.h>
#include <unistd.h>

Include dependency graph for test.c:

Functions

- void test_cleanup (void)
- void test_inhibit (void)
- void test_uninhibit (void)
- int test_main (void)
- int test_ondata (uint8_t key, const void *data, int size)
- int bhv_init (bhv_t *bhv)

Entry point into the behavior.

7.1.1 Detailed Description

Test behavior.

Author:
Kris Beevers (beevek@cs.rpi.edu)
Version:
\texttt{test.c,v 1.6 2003/08/19 18:42:45 beevk Exp}

Doesn’t really do anything, just prints stuff to a log file. Echoes data received on input directly to output.

7.1.2 Function Documentation

7.1.2.1 \textbf{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).

\textbf{Parameters:}

- \textit{bhv} Pointer to a \texttt{bhv\_t} struct to be initialized by the behavior

\textbf{Returns:}

- $< 0$ on failure, $\geq 0$ on success

7.1.2.2 \textbf{void test\_cleanup (void)}

7.1.2.3 \textbf{void test\_inhibit (void)}

7.1.2.4 \textbf{int test\_main (void)}

7.1.2.5 \textbf{int test\_ondata (uint8\_t key, const void * data, int size)}

7.1.2.6 \textbf{void test\_uninhibit (void)}
7.2 arch/bhv/testsub.c File Reference

Test behavior for loading sub-behaviors.

```c
#include <robot/behavior.h>
#include <robot.h>
```

Include dependency graph for testsub.c:

```
robot/behavior.h  robot.h
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>testsub.c</td>
</tr>
</tbody>
</table>
```

Functions

- int bhv_init (bhv_t *bhv)
  
  *Entry point into the behavior.*

Variables

- bhv_handle_t * test

7.2.1 Detailed Description

Test behavior for loading sub-behaviors.

**Author:**

Kris Beevers (beevek@cs.rpi.edu)

**Version:**

`testsub.c,v 1.1 2003/08/19 18:44:13 beevek Exp`

Loads a sub-behavior (test), nothing else.

7.2.2 Function Documentation

7.2.2.1 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`
7.2.3 Variable Documentation

7.2.3.1 bhv_handle_f* test
7.3 arch/libbehavior/bhv_main.c File Reference

Provides a main program to behaviors, and handles communication with controlling programs (and the sequencer).

```c
#include <robot.h>
#include <robot/behavior.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>
#include <fcntl.h>
#include <stdlib.h>
#include <errno.h>
```

Include dependency graph for bhv_main.c:

![Dependency Graph](dependency_graph.png)

### Compounds

- struct `input_t`
- struct `inq_t`
- struct `output_t`
- struct `inq_list_t`

### Typedefs

- typedef `input_t` `input_t`
- typedef `output_t` `output_t`
- typedef `inq_t` `inq_t`

### Functions

- int `seq_init` (void)
  
  *Internal use only.*

- void `seq_cleanup` (void)
  
  *Internal use only.*

- int `net_init` (const char *ip, uint16_t port, robot_handle_t *handle)

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
Internal use only.

- **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).

- **int main** (int argc, char **argv)

Variables

- int errno
- int seq_qid
- int my_qid
- robot_handle_t bhv_robot_handle
- bhv_handle_t bhv_self
- const char * bhv_name
- input_t * inputs [256]
- output_t * outputs [256]
- inqlist_t inqs [256]

7.3.1 Detailed Description

Provides a main program to behaviors, and handles communication with controlling programs (and the sequencer).

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

bhv_main.c,v 1.17 2003/08/20 19:43:51 beevek Exp
7.3.2 Typedef Documentation

7.3.2.1 typedef struct _input_t input_t

7.3.2.2 typedef struct _inquiry_t inquiry_t

7.3.2.3 typedef struct _output_t output_t

7.3.3 Function Documentation

7.3.3.1 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

7.3.3.2 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!

7.3.3.3 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

7.3.3.4 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

7.3.3.5 void bhv_shutdown (int exitval)

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

Parameters:
exitval Value passed to exit

7.3.3.6 int main (int argc, char ** argv)

7.3.3.7 int net_init (const char * ip, uint16_t port, robot_handle_t * handle)

Internal use only.
Initialize a robot handle with ip address and port. Does not actually connect.

Parameters:
ip IP address string
port Remote port (usually ROBOT_DEFAULT_NET_PORT)
handle Pointer to handle to initialize

Returns:
< 0 on failure, >= 0 otherwise

7.3.3.8 void seq_cleanmp (void)

Internal use only.
De-initialize message queue communication with the sequencer and any loaded behaviors.
7.3.3.9 int seq_init (void)

Internal use only.
Initialize message queue communication with the sequencer.

Returns:
< 0 on failure, >= 0 otherwise

7.3.4 Variable Documentation

7.3.4.1 const char* bhv_name

the name of this behavior

7.3.4.2 robot_handle_t bhv_robot_handle

7.3.4.3 bhv_handle_t bhv_self

handle referring to ourself

7.3.4.4 int errno

7.3.4.5 input_t* inputs[256]

7.3.4.6 inq_list.t inqs[256]

7.3.4.7 int my_qid ()

my msg queue id

7.3.4.8 output_t* outputs[256]

7.3.4.9 int seq_qid ()

msg queue id for talking to seq
7.4 arch/seq/bhvctl.cpp File Reference

Start, stop and get status of reactive behaviors.

```c
#include <robot.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>
#include <fcntl.h>
```

Include dependency graph for bhvctl.cpp:

![Dependency Graph](dependency_graph.png)

Defines

- `#define printf(s...) do { printf(s); flush(stdout); } while(0)
- `#define perror(s) do { perror(s); flush(stderr); } while(0)

Functions

- `int seq_init (void)
  `Internal use only.''
- `void seq_cleanup (void)
  `Internal use only.''
- `void print_bhv_info (pid_t pid)
  `Get/print info about a behavior based on its pid.''
- `void show_blhs (void)
  `Print a list of running behaviors and info about each.''
- `void kill_bhv_pid (int pid)
  `Kill a behavior based on its pid.''
- `void kill_bhv_name (const char *name)
  `Kill a behavior based on its name.'"
• void start_bhv (const char *name)
  
  Start a behavior (and leave it running).

• int parse_command_line (int argc, char **argv)

  Parse command line for bhvctl.

• int main (int argc, char **argv)

Variables

• int seq_qid
• int my_qid
• char * bchelp

7.4.1 Detailed Description

Start, stop and get status of reactive behaviors.

Author:
  Kris Beevers (beeveks@gmail.com)

Version:
  bhvctl.cpp,v 1.4 2003/08/01 16:45:19 beevek Exp

For usage information, run bhvctl -help.

7.4.2 Define Documentation

7.4.2.1 #define perror(s) do { perror(s); fflush(stderr); } while(0)

7.4.2.2 #define printf(s...) do { printf(s); fflush(stdout); } while(0)

7.4.3 Function Documentation

7.4.3.1 void kill_bhv_name (const char * name)

Kill a behavior based on its name.

Parameters:
  name Name of behavior to kill

7.4.3.2 void kill_bhv_pid (int pid)

Kill a behavior based on its pid.

Parameters:
  pid Pid of behavior to kill
7.4.3.3  int main (int argc, char ** argv)

7.4.3.4  int parse_command_line (int argc, char ** argv)

Parse command line for bhvctl.

Returns:
< 0 on failure, >= 0 on success

7.4.3.5  void print_bhv_info (pid_t pid)

Get/print info about a behavior based on its pid.

7.4.3.6  void seq_cleanp (void)

Internal use only.
De-initialize message queue communication with the sequencer and any loaded behaviors.

7.4.3.7  int seq_init (void)

Internal use only.
Initialize message queue communication with the sequencer.

Returns:
< 0 on failure, >= 0 otherwise

7.4.3.8  void show_bhvs (void)

Print a list of running behaviors and info about each.

7.4.3.9  void start_bhv (const char * name)

Start a behavior (and leave it running).

Parameters:

name Name of behavior to start

7.4.4 Variable Documentation

7.4.4.1  char * bchelp

Initial value:

"\n"
"Normally, reactive behaviors are started and stopped as\n"
"part of a larger process that remains \"attached\" to them,\n"
"and they are automatically killed when that process quits.\n"
"Sometimes it is useful to simply start a behavior with no\n"
""owner"" that sticks around to control them. Similarly, it
"may be convenient to kill behaviors outside of their owner
"process (or if they were started in detached form). This
"program facilitates these actions.
"Beware that killing a behavior to which another process is
"attached may cause unexpected consequences in the process.
"since it will not be notified. Also keep in mind that any
"sub-behaviors" started by the behavior you kill will be
"stopped recursively.
"Commands:
"show list information about running behaviors
"kill <pid> kill the behavior with pid <pid> (gracefully)
"stop <name> stop all behaviors called <name> (gracefully)
"start <name> start the behavior called <name>

7.4.4.2 int my_qid

7.4.4.3 int seq_qid
7.5 arch/seq/sequencer.cpp File Reference

#include <robot/types.h>
#include <robot/util.h>
#include <robot/seq.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <list>

Include dependency graph for sequencer.cpp:

Compounds

- struct loaded_bhv_t
Defines

- `#define printf(...) do { printf(s); flush(stdout); } while(0)`
- `#define ROBOT_DEFAULT_BHV_PATH "./bvh"
  default behavior search paths`
- `#define PREFIX "[behavior]"`

Enumerations

- `enum { OPT_RUN_BG = (1 << 0), OPT_HAVE_LOG = (1 << 1) }
  command line options`

Functions

- `int init_search_path (char *path)`
  Parse a path argument.
- `int parse_command_line (int argc, char **argv)`
  Parse the metarobit command line.
- `int make_daemon (void)`
  Become a daemon.
- `void shutdown (int sig)`
  Signal handler, shutdown gracefully.
- `const char *find_behavior (const char *name, const char *cwd)`
  Search internal paths and cwd for a behavior called name.
- `int load_behavior (const char *path, const char *name, int qid, uid_t uid, char *args)`
  Actually start a new behavior if necessary.
- `int find_and_load (const char *name, const char *cwd, pid_t pid, uid_t uid, char *args)`
  Convenience; search for the behavior and then try to load it.
- `void send_err (pid_t pid, uint8_t cmd)`
- `void handle_load (void)`
  Load the requested behavior.
- `void handle_attach (void)`
  Give the requester a handle to an already-running behavior.
- `void handle_unload (void)`
  Unload the specified behavior.
- `void handle_list (void)`
  Send a list of pids of currently running behaviors back to the requester.
• void zombie (int sig)
• void myalarm (int sig)
  
Handle SIGALRM.

• int load_required_bhv (void)
  
Parse list of required behaviors and load them.

• int init_queue ()
  
Initialize the sequencer message queue.

• int main (int argc, char **argv)

Variables
• int errno
• char * seqlist
• std::list< loaded_bhv > behaviors
  currently running behaviors

• std::list< char * > search_dirs
  search paths

• char * reqd_bhv = 0
  required behaviors

• char * ip = 0
  ip address of robot

• char * port = 0
  port on robot to connect to

7.5.1 Define Documentation
7.5.1.1 #define PREFIX "[behavior]"

7.5.1.2 #define printf(s...) do { printf(s); fflush(stdout); } while(0)

7.5.1.3 #define ROBOT_DEFAULT_BHV_PATH "./.bhv"

default behavior search paths

7.5.2 Enumeration Type Documentation
7.5.2.1 anonymous enum

command line options
Enumeration values:

OPT_RUN_BG
OPT_HAVE_LOG

7.5.3 Function Documentation

7.5.3.1 int find_and_load (const char * name, const char * cwd, pid_t pid, uid_t uid, char * args)

Convenience; search for the behavior and then try to load it.

Returns:
< 0 on failure, >= 0 otherwise

7.5.3.2 const char* find_behavior (const char * name, const char * cwd)

Search internal paths and cwd for a behavior called name.

Parameters:
name Name of the behavior
cwd Working directory of calling process, will also be searched

Returns:
Full path of behavior if found, null otherwise

7.5.3.3 void handle_attach (void)

Give the requester a handle to an already-running behavior.
On error, send back an error message with errno.

7.5.3.4 void handle_list (void)

Send a list of pids of currently running behaviors back to the requester.

7.5.3.5 void handle_load (void)

Load the requested behavior.
Send bhv_handle_t information back to the requesting process. On an error, send back an error message with errno.

7.5.3.6 void handle_unload (void)

Unload the specified behavior.
Send back SEQ_UNLOAD on success, and an errno on failure.
7.5.3.7  int init_queue ()

Initialize the sequencer message queue.
Create the message queue. If it already exists, fail. Set its permissions to 00777 (all read/write).

Returns:
< 0 on failure, >= 0 otherwise

7.5.3.8  int init_search_path (char * path)

Parse a path argument.

Parameters:
    path The argument to -p

Returns:
< 0 on failure, number of paths to search otherwise

7.5.3.9  int load_behavior (const char * path, const char * name, int qid, uid_t uid, char * args)

Actually start a new behavior if necessary.

Parameters:
    path Path to the behavior
    name Name to pass to the behavior
    qid Msg queue of requesting process to notify
    uid Uid to run behavior as
    args Argument string to pass to behavior; if this exists, do not pass ip/port from our own cmdline

Returns:
< 0 on failure, >= 0 otherwise

7.5.3.10 int load_required_bhv (void)

Parse list of required behaviors and load them.

Returns:
< 0 on failure, >= 0 on success
7.5.3.11 int main (int argc, char ** argv)

7.5.3.12 int make_daemon (void)

Become a daemon.
Close open files if necessary, or point them to some other place.

Returns:
< 0 on failure, >= 0 on success

7.5.3.13 void myalarm (int sig)

Handle SIGALRM.
Do nothing, we just want the interruption.

7.5.3.14 int parse_command_line (int argc, char ** argv)

Parse the netrobot command line.
Print usage information if necessary.

Returns:
-1 on failure, 0 on success

Parameters:
argc main’s argc
argv main’s argv

7.5.3.15 void send_err (pid_t pid, uint8_t cmd) [inline]

7.5.3.16 void shutdown (int sig)

Signal handler, shutdown gracefully.

7.5.3.17 void zombie (int sig)

7.5.4 Variable Documentation

7.5.4.1 std::list<loaded_bhv_t> behaviors

currently running behaviors

7.5.4.2 int errno

7.5.4.3 char* ip = 0

ip address of robot
7.5.4.4 char* port = 0
port on robot to connect to

7.5.4.5 char* reqd_bhv = 0
required behaviors

7.5.4.6 std::list<char*> search_dirs
search paths

7.5.4.7 char* seqhelp

Initial value:

"\n"
"-d run as daemon\n"
"-p<path> specify search path for behaviors; of the form:\n"
" /path/a:/path/b:/path/c:/etc/\n"
" by default, path is just "ROBOT_DEFAULT_BHV_PATH \n"
"-b <bhvs> specify a comma-separated list of behaviors to load\n"
" automatically; they will not be stopped until the\n"
" sequencer exits. if any cannot be loaded, the sequencer\n"
" will fail and quit. e.g.:\n"
" emergency-stop,wall-follow,...\n"
"-i <ip> remote ip address of robot for behaviors to connect to;\n"
" this is completely optional, and if it is not specified,\n"
" behaviors will assume they are running on the robot\n"
"-o <port> port on remote robot to connect to (if ip is specified)\n"
"-l <file> print to this file instead of stdout\n"
7.6  driver/devices.c File Reference

Device info for all the different robot devices.
#include "robotdrv.h"
Include dependency graph for devices.c:

```c
#include "robotdrv.h"

// dependency graph

// device.c
```

Defines

- define NUM_DIRS (1 + NUM_DEVICE_DIRS)
- define INIT_DIR(n, c, ds, cs) if(init.single.dir(n, c, ds, cs) < 0) return -1

Functions

- int init_devfs (void)
  
  Initialize devfs /dev/robot filesystem entries.
- void cleanup_devfs (void)
  
  Destroy /dev/robot filesystem.

Variables

- robot_dev_t robot_devices [NUM_DEVICES]

7.6.1 Detailed Description

Device info for all the different robot devices.

Author:
  Kris Beevers (beevek@cs.rpi.edu)

Version:
  devices.c, v 1.12 2003/08/01 15:30:43 beevek Exp

A word on device types: The stream, change and current types are essentially just for applications to get data from. They may only be opened for writing by one process (which should be interp), but can be opened for reading by any number of processes. The stream and change types both act the same, but interp sends data to them differently (it sends everything to stream, and only sends data to change when the data is different from the previously sent data).

The ctl type, on the other hand, may be opened for writing by any number of processes. Data sent to this device will be read by interp and forwarded (sometimes after some transformation) to the hardware.
A read or write on any of these devices will fail if the size of the buffer being read to or written from is not exactly what the device expects. Look in the documentation for what these values are.

7.6.2 Define Documentation

7.6.2.1 #define INIT_DIR(n, c, ds, cs) if(init_single_dir(n, c, ds, cs) < 0) return -1

7.6.2.2 #define NUM_DIRS (1 + NUM_DEVICE_DIRS)

7.6.3 Function Documentation

7.6.3.1 void cleanup_devfs (void)

Destroy /dev/robot filesystem.

Clean up and call devfs_unregister on all /dev/robot entries

Precondition:

init_devfs must have been called

See also:

init_devfs

7.6.3.2 int init_devfs (void)

Initialize devfs /dev/robot filesystem entries.

This actually creates all of the /dev/robot files and registers our driver’s system calls to handle data on them.

Returns:

< 0 on failure, >= 0 on success

See also:

cleanup_devfs

7.6.4 Variable Documentation

7.6.4.1 robot_dev_t robot_devices[NUM_DEVICES]

global list of device structures
7.7 driver/robotdrv.c File Reference

Robot kernel driver for /dev/robot.
#include "robotdrv.h"

Include dependency graph for robotdrv.c:

```
robotdrv.h
```

```
robotdrv.c
```

Functions

- **MODULE_AUTHOR** ("Kris Beevers(beevek@cs.rpi.edu)")
- **MODULE_DESCRIPTION** ("/dev/robot driver for RPI ARL robots")
- **MODULE_LICENSE** ("GPL")
- int **init_devfs** (void)
  
  Initialize devfs /dev/robot filesystem entries

- void **cleanup_devfs** (void)
  
  Destroy /dev/robot filesystem.

- void **cleanup_module** (void)
  
  Called by the kernel on module unload.

- int **init_module** (void)
  
  Called by the kernel on module load; essentially just calls init_devfs and does nothing else.

7.7.1 Detailed Description

Robot kernel driver for /dev/robot.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
robotdrv.c,v 1.8 2003/07/31 18:24:48 beevek Exp

Linux kernel driver to provide an interface for user-space programs to the robot sensor/motor hardware. Actually, just acts as a communication interface with interp.

It requires a recentish kernel (2.4.x or greater) because I'm not writing support for older kernels, since the robot uses a 2.4. It also requires devfs support in the kernel!
### 7.7.2 Function Documentation

#### 7.7.2.1 void cleanup_devfs (void)

Destroy /dev/robot filesystem.
Clean up and call devfs_unregister on all /dev/robot entries

**Precondition:**
init_devfs must have been called

**See also:**
init_devfs

#### 7.7.2.2 void cleanup_module (void)

Called by the kernel on module unload.

**See also:**
init_module

#### 7.7.2.3 int init_devfs (void)

Initialize devfs /dev/robot filesystem entries.
This actually creates all of the /dev/robot files and registers our driver’s system calls to handle data on them.

**Returns:**
< 0 on failure, >= 0 on success

**See also:**
cleanup_devfs

#### 7.7.2.4 int init_module (void)

Called by the kernel on module load; essentially just calls init_devfs and does nothing else.

**See also:**
cleanup_module init_devfs

**Returns:**
-1 on failure, 0 on success

#### 7.7.2.5 MODULE_AUTHOR ("Kris Beevers(beevek @cs.rpi.edu)"")

#### 7.7.2.6 MODULE_DESCRIPTION ("/dev/robot driver for RPI ARL robots")

#### 7.7.2.7 MODULE_LICENSE ("GPL")
7.8 driver/robotdrv.h File Reference

Robot driver global include file.
#include <robot/constants.h>
#include <robot/devfs.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/poll.h>
#include <linux/devfs_fs_kernel.h>

Include dependency graph for robotdrv.h:

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

Compounds

- struct robot_dev_t
- struct robot_info_t

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
Defines

- `#define STREAM 0x00`
- `#define CHANGE 0x01`
- `#define CURRENT 0x02`
- `#define CTL 0x03`
- `#define MAX_TYPE 0x03`

Variables

- `robot_dev_t robot_devices[]`
- `file_operations robot_fops`

7.8.1 Detailed Description

Robot driver global include file.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
`robotdrv.h,v 1.9 2003/07/31 22:30:03 bheeve Exp`

7.8.2 Define Documentation

7.8.2.1 `#define CHANGE 0x01`
7.8.2.2 `#define CTL 0x03`
7.8.2.3 `#define CURRENT 0x02`
7.8.2.4 `#define MAX_TYPE 0x03`
7.8.2.5 `#define STREAM 0x00`

7.8.3 Variable Documentation

7.8.3.1 `robot_dev_t robot_devices[] ()`
global list of device structures

7.8.3.2 `struct file_operations robot_fops ()`
point to our system calls so the kernel knows what to use
7.9 driver/syscalls.c File Reference

System calls (common to all the device types).
#include "robotdrv.h"

Include dependency graph for syscalls.c:

Functions

- int robot_open (struct inode *inode, struct file *filp)
  Implementation of the open system call.
- int robot_release (struct inode *inode, struct file *filp)
  Implementation of the release (close) system call.
- ssize_t robot_read (struct file *filp, char *buf, size_t count, loff_t *f_pos)
  Implementation of the read system call.
- ssize_t robot_write (struct file *filp, const char *buf, size_t count, loff_t *f_pos)
- unsigned int robot_poll (struct file *filp, poll_table *wait)
- int robot_ioctl (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
  Implementation of the ioctl system call

Variables

- file_operations robot_fops

7.9.1 Detailed Description

System calls (common to all the device types).

Author:
  Kris Bevers (beevek@cs.rpi.edu)

Version:
  syascalls.c,v 1.7 2003/08/01 15:30:43 beevek Exp

We only implement the open, release (close), read, write, and poll system calls.
7.9.2 Function Documentation

7.9.2.1 int robot_ioctl (struct inode * inode, struct file * filp, unsigned int cmd, unsigned long arg)

Implementation of the ioctl system call.

The robot driver supports locking and unlocking of ctl-type files via this interface. "Locking" a file makes it so only the owner of the lock can write to it.

Parameters:
arg Should be set to the "priority" of the lock/unlock command. The larger the value, the lower the priority. If a file is locked but another lock attempt is made by a higher priority process, this attempt will be granted and the original lock owner will no longer own the file. Similarly, a higher- priority unlock request will release the lock of a lower-priority owner.

Returns:
< 0 on failure, >= 0 on success

7.9.2.2 int robot_open (struct inode * inode, struct file * filp)

Implementation of the open system call.

This call only allows one writer at a type for CTL-type devices.

Returns:
< 0 (a negative errno) on failure, 0 on success

7.9.2.3 unsigned int robot_poll (struct file * filp, poll_table * wait)

7.9.2.4 ssize_t robot_read (struct file * filp, char * buf, size_t count, off_t * f_pos)

Implementation of the read system call.

This call blocks for STREAM, CHANGE and CTL types if no data is available, otherwise it returns immediately.

Parameters:
count must be exactly equal to the expected value!

Returns:
< 0 on failure, number of bytes read on success

7.9.2.5 int robot_release (struct inode * inode, struct file * filp)

Implementation of the release (close) system call.

Returns:
< 0 (a negative errno) on failure, 0 on success
7.9.2.6 ssize_t robot_write (struct file * filp, const char * buf, size_t count, loff_t * f_pos)

7.9.3 Variable Documentation

7.9.3.1 struct file_operations robot_fops

Initial value:

{  
  read: robot_read,
  write: robot_write,
  poll: robot_poll,
  open: robot_open,
  release: robot_release,
  ioctl: robot_ioctl,
}

point to our system calls so the kernel knows what to use
7.10 include/robot.h File Reference

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

```c
#include <robot/config.h>
#include <robot/consts.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>
```

Include dependency graph for robot.h:

---

This graph shows which files directly or indirectly include this file:
7.10.1 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
7.11 include/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>
Include dependency graph for behavior.h:

```
inttypes.h  sys/types.h  robot/config.h

robot/types.h

robot/seq.h  stdio.h

behavior.h
```

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

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

robot/types.h

behavior.h

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

test.c  testsub.c  bhv_main.c

behavior.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(* blv_inhibit_func_t)(void)
  
  Called whenever the behavior is about to be inhibited.

• typedef void(* blv_uninhibit_func_t)(void)
  
  Called whenever the behavior is about to be uninhibited.

• typedef int(* blv_main_func_t)(void)
  
  The main function of a behavior.

Enumerations

• enum blv_flags_t { NO_ROBOT_INIT = (1 << 31), QUEUE WHEN INHIBITED
  = (1 << 30) }

Functions

• int blv_init (blv_t *bhv)
  
  Entry point into the behavior.

• int blv_add_input (uint8_t key, blv_data_func_t ondata)
  
  Add an input key to accept input from.

• int blv_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 blv_input_pop (uint8_t key, void *buf, int size)
  
  Extract input data from an input key’s queue.

• void blv_shutdown (int exitval)
  
  Shut down gracefully. Never call exit! Always call this instead!

• void blv_block_forever (void)
  
  Block forever (handling EINTR).

Variables

• blv_handle_t blv_self
• const char * blv_name

7.11.1 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.

7.11.2 Define Documentation

7.11.2.1 #define bhv_error(s)

Value:

```c
void bhv_error(const char *s);
```

Print error message to sequencer’s log file with some nice formatting (our name, pid).

7.11.2.2 #define bhv_printf(s...)

Value:

```c
void bhv_printf(const char *s);
```

Print to sequencer’s log file with some nice formatting (our name, pid).

7.11.3 Typedef Documentation

7.11.3.1 typedef void(* bhv_cleanup_func_t)(void)

Called on behavior cleanup.

Should perform any necessary de-initialization of the behavior.

7.11.3.2 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_data_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
7.11.3.3  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.

7.11.3.4  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.

7.11.3.5  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.

7.11.4  Enumeration Type Documentation

7.11.4.1  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)

7.11.5  Function Documentation

7.11.5.1  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:
  \textit{key} Input key to accept
  
  \textit{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 \texttt{bhv\_input\_pop} with the correct key.

Returns:
  \begin{itemize}
    \item \textless 0 on failure, \textgreater 0 on success
  \end{itemize}

See also:
  \begin{itemize}
    \item \texttt{bhv\_input\_pop}
  \end{itemize}

\subsection{7.11.5.2 \texttt{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!

\subsection{7.11.5.3 \texttt{int bhv\_init (bhv\_t * bhv)}}

Entry point into the behavior.

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

Parameters:
  \begin{itemize}
    \item \texttt{bhv} Pointer to a \texttt{bhv\_t} struct to be initialized by the behavior
  \end{itemize}

Returns:
  \begin{itemize}
    \item \textless 0 on failure, \textgreater 0 on success
  \end{itemize}

\subsection{7.11.5.4 \texttt{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 \texttt{bhv\_add\_input}, the data is placed in a queue for that input and is accessible through this function.

Parameters:
  \begin{itemize}
    \item \texttt{key} Input key to check for queued data
    \item \texttt{buf} Buffer in which to place data if found
    \item \texttt{size} Size of buffer (in bytes); if waiting data is larger than size, an error is generated and ERANGE is set
  \end{itemize}

Returns:
  \begin{itemize}
    \item \textless 0 on failure, or the size of the data placed in \texttt{buf} on success; if no items are on the specified queue, -1 is returned and ENODATA is set
  \end{itemize}
7.11.5.5 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

7.11.5.6 void bhv_shutdown (int exitval)

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

Parameters:
  *exitval* Value passed to exit

7.11.6 Variable Documentation

7.11.6.1 const char* bhv_name

the name of this behavior

7.11.6.2 bhv_handle_t bhv_self

handle referring to ourself
7.12 include/robot/constants.h File Reference

Global constants for the robot, should be available to any and all robot software.
This graph shows which files directly or indirectly include this file:

![Graph showing include relationships]

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/ttyS/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
7.12.1 Detailed Description

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

Author:
Kris Beever (beevek@cs.rpi.edu)

Version:

Todo
Most of these values need to be appropriately set.

7.12.2 Define Documentation

7.12.2.1 #define ROBOT_AXLE_WIDTH 0.37

7.12.2.2 #define ROBOT_DEFAULT_IR_FREQ 4.0 /*! hertz */

7.12.2.3 #define ROBOT_DEFAULT_NET_PORT 10798

7.12.2.4 #define ROBOT_DEFAULT_NET_TIMEOUT 2000

milliseconds

7.12.2.5 #define ROBOT_DEFAULT_SONAR_FREQ 4.0 /*! hertz */

7.12.2.6 #define ROBOT_DEFAULT_VEL_V 0.2

m/s

7.12.2.7 #define ROBOT_DEFAULT_VEL_W 0.4

m/s

7.12.2.8 #define ROBOT_ENCODER_STEPS_PER_REV 4096

number of encoder steps in one wheel revolution

7.12.2.9 #define ROBOT_HW_TTY ”/dev/tts/1”

7.12.2.10 #define ROBOT_IR_MULTIPLIER 0

7.12.2.11 #define ROBOT_MAX_NAME_LEN 128

7.12.2.12 #define ROBOT_MAX_VEL_V 1.0

m/s
7.12.2.13 #define ROBOT_MAX_VEL_W 3.14
m/s

7.12.2.14 #define ROBOT_NUM_BUMP 6
must be <= 8

7.12.2.15 #define ROBOT_NUM_IR 10
must be <= 16

7.12.2.16 #define ROBOT_NUM_SONAR 10
must be <= 16

7.12.2.17 #define ROBOT_SONAR_MULTIPLIER 274.4e-6
multiply sonar_time_val, ir_voltage_val by these to get dist in meters

7.12.2.18 #define ROBOT_THRESH_IR 0.01
hertz

7.12.2.19 #define ROBOT_THRESH_ODOM_THETA 0.01
meters

7.12.2.20 #define ROBOT_THRESH_ODOM_X 0.01
meters

7.12.2.21 #define ROBOT_THRESH_ODOM_Y 0.01
meters

7.12.2.22 #define ROBOT_THRESH_ROTATE 0.03

7.12.2.23 #define ROBOT_THRESH_SONAR 0.01
hertz

7.12.2.24 #define ROBOT_THRESH_TRANSLATE 0.03

7.12.2.25 #define ROBOT_THRESH_VEL_V 0.001
m/s
#include/robot/constants.h File Reference

7.12.2.26  #define ROBOT_THRESHVEL_W 0.001

m/s

7.12.2.27  #define ROBOT_WHEEL_RADIUS 0.076
7.12.2.28  #define ROBOT_WHEEL_RADIUS_L ROBOT_WHEEL_RADIUS
7.12.2.29  #define ROBOT_WHEEL_RADIUS_R ROBOT_WHEEL_RADIUS
7.13 include/robot/devfs.h File Reference

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

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

Include dependency graph for devfs.h:

```
  robot/config.h
  robot/hwid.h
  robot/types.h
  inttypes.h
  robot/constants.h
  inttypes.h
  sys/types.h
```

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

```
  robot.h
  devfs.c
  devfs_net.c
  devfs_local.c
  sys/c
  robotdrv.h
  handle.h
  interp.h
  motors.c
  sensors.c
  devfs.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)
  ‘Initializes the devfs subsystem.

• void devfs_cleanup (void)
  ‘De-initializes the devfs subsystem.

• devfs_set_t * devfs_find (uint8_t hwid)
  ‘Returns the devfs_set_t for the specified hardware id in the current robot’s devfs/ds list.

• int devfs_get_data_size (uint8_t hwid, devfs_type_t type)
  ‘Returns 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.

7.13.1 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
7.13.2 Define Documentation

7.13.2.1 #define CTL_SIZE_BUMP 0
7.13.2.2 #define CTL_SIZE_ENCODER 0
7.13.2.3 #define CTL_SIZE_IR (sizeof(freq_req_val_t))
7.13.2.4 #define CTL_SIZE_ODOM (3 * sizeof(odom_val_t))
7.13.2.5 #define CTL_SIZE_PWM 0
7.13.2.6 #define CTL_SIZE_SONAR (sizeof(freq_req_val_t))
7.13.2.7 #define CTL_SIZE_VEL (2 * sizeof(vel_val_t))
7.13.2.8 #define DATA_SIZE_BUMP (sizeof(bump_val_t))
7.13.2.9 #define DATA_SIZE_ENCODER (2 * sizeof(encoder_val_t))
7.13.2.10 #define DATA_SIZE_IR (sizeof(ir_val_t))
7.13.2.11 #define DATA_SIZE_ODOM (3 * sizeof(odom_val_t))
7.13.2.12 #define DATA_SIZE_PWM (2 * sizeof(pwm_val_t))
7.13.2.13 #define DATA_SIZE_SONAR (sizeof(sonar_val_t))
7.13.2.14 #define DATA_SIZE_VEL (2 * sizeof(vel_val_t))
7.13.2.15 #define DEV_TYPE_CHANGE_STR ”change”
7.13.2.16 #define DEV_TYPE_CTL_STR ”ctl”
7.13.2.17 #define DEV_TYPE_CURRENT_STR ”current”
7.13.2.18 #define DEV_TYPE_STREAM_STR ”stream”
7.13.2.19 #define DEVFS_BUMP_STR ”robot/bump”
7.13.2.20 #define DEVFS_ENCODER_STR ”robot/encoder”
7.13.2.21 #define DEVFS_IOCGETLOCKOWNER 0xc303
7.13.2.22 #define DEVFS_IOCLOCKCTL 0xc301

ioctls for /dev/robot driver. Not really the proper way to do these but simplifies things significantly.
7.13.2.23  #define DEVFS_IOCUNLOCKCTL 0xc302
7.13.2.24  #define DEVFS_IR_STR "robot/ir"
7.13.2.25  #define DEVFS_ODOM_STR "robot/odom"
7.13.2.26  #define DEVFS_PWM_STR "robot/pwm"
7.13.2.27  #define DEVFS_ROOT_STR "robot"
7.13.2.28  #define DEVFS_SONAR_STR "robot/sonar"
7.13.2.29  #define DEVFS_VEL_STR "robot/vel"
7.13.2.30  #define NUM_DEVICE_DIRS (7 + ROBOT_NUM_SONAR + ROBOT_NUM_IR + ROBOT_NUM_BUMP)

**Warning:**
if new devices are added this MUST BE CHANGED

7.13.2.31  #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

### 7.13.3  Enumeration Type Documentation

#### 7.13.3.1  anonymous enum
devfs initialization options

**Enumeration values:**
- DEVFS_SERVER
- DEVFS_CLIENT
- DEVFS_RDONLY

#### 7.13.3.2  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
7.13.3.3 enum devs_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

7.13.3.4 enum devs_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

7.13.4 Function Documentation

7.13.4.1 void devs_cleanup (void)

De-initialize the devs subsystem.

7.13.4.2 devs_set_t* devs_find (uint8_t hwid)

Find the devs_set_t for the specified hardware id in the current robot’s devs_fds list.

Parameters:
  hwid A hardware id from hwid.h

Returns:
  A pointer to the devs_set_t matching hwid from the current robot’s devs_fds list
7.13.4.3  `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)

7.13.4.4  `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

7.13.4.5  `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

7.13.4.6  `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
7.13.4.7 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

7.13.4.8 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

7.13.4.9 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
7.13.4.10  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
### 7.14 include/robot/handle.h File Reference

**robot_handle_t** and associated functionality

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

Include dependency graph for handle.h:

![Dependency Graph](image.png)

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

![Dependency Graph](image.png)

#### Compounds

- `struct robot_handle_t`

#### Defines

- `#define handle_is_net(h) ((h) \rightarrow \text{sock} > 0 || (h) \rightarrow \text{sockaddr.sin.addr.s_addr})`
- `#define handle_is_loc(h) (handle_is_net(h))`
- `#define handle_is_connected(h) (handle_is_net(h) && (h) \rightarrow \text{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`
7.14.1 Detailed Description

`robot_handle_t` and associated functionality

**Author:**
Kris Beevers (beeve@cs.rpi.edu)

**Version:**
`handle.h,v 1.4 2003/07/18 01:13:11 beeve 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.

7.14.2 Define Documentation

7.14.2.1 `#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 *`.

7.14.2.2 `#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 *`.

7.14.2.3 `#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 *`.

7.14.3 Function Documentation

7.14.3.1 `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`
7.14.4 Variable Documentation

7.14.4.1 robot_handle_t* cur_robot (

pointer to the handle for the robot librobot is currently controlling
7.15 include/robot/hwid.h File Reference

Id numbers for hardware devices talking through serial.

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

Include dependency graph for hwid.h:

![Dependency Graph]

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

![Dependency Graph with Include Dependency]

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.

7.15.1 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.
7.15.2 Define Documentation

7.15.2.1 #define HW_IS_ALL_BUMP(d) (d == HW_ALL_BUMP)

7.15.2.2 #define HW_IS_ALL_IR(d) (d == HW_ALL_IR)

7.15.2.3 #define HW_IS_ALL_SONAR(d) (d == HW_ALL_SONAR)

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

7.15.2.5 #define HW_IS_ENCODER(d) (d == HW_ENCODER)

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

7.15.2.7 #define HW_IS_MC(d) (d == HW_MC)

7.15.2.8 #define HW_IS_MOTORS(d) (d == HW_MOTORS)

7.15.2.9 #define HW_IS_ODOM(d) (d == HW_ODOM)

7.15.2.10 #define HW_IS_PWM(d) (d == HW_PWM)

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

7.15.2.12 #define HW_IS_VEL(d) (d == HW_VEL)

7.15.2.13 #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).

7.15.3 Enumeration Type Documentation

7.15.3.1 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
7.16 include/robot/mclib.h File Reference

Definitions related to loading a motor control shared library.

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

Include dependency graph for mclib.h:

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

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.
7.16.1 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.

7.16.2 Typedef Documentation

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

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

7.16.2.3 typedef int(* mc_init_func_t)(uint32_t)

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

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

7.16.2.6 typedef void(* mc_shutdown_func_t)(void)

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

7.16.3 Function Documentation

7.16.3.1 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

7.16.3.2 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
7.17  include/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>
```

Include dependency graph for motors.h:

```
motors.h  inttypes.h  sys/types.h
          |          |          | robot/config.h
          |          |          |          | robot/types.h
          |          |          |          |          | robot/constants.h
```

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

```
motors.h  robot.h  motors.c  sys.c
```

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.
7.17.1 Detailed Description

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

**Author:**
Kris Bevers (beevek@cs.rpi.edu)

**Version:**
* motors.h 1.6 2003/08/01 15:30:43 beevek Exp*

**Todo**
FIXME: acceleration?

7.17.2 Function Documentation

7.17.2.1 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`

7.17.2.2 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`

7.17.2.3 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`
7.17.2.4  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:
  radi ans  Distance (in radians) to rotate; positive for counterclockwise rotation, negative for
clockwise rotation
  w  Rotational velocity (radians/sec)

Returns:
  < 0 on failure, >= 0 otherwise

7.17.2.5  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

7.17.2.6  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

7.17.2.7  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
7.17.2.8  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
7.18  include/robot/net.h File Reference

Methods for setting up network control of robots.
#include <robot/types.h>
#include <robot/handle.h>
Include dependency graph for net.h:

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

Compounds

- struct robot_net_msg_t

Enumerations

- enum {
  MSG_PING = 0xffffffff, MSG_INIT = 0x01, MSG_SHUTDOWN = 0x02, MSG_WAIT_FOREVER = 0x03,
  MSG_LOCK_CTL = 0xff, MSG_UNLOCK_CTL = 0xfe, MSG_GET_LOCK_OWNER = 0x0f
  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.

### 7.18.1 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!

### 7.18.2 Enumeration Type Documentation

#### 7.18.2.1 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

### 7.18.3 Function Documentation

#### 7.18.3.1 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.
7.18.3.2 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.

7.18.3.3 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
7.19 include/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>

Include dependency graph for sensors.h:

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

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.

### 7.19.1 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` v1.4 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);`

### 7.19.2 Function Documentation

#### 7.19.2.1 int robot_force_reset_all_sensors (void)

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

**Returns:**

\(< 0 \text{ on failure, } \geq 0 \text{ on success}\)

#### 7.19.2.2 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:**

\(< 0 \text{ on failure, } \geq 0 \text{ otherwise}\)
7.19.2.3 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

7.19.2.4 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

7.19.2.5 bump_val_t robot_get_bump (uint8_t hvid)

Get current bump toggle from a single bump sensor.

Parameters:
    hvid A bump sensor hardware id from hvid.h

Returns:
    Toggle value; 0 for off, nonzero for on

7.19.2.6 ir_val_t robot_get_ir (uint8_t hvid)

Get current range value from a single ir sensor.

Parameters:
    hvid An ir hardware id from hvid.h

Returns:
    An ir range; zero generally indicates a failure

7.19.2.7 sonar_val_t robot_get_sonar (uint8_t hvid)

Get current range value from a single sonar.

Parameters:
    hvid A sonar hardware id from hvid.h

Returns:
    A sonar range; zero generally indicates a failure
7.19.2.8 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

7.19.2.9 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

7.19.2.10 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

7.19.2.11 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

7.19.2.12 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
7.20 include/robot/seq.h File Reference

Remote control of reactive behaviors (through the sequencer).
#include <robot/types.h>

Include dependency graph for seq.h:

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

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 }

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
Functions

- `blv_handle_t * seq_load (const char *name)`
  Load a reactive behavior and initialize it.

- `blv_handle_t * seq_load_args (const char *name, const char *args)`
  Load a reactive behavior and pass it command line arguments.

- `blv_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.

- `blv_handle_t * seq_attach (const char *name)`
  Get the handle of an already-running behavior.

- `int seq_unload (blv_handle_t *bhv)`
  Unload a reactive behavior.

- `int seq_unload_all (void)`
  Unload all loaded behaviors.

- `int seq_inhibit (blv_handle_t *bhv)`
  Inhibit ("pause") a behavior.

- `int seq_inhibit_all (void)`
  Inhibit all loaded behaviors.

- `int seq_uninhibit (blv_handle_t *bhv)`
  Uninhibit ("unpause") a behavior.

- `int seq_uninhibit_all (void)`
  Uninhibit all loaded behaviors.

- `int seq_send (blv_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, blv_data_t *buf)`
  Wait for a behavior to send data to us, with input key key.

- `int seq_connect (const blv_connection_t *conn)`
  Connect an output of one behavior to an input of another.

- `int seq_disconnect (const blv_connection_t *conn)`
  Disconnect an output of one behavior from an input of another.

- `int seq_get_my_handle (blv_handle_t *handle)`
  Get a `blv_handle_t` representing the calling process, even if it isn't a behavior.
7.20.1 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

7.20.2 Define Documentation
7.20.2.1 #define MSGBUF_BASE_SZ (sizeof(uint8_t))
7.20.2.2 #define ROBOT_MAX_BHV_DATA_SIZE 512
maximum size of data that can be read from or written to a behavior

7.20.2.3 #define ROBOT_SEQ_QUEUE_KEY 1
message queue key for sequencer

7.20.3 Enumeration Type Documentation
7.20.3.1 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

7.20.4 Function Documentation
7.20.4.1 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 EACCESS 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

### 7.20.4.2 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

### 7.20.4.3 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

### 7.20.4.4 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:
   IMPORTANT: 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

7.20.4.5  *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

7.20.4.6  *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**

7.20.4.7  *int*  **seq_inhibit_all**  (*void*)

Inhibit all loaded behaviors.

Returns:

< 0 on failure, >= 0 otherwise

See also:

- **seq_inhibit**
7.20.4.8 blv_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

7.20.4.9 blv_handle_t* seq_load_args (const char * name, const char * args)

Load a reactive behavior and pass it command line arguments.
Same as seq_load, but passes command line arguments to the behavior. Note that all behaviors
that use libbehavior will treat the first command line 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

7.20.4.10 blv_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
7.20.4.11 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

7.20.4.12 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

7.20.4.13 int seq_uninhibit_all (void)

Uninhibit all loaded behaviors.

Returns:
- < 0 on failure, >= 0 otherwise

See also:
- seq_inhibit

7.20.4.14 int seqUnload (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 \texttt{seq\_load}

Returns:
    \(< 0 \) on failure, \( \geq 0 \) otherwise

See also:
    \texttt{seq\_load}

7.20.4.15 \hspace{1em} \textbf{int seq\_unload\_all} (void)

Unload all loaded behaviors.

Returns:
    \(< 0 \) on failure, \( \geq 0 \) otherwise

See also:
    \texttt{seq\_unload}
7.21  include/robot/sys.h File Reference

Robot system initialization and shutdown.

```c
#include <robot/types.h>
#include <robot/handle.h>
Include dependency graph for sys.h:
```

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

```
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` | `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.

### 7.21.1 Detailed Description

Robot system initialization and shutdown.

**Author:**
Kris Bevers (beevek@cs.rpi.edu)

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

### 7.21.2 Define Documentation

#### 7.21.2.1 `#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`

### 7.21.3 Enumeration Type Documentation

#### 7.21.3.1 `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
7.21.4 Function Documentation

7.21.4.1 robot_id typedef robot_get_id (void)

Get the unique id number of the current robot.

Todo
FIXME implement

7.21.4.2 const char* robot_get_name (void)

Get the name of the current robot.

Todo
FIXME implement

7.21.4.3 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 RL_NO_HANDLE_SIGS is not set, register signal handler for cleanly shutting down the robot
on receipt of SIGINT, SIGQUIT, SIGTERM or SIGSEGV
Initialize the devs subsystem (if RL_NO_DEVFS is not set)
Initialize sensors; if RL_NO_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.

7.21.4.4 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.
7.22 include/robot/time.h File Reference

Precision timing methods for librobot.
#include <robot/types.h>

Include dependency graph for time.h:

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

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().

7.22.1 Detailed Description

Precision timing methods for librobot.

Author:
  Kris Beevers (beevek@cs.rpi.edu)
It is useful to have very precise timing information for certain tasks on the robot. We'll do this timing using microseconds.

### 7.22.2 Define Documentation

#### 7.22.2.1 #define MSEC_PER_SEC 1000

#### 7.22.2.2 #define robot_time_to_float_ms(rt) ((float)((double)(rt) / (double)USEC_PER_MSEC))

convert robot time to a float representing milliseconds

#### 7.22.2.3 #define robot_time_to_float_sec(rt) ((float)((double)(rt) / (double)USEC_PER_SEC))

convert robot time to a float representing seconds

#### 7.22.2.4 #define USEC_PER_MSEC 1000

#### 7.22.2.5 #define USEC_PER_SEC 1000000

### 7.22.3 Function Documentation

#### 7.22.3.1 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

#### 7.22.3.2 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
7.22.3.3  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
7.23 include/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>
Include dependency graph for types.h:

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

7.23.1 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

7.23.2 Define Documentation

7.23.2.1 #define ROBOT_BYTE_ORDER _BIG_ENDIAN

powerpc

7.23.3 Typedef Documentation

7.23.3.1 typedef uint8_t bump_bitfield_val_t

7.23.3.2 typedef uint8_t bump_val_t

7.23.3.3 typedef int16_t encoder_val_t

7.23.3.4 typedef float freq_val_t

to ctl sonar/ir

7.23.3.5 typedef uint8_t hw_freq_val_t

freq sent to hw
7.23.3.6  typedef float ir_val_t
7.23.3.7  typedef uint16_t ir_voltage_val_t
7.23.3.8  typedef float odom_val_t
7.23.3.9  typedef pid_t owner_val_t
7.23.3.10 typedef int8_t pwm_val_t
7.23.3.11 typedef uint32_t robot_id_t
7.23.3.12 typedef uint64_t robot_time_us_t
7.23.3.13 typedef uint16_t sonar_time_val_t
7.23.3.14 typedef float sonar_val_t
7.23.3.15 typedef float vel_val_t
7.24 include/robot/util.h File Reference

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

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

Include dependency graph for util.h:

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

**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.

7.24.1 Detailed Description

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

Author:
Kris Beever (beevek@cs.rpi.edu)

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

7.24.2 Define Documentation

7.24.2.1 #define assert(a)

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

degrees to radians

7.24.2.3 #define robot_dprintf(s...)

7.24.2.4 #define robot_fix_byte_order(buf, item_sz, n)

7.24.2.5 #define robot_rad2deg(a) ((a) * 180 / M_PI)

radians to degrees

7.24.3 Function Documentation

7.24.3.1 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.
7.24.3.2  \texttt{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.
7.25 interp/data_lengths.c File Reference

Constants indicating the length of data to be sent/received from each type of device (through serial!).

#include "interp.h"
#include <string.h>

Include dependency graph for data_lengths.c:

![Dependency graph of data_lengths.c]

Functions

- void data_lengths_init (void)
  
  *Fill in the data_lengths array with proper values.*

Variables

- data_len_t data_lengths [HW_MAX]

7.25.1 Detailed Description

Constants indicating the length of data to be sent/received from each type of device (through serial!).

**Author:**

Kris Beavers (beever@cs.rpi.edu)

**Version:**

*data_lengths.c,v 1.11 2003/07/18 15:27:22 beever Exp*

This assumes that every device only ever needs to send data of one type (length), and receive data of only one type (length).

7.25.2 Function Documentation

7.25.2.1 void data_lengths_init (void)

*Fill in the data_lengths array with proper values.*

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.25.3 Variable Documentation

7.25.3.1 data_len_t data_lengths[HW_MAX]

data sizes (in bytes) for sending/receiving from HW
7.26 interp/freq.c File Reference

Manage sensor "firing frequencies" according to requests by processes.

```c
#include "interp.h"
#include <robot/util.h>
#include <robot/time.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for freq.c:

![Dependency Graph](image.png)

### Compounds

- `struct _freq_list_t`

### Defines

- `#define freq_to_hw_freq(f)`

### Typedefs

- `typedef _freq_list_t freq_list_t`

### Functions

- `int freq_init (void)
  
  Initialize our internal data related to maintaining proper frequencies.`

- `void freq_cleanup (void)
  
  Clean up the list of processes and their requested frequencies.`

- `int freq_set (uint8_t hwid, freq_req_val_t *req)
  
  Set the frequencies for a HWID based on a request from some robot-related process.`
Variables

- `serial_cmd.t out_cmd`

7.26.1 Detailed Description

Manage sensor "firing frequencies" according to requests by processes.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
`freq.c,v 1.6 2003/07/31 22:30:03 beevek Exp`

Here’s how this works:

We have a linked list (freq_head) of structs that contain information about frequencies requested by each process that is currently running. Any process that requests the sensors to fire at a certain frequency gets added to this list, and is only removed when either:

1. the process with the same id requests frequencies of all zero
2. a process sending id 0 requests frequencies from any hwid. In this case the list is cleared and the frequencies from id 0 are used unconditionally. In general this should only be used to set all frequencies to zero.

This is kind of clunky but is what works best because of the /dev/robot interface.

Whenever the list or elements of it change, we re-examine it. If any of the requested frequencies are larger than those previously sent to the hardware, we send updates to the affected hardware. If the maximum frequency for a sensor is less than what the sensor is currently firing at, we send it.

This way, we are always firing sensors at the highest necessary frequency and no higher. This saves power while guaranteeing every process will get sensor readings as fast as it needs (or faster).

7.26.2 Define Documentation

7.26.2.1 `#define freq_to_hw_freq(f)`

Value:

```c
(f) = 0 ? 0 :
   (hw_freq_val_t)(100.0 / (f))
```

convert from a frequency in HZ to one that the hardware expects (basically a number indicating how many ms to wait between firing)

7.26.3 Typedef Documentation

7.26.3.1 `typedef struct _freq_list_t freq_list_t`

linked list containing frequency request information from each running robot process
7.26.4  Function Documentation

7.26.4.1  void freq_cleanup (void)
Clean up the list of processes and their requested frequencies.
This does not set any frequencies on the hardware.

7.26.4.2  int freq_init (void)
Initialize our internal data related to maintaining proper frequencies.
This does not actually set any frequencies on the hardware.

Returns:
-1 on failure, 0 on success

7.26.4.3  int freq_set (uint8_t hwid, freq_req_val_t * req)
Set the frequencies for a HWID based on a request from some robot-related process.

Returns:  
< 0 on failure, >= 0 on success

Parameters:

hwid  A hardware id from hwid.h
req  A pointer to a frequency request.

7.26.5  Variable Documentation

7.26.5.1  serial_cmd_t out_cmd ()
outgoing to serial
7.27 interp/interp.c File Reference

Interpreter: low-level communication interface between microcontrollers and higher-level software.

```
#include "interp.h"
#include <robot/util.h>
#include <robot/sys.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/select.h>
#include <unistd.h>
#include <endian.h>
```

Include dependency graph for interp.c:

Defines

- `#define printf(s...) do { printf(s); flush(stdout); } while(0)`
- `#define MKUINT16(a, b) (((uint16_t)a << 8) | (uint16_t)b)`
- `#define read_or_fail(buf)`

Enumerations

```
enum {
    OPT_RUN_BG = (1 << 0), OPT_HIGH_PRIO = (1 << 1), OPT_DEVFS = (1 << 2),
    OPT_HW = (1 << 3),
    OPT_HAVE_LOG = (1 << 4)
}
```

Functions

- `void shutdown (int sig)`
  
  Signal handler, shut down gracefully.

- `void hw_dispatch (void)`
  
  Take a serial_cmd_t from the hardware and handle it properly.

- `int ctl_read_and_dispatch (int fd, devfs_set_t *owner)`

- `void loop_forever (void)`

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
Main program loop.

- int parse_command_line (int argc, char **argv)
  Parse the interp command line.

- int make_daemon (void)
  Become a daemon.

- int main (int argc, char **argv)

Variables

- char *ihelp
- mc_lib_t mc_lib
- serial_cmd_t in_cmd
- serial_cmd_t out_cmd

7.27.1 Detailed Description

Interpreter: low-level communication interface between microcontrollers and higher-level software.

Author:
Kris Beever (beevek@cs.rpi.edu)

Version:
interp.c,v 1.32 2003/08/19 15:14:56 beevek Exp

7.27.2 Define Documentation

7.27.2.1 #define MKUINT16(a, b) (((uint16_t)a << 8) | (uint16_t)b)
for constructing uint16_t from 2 bytes sent by hw

7.27.2.2 #define printf(s...) do { printf(s); flush(stdout); } while(0)

7.27.2.3 #define read_or_fail(buf)

Value:

do{
  if(read(fd, buf, sizeof(buf)) < 0) {
    return -i; 
  } while(0)

7.27.3 Enumeration Type Documentation

7.27.3.1 anonymous enum

cmd line options
Enumeration values:
  OPT_RUN_BG  run as a daemon
  OPT_HIGH_PRIO attempt to run with nice -20
  OPT_DEVFS  talk to /dev/robot
  OPT_HW  talk to hardware
  OPT_HAVE_LOG  write somewhere other than stdout

7.27.4  Function Documentation

7.27.4.1  int ctrl_read_and_dispatch (int fd, devfs_set_t * owner)
Determine which kind of command this is, read it, pack it up and send it to the right hw device
(or do the right thing internally if it isn’t meant for hw).

Returns:
  < 0 on failure, >= 0 on success

Parameters:
  fd  file descriptor to read from
    owner  where in /dev/robot fd came from

7.27.4.2  void hw_dispatch (void)
Take a serial_cmd_t from the hardware and handle it properly.
IMPORTANT: the byte order on the microcontrollers is little endian and the byte order on powerpc
is big endian. MKUINT16 handles this properly.

7.27.4.3  void loop_forever (void)
Main program loop.
Wait for data on hardware fds (serial) and /dev/robot fds, and handle it properly when it comes
in. Never returns.

7.27.4.4  int main (int argc, char ** argv)
7.27.4.5  int make_daemon (void)
Become a daemon.
Close open files if necessary, or point them to some other place.

Returns:
  < 0 on failure, >= 0 on success
7.27.4.6  int parse_command_line (int argc, char ** argv)

Parse the interp command line.
Print usage information if necessary.

Returns:
   -1 on failure, 0 on success

Parameters:
   argc  main's argc
   argv  main's argv

7.27.4.7  void shutdown (int sig)

Signal handler, shut down gracefully.

7.27.5   Variable Documentation

7.27.5.1  char * ihelp

Initial value:

"\n"
"-d  run as daemon\n"
"-p  high priority\n"
"-f  do not read/write from /dev/robot\n"
"-h  do not talk to hardware\n"
"-m <path> use the specified motor control library (default ..//mc//mc_pid.so)\n"
"-l <file> print to this file instead of stdout\n"

Todo
   FIXME: mc path

7.27.5.2  serial_cmd_t in_cmd

incoming from serial

7.27.5.3  mc_lib_t mc_lib

motor control library

7.27.5.4  serial_cmd_t out_cmd

outgoing to serial
7.28 interp/interp.h File Reference

Interpreter definitions.

```
#include <robot/constants.h>
#include <robot/hwid.h>
#include <robot/devfs.h>
#include <robot/mclib.h>
```

Include dependency graph for interp.h:

```
data_lengths.c  freq.c  interp.c  sensors.c  serial.c
```

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

```
robot/constants.h  inttypes.h  sys/types.h  robot/config.h
```

```
robot/hwid.h  robot/types.h
```

```
robot/devfs.h  robot/mclib.h
```

Compounds

- struct `data_len_t`
- struct `serial_cmd_t`

Defines

- `#define HW_TIMEOUT 2`
- `#define DEFAULT_MC_LIB "../mc/mc.pid.so"

Functions

- `void data_lengths_init (void)`
  
  _Fill in the data_lengths array with proper values._

- `int serial_init (const char *mcty)`
  
  _Initialize serial communications with hardware._

- `int serial_shutdown (void)`
  
  _Shutdown the hardware communications and the serial device they go through._
• int serial_read (serial_cmd_t *cmd)
  Read a hardware data packet from the serial device.

• int serial_write (const serial_cmd_t *cmd)
  Write a data packet to the serial device.

• int sens_init (void)
  Initialize internal sensor data storage.

• void sens_update_motors (encoder_val_t enc_left, encoder_val_t enc_right, pwm_val_t pwm_left, pwm_val_t pwm_right)
  Send motor-related data to /dev/robot.

• void sens_update_sonar (uint8_t hwid, sonar_time_val_t time)
  Send sonar-related data to /dev/robot.

• void sens_update_ir (uint8_t hwid, ir_voltage_val_t voltage)
  Send ir-related data to /dev/robot.

• void sens_update_bump (bump_bitfield_val_t bits)
  Send bump-related data to /dev/robot.

• int freq_init (void)
  Initialize our internal data related to maintaining proper frequencies.

• void freq_cleanup (void)
  Clean up the list of processes and their requested frequencies.

• int freq_set (uint8_t hwid, freq_req_val_t *req)
  Set the frequencies for a HWID based on a request from some robot-related process.

Variables

• mc_lib_t mc_lib
• data_len_t data_lengths [HW_MAX]
• odom_val_t odom [3]

7.28.1 Detailed Description

Interpreter definitions.

Author:
  Kris Beevers (beevek@cs.rpi.edu)

Version:
  interp.h,v 1.16 2003/07/18 01:13:12 beevek Exp
7.28.2 Define Documentation

7.28.2.1 #define DEFAULT_MC_LIB "/mc/mc_pid.so"

Todo

FIXME

7.28.2.2 #define HW_TIMEOUT 2

timeout for initializing and shutting down microcontroller (seconds)

7.28.3 Function Documentation

7.28.3.1 void data_lengths_init (void)

Fill in the data_lengths array with proper values.

7.28.3.2 void freq_cleanup (void)

Clean up the list of processes and their requested frequencies.
This does not set any frequencies on the hardware.

7.28.3.3 int freq_init (void)

Initialize our internal data related to maintaining proper frequencies.
This does not actually set any frequencies on the hardware.

Returns:

-1 on failure, 0 on success

7.28.3.4 int freq_set (uint8_t hwid, freq_req_val_t * req)

Set the frequencies for a HWID based on a request from some robot-related process.

Returns:

< 0 on failure, >= 0 on success

Parameters:

hwid A hardware id from hwid.h
req A pointer to a frequency request.

7.28.3.5 int sens_init (void)

Initialize internal sensor data storage.

Returns:

-1 on failure, 0 on success
7.28.3.6 void sens_update_bump (bump_bitfield_val_t bits)
Send bump-related data to /dev/robot.

Parameters:
- bits A bitfield indicating which bump sensors are toggled

7.28.3.7 void sens_update_ir (uint8_t hwid, ir_voltage_val_t voltage)
Send ir-related data to /dev/robot.

Parameters:
- hwid A ir hardware id from hwid.h
- voltage Voltage sent by the hardware

7.28.3.8 void sens_update_motors (encoder_val_t enc_left, encoder_val_t enc_right, pwm_val_t left_pwm, pwm_val_t right_pwm)
Send motor-related data to /dev/robot.
This includes encoder, pwm, velocity and odometry data. This function calls motor control library
functions to get the current values for velocity and odometry.

Parameters:
- enc_left encoder count for left motor
- enc_right encoder count for right motor
- left_pwm pwm count for left motor
- right_pwm pwm count for right motor

7.28.3.9 void sens_update_sonar (uint8_t hwid, sonar_time_val_t time)
Send sonar-related data to /dev/robot.

Parameters:
- hwid A sonar hardware id from hwid.h
- time Hardware-calculated time of a sonar ping

7.28.3.10 int serial_init (const char * mc_tty)
Initialize serial communications with hardware.

Opens the serial device and sends an initialize command to the hardware, and then waits for a
proper response. Sets errno appropriately on failure; in particular, sets EPROTO if the serial de-
vice initialized properly but the hardware sent back improper data in response to the initialization
command.

Returns:
- < 0 on failure, >= 0 on success

Parameters:
- mc_tty string path to the tty device to connect to
7.28.3.11  int serial_read (serial_cmd_t * cmd)

Read a hardware data packet from the serial device.
Sets errno appropriately; in particular, sets EBADF if the serial device has not been properly
initialized, and sets ENODATA if the timeout expires before data is available on the serial port.

Returns:
< 0 on failure, >= 0 on success

Parameters:
  cmd A pointer to a serial_cmd_t where the data will be placed

7.28.3.12  int serial_shutdown (void)

Shutdown the hardware communications and the serial device they go through.

Returns:
-1 on failure, 0 on success

7.28.3.13  int serial_write (const serial_cmd_t * cmd)

Write a data packet to the serial device.
Sets errno to EBADF if serial device not initialized.

Returns:
< 0 on failure, >= 0 on success

Parameters:
  cmd A pointer to a serial_cmd_t containing the data to send

7.28.4  Variable Documentation

7.28.4.1  data_len_t data_lengths[HW_MAX] ()

Data sizes (in bytes) for sending/receiving from HW

7.28.4.2  mc_lib_t mc_lib ()

Motor control library

7.28.4.3  odom_val_t odom[3] ()

Odometry information
7.29 interp/sensors.c File Reference

Sensor data management.

```c
#include "interp.h"
#include <robot/util.h>
#include <math.h>
#include <string.h>
```

Include dependency graph for sensors.c:

```
```

## Functions

- **int sens_init (void)**
  
  Initialize internal sensor data storage.

- **void sens_update_motors (encoder_val_t enc_left, encoder_val_t enc_right, pwm_val_t left_pwm, pwm_val_t right_pwm)**
  
  Send motor-related data to /dev/robot.

- **void sens_update_sonar (uint8_t hwid, sonar_time_val_t time)**
  
  Send sonar-related data to /dev/robot.

- **void sens_update_ir (uint8_t hwid, ir_voltage_val_t voltage)**
  
  Send ir-related data to /dev/robot.

- **void sens_update_bump (bump_bitfield_val_t bits)**
  
  Send bump-related data to /dev/robot.

## Variables

- **encoder_val_t enc [2]**
- **pwm_val_t pwm [2]**
- **odom_val_t odom [3]**
- **vel_val_t vel [2]**
- **sonar_val_t sonar [ROBOT_NUM_SONAR]**
• `ir_val_t ir [ROBOT_NUM_IR]`
• `bump_val_t bump [ROBOT_NUM_BUMP]`

### 7.29.1 Detailed Description

Sensor data management.

**Author:**
Kris Bevers (beevek@cs.rpi.edu)

**Version:**
sensors.c,v 1.15 2003/07/18 15:27:22 beevek Exp

Data from sensors, including: velocities/encoder counts, pwm counts, odometry, sonar, ir, bump; and methods for updating the data throughout the system.

### 7.29.2 Function Documentation

#### 7.29.2.1 int sens_init (void)

Initialize internal sensor data storage.

**Returns:**
-1 on failure, 0 on success

#### 7.29.2.2 void sens_update_bump (bump_bitfield_val_t bits)

Send bump-related data to /dev/robot.

**Parameters:**
- `bits` A bitfield indicating which bump sensors are toggled

#### 7.29.2.3 void sens_update_ir (uint8_t hw_id, ir_voltage_val_t voltage)

Send ir-related data to /dev/robot.

**Parameters:**
- `hw_id` A ir hardware id from hwid.h
- `voltage` Voltage sent by the hardware

#### 7.29.2.4 void sens_update_motors (encoder_val_t enc_left, encoder_val_t enc_right, pwm_val_t left_pwm, pwm_val_t right_pwm)

Send motor-related data to /dev/robot.

This includes encoder, pwm, velocity and odometry data. This function calls motor control library functions to get the current values for velocity and odometry.
Parameters:
- `enc_left` encoder count for left motor
- `enc_right` encoder count for right motor
- `left_pwm` pwm count for left motor
- `right_pwm` pwm count for right motor

7.29.2.5 void sens_update_sonar (uint8_t hwid, sonar_time_val_t time)

Send sonar-related data to /dev/robot.

Parameters:
- `hwid` A sonar hardware id from hwid.h
- `time` Hardware-calculated time of a sonar ping

7.29.3 Variable Documentation

7.29.3.1 bump_val_t bump[ROBOT_NUM_BUMP]
bump sensor toggles

7.29.3.2 encoder_val_t enc[2]
encoder counts

7.29.3.3 ir_val_t ir[ROBOT_NUM_IR]
ir readings

7.29.3.4 odom_val_t odom[3]
odometry information

7.29.3.5 pwm_val_t pwm[2]
pwm counts

7.29.3.6 sonar_val_t sonar[ROBOT_NUM_SONAR]
sonar readings

7.29.3.7 vel_val_t vel[2]
velocity information
7.30 librobot/sensors.c File Reference

Implementation of methods from sensors.h.

```c
#include <robot/sensors.h>
#include <robot/devfs.h>
#include <robot/util.h>
#include <unistd.h>
#include <errno.h>
```

Include dependency graph for sensors.c:

![Dependency Graph](image)

**Functions**

- `int sensors_init (int set_default_freq)`
  
  *Internal use only.*

- `int sensors_shutdown (void)`
  
  *Internal use only.*

- `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)`

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
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.

**Variables**

- **int errno**

### 7.30.1 Detailed Description

Implementation of methods from **sensors.h**.

**Author:**
Kris Beevers (beevek@cs.rpi.edu)

**Version:**

### 7.30.2 Function Documentation

#### 7.30.2.1 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

#### 7.30.2.2 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
7.30.2.3 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

7.30.2.4 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

7.30.2.5 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

7.30.2.6 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

7.30.2.7 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
7.30.2.8  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

7.30.2.9  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

7.30.2.10 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

7.30.2.11 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

7.30.2.12 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

7.30.2.13 int sensors_init (int set_default_freq)
Internal use only.
Initialize the sensor subsystem.
Parameters:
    set_default_freq If nonzero, request default frequencies from all range sensors
Returns:
    < 0 on failure, >= 0 otherwise
7.30.2.14  int sensors_shutdown (void)

Internal use only.
Shut down the sensor subsystem. Remove all frequency requests from this process.

Returns:
\[< 0 \text{ on failure, } > = 0 \text{ otherwise}\]

7.30.3  Variable Documentation

7.30.3.1  int errno
7.31 interp/serial.c File Reference

Serial communication with microcontrollers controlling sensor devices and motors.

```c
#include <sys/types.h>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/select.h>
#include <assert.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/select.h>
#include <errno.h>
```

Include dependency graph for serial.c:

```
```

### Functions

- **int serial_init (const char *mc_type)**
  
  *Initialize serial communications with hardware.*

- **int serial_shutdown (void)**
  
  *Shutdown the hardware communications and the serial device they go through.*

- **int serial_read (serial_cmd_t *cmd)**
  
  *Read a hardware data packet from the serial device.*

- **int serial_write (const serial_cmd_t *cmd)**
  
  *Write a data packet to the serial device.*

#### 7.31.1 Detailed Description

Serial communication with microcontrollers controlling sensor devices and motors.
Author:
  Kris Bevers (beevek@cs.rpi.edu)

Version:
  serial.c,v 1.12 2003/07/18 01:13:12 beevek Exp

This code assumes a baud rate of 115200.

7.31.2 Function Documentation

7.31.2.1 int serial_init (const char * mc_tty)

Initialize serial communications with hardware.
Opens the serial device and sends an initialize command to the hardware, and then waits for a proper response. Sets errno appropriately on failure; in particular, sets EPROTO if the serial device initialized properly but the hardware sent back improper data in response to the initialization command.

Returns:
  < 0 on failure, >= 0 on success

Parameters:
  mc_tty string path to the tty device to connect to

7.31.2.2 int serial_read (serial_cmd_t * cmd)

Read a hardware data packet from the serial device.
Sets errno appropriately; in particular, sets EBADF if the serial device has not been properly initialized, and sets ENODATA if the timeout expires before data is available on the serial port.

Returns:
  < 0 on failure, >= 0 on success

Parameters:
  cmd A pointer to a serial_cmd_t where the data will be placed

7.31.2.3 int serial_shutdown (void)

Shutdown the hardware communications and the serial device they go through.

Returns:
  -1 on failure, 0 on success

7.31.2.4 int serial_write (const serial_cmd_t * cmd)

Write a data packet to the serial device.
Sets errno to EBADF if serial device not initialized.
Returns:
< 0 on failure, >= 0 on success

Parameters:
    *cmd* A pointer to a `serial_cmd_t` containing the data to send
7.32 librobot/devfs.c File Reference

/dev/robot management and communication.

```c
#include <robot/devfs.h>
#include <robot/handle.h>
#include <robot/util.h>
#include <errno.h>
```

Include dependency graph for devfs.c:

```
functions:
- int devfs_loc_init (int)
- void devfs_loc_cleanup (void)
- int devfs_loc_write (devfs_set_t *, const void *, int)
- int devfs_loc_read (devfs_set_t *, devfs_type_t, void *, int)
- int devfs_loc_wait_for_change (uint8_t *, int)
- int devfs_loc_lock_ctl (devfs_set_t *, uint32_t)
- int devfs_loc_unlock_ctl (devfs_set_t *, uint32_t)
- int devfs_loc_get_lock_owner (devfs_set_t *)
- int devfs_net_init (int)
- void devfs_net_cleanup (void)
- int devfs_net_write (devfs_set_t *, const void *, int)
- int devfs_net_read (devfs_set_t *, devfs_type_t, void *, int)
- int devfs_net_wait_for_change (uint8_t *, int)
- int devfs_net_lock_ctl (devfs_set_t *, uint32_t)
- int devfs_net_unlock_ctl (devfs_set_t *, uint32_t)
- int devfs_net_get_lock_owner (devfs_set_t *)
- int devfs_init (int flags)
  
  Initialize the devfs subsystem.
```

```
- void devfs_cleanup (void)
  De-initialize the devfs subsystem.
```

```
- 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.
```
7.32 librobot/devfs.c File Reference

- 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.

- 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.

Variables

- int errno

7.32.1 Detailed Description

/dev/robot management and communication.

Author:
  Kris Beevers (beevek@cs.rpi.edu)

Version:
  devfs.c,v 1.15 2003/07/31 22:30:03 beevek Exp

Supports talking to the /dev/robot entries both locally and via the network; this file mostly just decides which "real" function (local or networked) to call.
Actually implements just devfs_get_data_size and devfs_find.

7.32.2 Function Documentation

7.32.2.1 void devfs_cleanup (void)

De-initialize the devfs subsystem.
7.32.2.2  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

7.32.2.3  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)

7.32.2.4  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

7.32.2.5  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
7.32.2.6  void devfs_loc_cleanup (void)
7.32.2.7  int devfs_loc_get_lock_owner (devfs_set_t *)
7.32.2.8  int devfs_loc_init (int)
7.32.2.9  int devfs_loc_lock_ctl (devfs_set_t *, uint32_t)
7.32.2.10 int devfs_loc_read (devfs_set_t *, devfs_type_t, void *, int)
7.32.2.11 int devfs_loc_unlock_ctl (devfs_set_t *, uint32_t)
7.32.2.12 int devfs_loc_wait_for_change (uint8_t *, int)
7.32.2.13 int devfs_loc_write (devfs_set_t *, const void *, int)
7.32.2.14 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

7.32.2.15 void devfs_net_cleanup (void)
7.32.2.16 int devfs_net_get_lock_owner (devfs_set_t *)
7.32.2.17 int devfs_net_init (int)
7.32.2.18 int devfs_net_lock_ctl (devfs_set_t *, uint32_t)
7.32.2.19 int devfs_net_read (devfs_set_t *, devfs_type_t, void *, int)
7.32.2.20 int devfs_net_unlock_ctl (devfs_set_t *, uint32_t)
7.32.2.21 int devfs_net_wait_for_change (uint8_t *, int)
7.32.2.22 int devfs_net_write (devfs_set_t *, const void *, int)
7.32.2.23 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

7.32.2.24  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

7.32.2.25  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

7.32.2.26  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
7.32.3 Variable Documentation

7.32.3.1 int errno
7.33 librobot/devfs_local.c File Reference

devfs_* functions for talking to a local /dev/robot filesystem.

```c
#include <robot/devfs.h>
#include <robot/handle.h>
#include <robot/util.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/select.h>
#include <sys/ioctl.h>
```

Include dependency graph for devfs_local.c:

```

Visual Graph of Dependencies

```

Defines

- `#define DNFMT "/dev/%s/%s"
- `#define set_or_fail(f, dir, type, flags)
- `#define init_dir_fail(a, b, c, d)

Functions

- `int devfs_loc_init (int flags)
- `void devfs_loc_cleanup (void)
- `int devfs_loc_write (devfs_set_t *set, void *buf, int count)
- `int devfs_loc_read (devfs_set_t *set, devfs_type_t type, void *buf, int count)
- `int devfs_loc_wait_for_change (uint8_t *hwids, int count)
- `int devfs_loc_lock_ctl (devfs_set_t *set, uint32_t prio)
- `int devfs_loc_unlock_ctl (devfs_set_t *set, uint32_t prio)
- `int devfs_loc_get_lock_owner (devfs_set_t *set)

Variables

- `int errno

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.33 Detailed Description

devfs_* functions for talking to a local /dev/robot filesystem.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
devs_local.c,v 1.7 2003/08/01 15:30:43 beevek Exp

In other words, reads/writes from/to /dev/robot on the same computer as the program.

7.33.2 Define Documentation

7.33.2.1 #define DNFMT "'/dev/%s/%s"

7.33.2.2 #define init_dir_fail(a, b, c, d)

Value:

```c
do { 
    if(devfs_init_dir(a,b,c,d,flags) < 0) { 
        robot_dprintf("failed initializing %s\n", c); 
        return -1; 
    } 
} while(0)
```

7.33.2.3 #define set_or_fail(f, dir, type, flags)

Value:

```c
do { 
    if((f = open_dev(dir, type, flags)) < 0) 
        return f; 
} while(0);
```
7.33.3 Function Documentation

7.33.3.1 void devfs_loc_cleanup (void)

7.33.3.2 int devfs_loc_get_lock_owner (devfs_set_t * set)

7.33.3.3 int devfs_loc_init (int flags)

7.33.3.4 int devfs_loc_lock_ctl (devfs_set_t * set, uint32_t prio)

7.33.3.5 int devfs_loc_read (devfs_set_t * set, devfs_type_t type, void * buf, int count)

7.33.3.6 int devfs_loc_unlock_ctl (devfs_set_t * set, uint32_t prio)

7.33.3.7 int devfs_loc_wait_for_change (uint8_t * hwids, int count)

7.33.3.8 int devfs_loc_write (devfs_set_t * set, const void * buf, int count)

7.33.4 Variable Documentation

7.33.4.1 int errno
7.34 librobot/devfs_net.c File Reference

devfs_* functions for talking to a remote /dev/robot (through neterobotd), via a network.
#include <robot/devfs.h>
#include <robot/hwid.h>
#include <robot/handle.h>
#include <robot/util.h>
#include <robot/net.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

Include dependency graph for devfs_net.c:

```
    robot/constants.h  inttypes.h  sys/types.h  robot/config.h
    robot/handle.h     robot/types.h
    robot/devfs.h      robot/net.h
                      robot/net.h
                      netinet.h
                      robot/net.h

    devfs_net.c
```

Defines
- #define add_hwid(h) (cur_robot → devfs_fds[i++], hwid = (h))

Functions
- int net_connect_tcp (robot_handle_t *handle)
  Internal use only.
- int net_write_msg (int sock, robot_net_msg_t *msg)
  Internal use only.
- int net_read_msg (int sock, robot_net_msg_t *msg)
  Internal use only.
- int devfs_net_init (int flags)
- void devfs_net_cleanup (void)
- int devfs_net_write (devfs_set_t *set, const void *buf, int count)
- int devfs_net_read (devfs_set_t *set, devfs_type_t type, void *buf, int count)
7.34.1 Detailed Description

devfs_* functions for talking to a remote /dev/robot (through netrobotd), via a network.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
devfs_net.c, v 1.8 2003/07/31 22:30:04 beevek Exp

7.34.2 Define Documentation

7.34.3 Function Documentation

7.34.3.1 void devfs_net_cleanup (void)

7.34.3.2 int devfs_net_get_lock_owner (devfs_set_t * set)

7.34.3.3 int devfs_net_init (int flags)

7.34.3.4 int devfs_net_lock_ctl (devfs_set_t * set, uint32_t prio)

7.34.3.5 int devfs_net_read (devfs_set_t * set, devfs_type_t type, void * buf, int count)

7.34.3.6 int devfs_net_unlock_ctl (devfs_set_t * set, uint32_t prio)

7.34.3.7 int devfs_net_wait_for_change (uint8_t * hwids, int count)

7.34.3.8 int devfs_net_write (devfs_set_t * set, const void * buf, int count)

7.34.3.9 int net_connect_TCP (robot_handle_t * handle)

Internal use only.
Connect to a remote host using information stored in handle. Handle must have been properly initialized with net_init.

**Parameters:**

*handle* Pointer to handle with connection information

**Returns:**

< 0 on failure, >= 0 otherwise

### 7.34.3.10 int net_read_msg (int sock, robot_net_msg_t *msg)

Internal use only.
Read a single robot message from the specified socket.

**Parameters:**

*sock* Socket file descriptor to read from
*msg* Pointer to message buffer to read into

**Returns:**

< 0 on failure, >= 0 on success

### 7.34.3.11 int net_write_msg (int sock, robot_net_msg_t *msg)

Internal use only.
Write a single robot message to the specified socket.

**Parameters:**

*sock* Socket file descriptor to write to
*msg* Pointer to message to send

**Returns:**

< 0 on failure, >= 0 on success

### 7.34.4 Variable Documentation

#### 7.34.4.1 int errno

#### 7.34.4.2 const robot_net_msg_t msg_get_lock_owner

#### 7.34.4.3 const robot_net_msg_t msg_init

#### 7.34.4.4 const robot_net_msg_t msg_lock_ctl

#### 7.34.4.5 const robot_net_msg_t msg_shutdown

#### 7.34.4.6 const robot_net_msg_t msg_unlock_ctl

#### 7.34.4.7 const robot_net_msg_t msg_wait_change

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
### 7.35 librobot/handle.c File Reference

`robot_handle_t` related methods, etc.

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

Include dependency graph for handle.c:

![Dependency Graph](image)

**Functions**

- `int robot_set_handle (robot_handle_t *handle)`

  Set the current handle for all robot operations.

**Variables**

- `robot_handle_t local_robot`
- `robot_handle_t *cur_robot = &local_robot`

### 7.35.1 Detailed Description

`robot_handle_t` related methods, etc.

**Author:**
Kris Beovers (beevek@cs.rpi.edu)

**Version:**
`handle.c,v 1.2 2003/07/18 07:08:40 beevek Exp`

Also the location of `cur_robot`, the globally available pointer to the robot handle that librobot is currently controlling.

### 7.35.2 Function Documentation

#### 7.35.2.1 `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

### 7.35.3 Variable Documentation

#### 7.35.3.1 `robot_handle_t* cur_robot = &local_robot`

pointer to the handle for the robot librobot is currently controlling

#### 7.35.3.2 `robot_handle_t local_robot`

internal robot handle for programs that talk only to a single robot
7.36  librobot/mclib.c File Reference

Load/unload motor control library and initialize and de-initialize it.

```c
#include <robot/mclib.h>
#include <robot/util.h>
#include <string.h>
#include <dlfcn.h>
```

Include dependency graph for mclib.c:

![Dependency Graph](image)

**Defines**

- `#define load(v, s)`

**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.*

7.36.1  Detailed Description

Load/unload motor control library and initialize and de-initialize it.

**Author:**

Kris Beevers (beevek@cs.rpi.edu)

**Version:**

mclib.c,v 1.4 2003/07/18 07:08:40 beevek Exp

7.36.2  Define Documentation

7.36.2.1  `#define load(v, s)`

**Value:**
do { 
    v = disym(lib->handle, s); 
    if(dierror()) 
        return -1; 
} while(0)

load a single symbol from a shared library

7.36.3 Function Documentation

7.36.3.1 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

7.36.3.2 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
7.37  librobot/motors.c File Reference

Implementation of methods from motors.h.

```
#include <robot/motors.h>
#include <robot/devfs.h>
#include <robot/hwid.h>
#include <robot/util.h>
#include <math.h>
#include <errno.h>
```

Include dependency graph for motors.c:

![Dependency Graph]

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.*
Variables

- int errno

7.37.1 Detailed Description

Implementation of methods from motors.h.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
motors.c,v 1.20 2003/08/01 15:30:43 beevek Exp

7.37.2 Function Documentation

7.37.2.1 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

7.37.2.2 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

7.37.2.3 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
7.37.2.4  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

7.37.2.5  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

7.37.2.6  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

7.37.2.7  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
7.37.2.8 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

7.37.3 Variable Documentation

7.37.3.1 int errno
7.38 librobot/net.c File Reference

Methods for setting up network control of robots and sending and receiving data packets.

```c
#include <robot/types.h>
#include <robot/constants.h>
#include <robot/net.h>
#include <robot/handle.h>
#include <robot/util.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <signal.h>
#include <setjmp.h>
```

Include dependency graph for net.c:
Compounds

- struct _find_list_t

Typedefs

- typedef _find_list_t find_list_t

Functions

- int net_init (const char *ip, uint16_t port, robot_handle_t *handle)
  
  *Internal use only.*

- void net_shutdown (robot_handle_t *handle)
  
  *Internal use only.*

- int net_connect_tcp (robot_handle_t *handle)
  
  *Internal use only.*

- int net_write_msg (int sock, robot_net_msg_t *msg)
  
  *Internal use only.*

- int net_read_msg (int sock, robot_net_msg_t *msg)
  
  *Internal use only.*
• int robot_net_set_timeout (robot_handle_t *handle, int32_t ms)
  
  Set the network timeout for the specified handle, in milliseconds.

• 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.

• int robot_net_find (robot_handle_t **handles, uint32_t timeout_ms)
  
  Search for network-controllable robots on the local subnet.

Variables

• int errno
• const robot_net_msg_t msg_init = {0, MSG_INIT, 0, 0, 0}
• const robot_net_msg_t msg_shutdown = {0, MSG_SHUTDOWN, 0, 0, 0}
• const robot_net_msg_t msg_wait_change = {0, MSG_WAIT_FOR_CHANGE, 0, 0, 0}
• const robot_net_msg_t msg_lock_ctl = {0, MSG_LOCK_CTL, 0, 0, 0}
• const robot_net_msg_t msg_unlock_ctl = {0, MSG_UNLOCK_CTL, 0, 0, 0}
• const robot_net_msg_t msg_get_lock_owner = {0, MSG_GET_LOCK_OWNER, 0, 0, 0}
• const robot_net_msg_t msg_error = {0, MSG_ERROR, 0, 0, 0}
• const uint32_t msg_ping = MSG_PING

7.38.1 Detailed Description

Methods for setting up network control of robots and sending and receiving data packets.

Author:
  
  Kris Beevers (beevek@cs.rpi.edu)

Version:
  
  net.c,v 1.10 2003/07/31 22:30:04 beevek Exp

The netrobotd program should be running on the robot to be controlled over the network.

7.38.2 Typedef Documentation

7.38.2.1 typedef struct_find_list_t find_list_t

simple linked list of discovered robots

7.38.3 Function Documentation

7.38.3.1 int net_connect_tcp (robot_handle_t * handle)

Internal use only.

Connect to a remote host using information stored in handle. Handle must have been properly initialized with net_init.
Parameters:
   *handle Pointer to handle with connection information

Returns:
   < 0 on failure, >= 0 otherwise

7.38.3.2  int net_init (const char * ip, uint16_t port, robot_handle_t * handle)

Internal use only.
Initialize a robot handle with ip address and port. Does not actually connect.

Parameters:
   *ip IP address string
   *port Remote port (usually ROBOT_DEFAULT_NET_PORT)
   *handle Pointer to handle to initialize

Returns:
   < 0 on failure, >= 0 otherwise

7.38.3.3  int net_read_msg (int sock, robot_net_msg_t * msg)

Internal use only.
Read a single robot message from the specified socket.

Parameters:
   *sock Socket file descriptor to read from
   *msg Pointer to message buffer to read into

Returns:
   < 0 on failure, >= 0 on success

7.38.3.4  void net_shutdown (robot_handle_t * handle)

Internal use only.
Disconnect from the network and clean up network-related memory in handle.

Parameters:
   *handle Pointer to handle to clean up

7.38.3.5  int net_write_msg (int sock, robot_net_msg_t * msg)

Internal use only.
Write a single robot message to the specified socket.

Parameters:
   *sock Socket file descriptor to write to

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
**msg** Pointer to message to send

**Returns:**
- < 0 on failure, >= 0 on success

**7.38.3.6** 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.

**7.38.3.7** 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.

**7.38.3.8** 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
7.38 Variable Documentation

7.38.1 int errno

7.38.2 const robot_net_msg_t msg_error = {0, MSG_ERROR, 0, 0, 0}

7.38.3 const robot_net_msg_t msg_get_lock_owner = {0, MSG_GET_LOCK_OWNER, 0, 0, 0}

7.38.4 const robot_net_msg_t msg_init = {0, MSG_INIT, 0, 0, 0}

7.38.5 const robot_net_msg_t msg_lock_ctl = {0, MSG_LOCK_CTL, 0, 0, 0}

7.38.6 const uint32_t msg_ping = MSG_PING

7.38.7 const robot_net_msg_t msg_shutdown = {0, MSG_SHUTDOWN, 0, 0, 0}

7.38.8 const robot_net_msg_t msg_unlock_ctl = {0, MSG_UNLOCK_CTL, 0, 0, 0}

7.38.9 const robot_net_msg_t msg_wait_change = {0, MSG_WAIT_FOR_CHANGE, 0, 0, 0}
7.39  librobot/seq.c File Reference

Communication with the sequencer.

```c
#include <robot/types.h>
#include <robot/util.h>
#include <robot/handle.h>
#include <robot/seq.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <signal.h>
#include <errno.h>
```

Include dependency graph for seq.c:
Defines

- #define MAX_BHV 32

Functions

- int seq_init (void)
  
  *Internal use only.*

- void seq_cleanup (void)
  
  *Internal use only.*

- blv_handle_t * seq_load_args (const char *name, const char *args)
  
  *Load a reactive behavior and pass it command line arguments.*

- blv_handle_t * seq_load (const char *name)
  
  *Load a reactive behavior and initialize it.*

- blv_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 netrobot.*

- blv_handle_t * seq_attach (const char *name)
  
  *Get the handle of an already-running behavior.*

- int seq_unload (blv_handle_t *bhv)
  
  *Unload a reactive behavior.*

- int seq_unload_all (void)
  
  *Unload all loaded behaviors.*

- int seq_inhibit (blv_handle_t *bhv)
  
  *Inhibit ("pause") a behavior.*

- int seq_inhibit_all (void)
  
  *Inhibit all loaded behaviors.*

- int seq_uninhibit (blv_handle_t *bhv)
  
  *Uninhibit ("unpause") a behavior.*

- int seq_uninhibit_all (void)
  
  *Uninhibit all loaded behaviors.*

- int seq_send (blv_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, blv_data_t *buf)
  
  *Wait for a behavior to send data to us, with input key key.*

- int seq_connect (const blv_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.

Variables

- int errno
- int seq_qid = -1
- int my_qid = -1

7.39.1 Detailed Description

Communication with the sequencer.

**Author:**
Kris Beevers (beevek@cs.rpi.edu)

**Version:**
seq.c,v 1.16 2003/08/20 19:43:51 beevek Exp

The seq program must be running on the robot in order for this interface to properly function. Also, these functions are only meant for use on the robot itself. They will not function over the network.

This portion of librobot is NOT EVEN REMOTELY THREAD SAFE.

7.39.2 Define Documentation

7.39.2.1 #define MAX_BHV 32

7.39.3 Function Documentation

7.39.3.1 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
7.39.3.2 void seq_cleanup (void)

Internal use only.
De-initialize message queue communication with the sequencer and any loaded behaviors.

7.39.3.3 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

7.39.3.4 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

7.39.3.5 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:
IMPORTANT: 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
7.39.3.6  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

7.39.3.7  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

7.39.3.8  int seq_inhibit_all (void)

Inhibit all loaded behaviors.

Returns:
   < 0 on failure, >= 0 otherwise

See also:
   seq_inhibit

7.39.3.9  int seq_init (void)

Internal use only.
Initialize message queue communication with the sequencer.

Returns:
   < 0 on failure, >= 0 otherwise
7.39.3.10  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

7.39.3.11  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

7.39.3.12  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
7.39.3.13  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

7.39.3.14  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

7.39.3.15  int seq_uninhibit_all (void)

Uninhibit all loaded behaviors.

Returns:
  < 0 on failure, >= 0 otherwise

See also:
  seq_inhibit

7.39.3.16  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`

### 7.39.3.17 int seq_unload_all (void)

Unload all loaded behaviors.

Returns:

- `< 0 on failure, >= 0 otherwise`

See also:

- `seq_unload`

### 7.39.4 Variable Documentation

#### 7.39.4.1 int errno

#### 7.39.4.2 int my_qid = -1

My msg queue id

#### 7.39.4.3 int seq_qid = -1

Msg queue id for talking to seq
7.40  librobot/sys.c File Reference

System initialization and shutdown.

```
#include <robot/sys.h>
#include <robot/types.h>
#include <robot/handle.h>
#include <robot/devfs.h>
#include <robot/motors.h>
#include <robot/util.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
```

Include dependency graph for sys.c:

![Dependency Graph](image)

**Functions**

- `int sensors_init (int)`
  
  *Internal use only.*

- `int sensors_shutdown (void)`
  
  *Internal use only.*

- `int net_init (const char *, uint16_t, robot_handle_t *)`
  
  *Internal use only.*

- `void net_shutdown (robot_handle_t *)`
  
  *Internal use only.*

- `int seq_init (void)`
  
  *Internal use only.*
- void seq_cleanup (void)
  
  *Internal use only.*

- 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.*

**Variables**

- robot_handle_t local_robot

### 7.40.1 Detailed Description

System initialization and shutdown.

**Author:**

Kris Beevers (beevek@cs.rpi.edu)

**Version:**

`sys.c` v 1.11 2003/08/19 15:14:56 beevek Exp

Also provides a default signal handler to shutdown properly on SIGINT, SIGQUIT, SIGTERM and SIGSEGV.

### 7.40.2 Function Documentation

#### 7.40.2.1 int net_init (const char * ip, uint16_t port, robot_handle_t * handle)

*Internal use only.*

Initialize a robot handle with ip address and port. Does not actually connect.

**Parameters:**

- *ip*  IP address string
- *port* Remote port (usually ROBOT_DEFAULT_NET_PORT)
- *handle* Pointer to handle to initialize

**Returns:**

- < 0 on failure, >= 0 otherwise
7.40.2.2 void net_shutdown (robot_handle_t * handle)

Internal use only.
Disconnect from the network and clean up network-related memory in handle.

Parameters:
  
  handle Pointer to handle to clean up

7.40.2.3 robot_id_t robot_get_id (void)

Get the unique id number of the current robot.

Todo
  FIXME implement

7.40.2.4 const char* robot_get_name (void)

Get the name of the current robot.

Todo
  FIXME implement

7.40.2.5 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 RL_NO_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 RL_NO_DEVFS is not set)

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

If RL_USE_SEQENCER 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.
7.40.2.6 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.

7.40.2.7 int sensors_init (int set_default_freq)

Internal use only.
Initialize the sensor subsystem.

Parameters:
  set_default_freq If nonzero, request default frequencies from all range sensors

Returns:
  < 0 on failure, >= 0 otherwise

7.40.2.8 int sensors_shutdown (void)

Internal use only.
Shut down the sensor subsystem. Remove all frequency requests from this process.

Returns:
  < 0 on failure, >= 0 otherwise

7.40.2.9 void seq_cleanup (void)

Internal use only.
De-initialize message queue communication with the sequencer and any loaded behaviors.

7.40.2.10 int seq_init (void)

Internal use only.
Initialize message queue communication with the sequencer.

Returns:
  < 0 on failure, >= 0 otherwise

7.40.3 Variable Documentation

7.40.3.1 robot_handle_t local_robot ()

internal robot handle for programs that talk only to a single robot
7.41  librobot/time.c File Reference

us-precision timing functions for the robot. Implements stuff from time.h.
#include <robot/types.h>
#include <robot/time.h>
#include <robot/util.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/select.h>
Include dependency graph for time.c:

Functions

- void robot_time_to_timeval (const robot_time_us_t *rt, struct timeval *tv)
- void timeval_to_robot_time (const struct timeval *tv, robot_time_us_t *rt)
- 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().

7.41.1  Detailed Description

us-precision timing functions for the robot. Implements stuff from time.h.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
time.c,v 1.3 2003/07/18 07:08:41 beevek Exp

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.41.2 Function Documentation

7.41.2.1 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

7.41.2.2 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

7.41.2.3 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

7.41.2.4 void robot_time_to_timeval (const robot_time_us_t * rt, struct timeval * tv) [inline]

7.41.2.5 void timeval_to_robot_time (const struct timeval * tv, robot_time_us_t * rt) [inline]
7.42  librobot/util.c File Reference

Utility functions.
#include <robot/util.h>
#include <robot/types.h>
#include <math.h>

Include dependency graph for util.c:

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.

7.42.1  Detailed Description

Utility functions.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
util.c,v 1.4 2003/07/18 15:53:54 beevek Exp

7.42.2  Function Documentation

7.42.2.1  `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.
7.42.2.2 `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.
7.43 mc/mc.h File Reference

Motor control shared library interface and internals.

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

Include dependency graph for mc.h:

```
inttypes.h  sys/types.h  robot/config.h
robot/constants.h  robot/types.h
mc.h
```

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

```
mc_common.c  mc_pid.c  mc.h
```

Enumerations

- enum { LEFT = 0, RIGHT = 1, TRANS = 0, ROT = 1 }

Functions

- int mc_init (uint32_t flags)
  *Initialize the motor controller.*

- void mc_shutdown (void)
  *Shut down the controller and perform any necessary de-initialization.*

- int mc_start_frame (encoder_val_t left, encoder_val_t right)
  *Called at the start of each "motor control frame."*

- int mc_set_velocity (vel_val_t v, vel_val_t w)
  *Set the target velocities for the motor controller.*

- void mc_get_velocity (vel_val_t *v, vel_val_t *w)
  *Get the current velocity (based on encoder counts for the current frame).*

- void mc_set_odometry (odom_val_t *x, odom_val_t *y, odom_val_t *theta)
  *Update the odometry values (passed as arguments) based on the current frame’s encoder counts.*

- int mc_do_control (pwm_val_t *left_pwm, pwm_val_t *right_pwm)
Calculate PWM counts to send to the motors based on internal state (target velocity and the encoder counts for the current frame).

- int mc_init_common (uint32_t flags)
  Perform internal initialization common to most motor controllers.

Variables
- vel_val_t vel_correct [2]
- vel_val_t vel_current [2]
- vel_val_t vel_current_vw [2]
- encoder_val_t enc_correct [2]
- encoder_val_t enc_current [2]
- pwm_val_t pwm_current [2]
- float mc_dt

7.43.1 Detailed Description

Motor control shared library interface and internals.

Author:
Christopher Chiaverini

Version:
mc_h v 1.6 2003/08/01 19:39:26 bheevek Exp

This file defines the motor control class proposed by Kris Beevers. This class will be used to set the speeds of the drive motors by taking encoder data as input and outputting a pwm count.

Motor controllers are built as shared libraries to be loaded by the interpreter. They must all provide the functions specified below. See doc/motor_control_interface.txt for more information.

7.43.2 Enumeration Type Documentation

7.43.2.1 anonymous enum

Enumeration values:
- LEFT
- RIGHT
- TRANS
- ROT

7.43.3 Function Documentation

7.43.3.1 int mc_do_control (pwm_val_t * left_pwm, pwm_val_t * right_pwm)

Calculate PWM counts to send to the motors based on internal state (target velocity and the encoder counts for the current frame).
This is the primary function that will be re-implemented by new motor controllers.

**Parameters:**
- `left_pwm` Pointer to location to store PWM count for the left motor
- `right_pwm` Pointer to location to store PWM count for the right motor

**Returns:**
- `< 0` on failure, `>= 0` on success

### 7.43.3.2 void mc_get_velocity (vel_val_t * v, vel_val_t * w)

Get the current velocity (based on encoder counts for the current frame).
This function should use internal state set by the `mc_start_frame` function to calculate current translational and rotational velocities.

**Parameters:**
- `v` Pointer to a location to store the current translational velocity
- `w` Pointer to a location to store the current rotational velocity

### 7.43.3.3 int mc_init (uint32_t flags)

Initialize the motor controller.

**Parameters:**
- `flags` Currently unused

**Returns:**
- `< 0` on failure, `>= 0` otherwise

### 7.43.3.4 int mc_init_common (uint32_t flags)

Perform internal initialization common to most motor controllers.
In general, motor controllers should call this at the top of their own `mc_init` function.

**Parameters:**
- `flags` Currently ignored

**Returns:**
- `< 0` on failure, `>= 0` on success

### 7.43.3.5 void mc_set_odometry (odom_val_t * x, odom_val_t * y, odom_val_t * theta)

Update the odometry values (passed as arguments) based on the current frame’s encoder counts.
This function should ADD or SUBTRACT from the current values, but not overwrite them.

**Parameters:**
- `x` Pointer to current x-component of odometry
- `y` Pointer to current y-component of odometry
- `theta` Pointer to current theta-component of odometry
7.43.3.6 int mc_set_velocity (vel_val_t v, vel_val_t w)

Set the target velocities for the motor controller.
Called whenever a high-level program sets the desired translational and rotational velocities for
the robot. This should set internal variables as appropriate.

Parameters:
   v Target translational velocity
   w Target rotational velocity

Returns:
   < 0 on failure, >= 0 otherwise

7.43.3.7 void mc_shutdown (void)

Shut down the controller and perform any necessary de-initialization.

7.43.3.8 int mc_start_frame (encoder_val_t left, encoder_val_t right)

Called at the start of each "motor control frame."
In other words, this function is called before any other each time new encoder data becomes
available. It should perform any per-frame calculations and store their results internally. Other
functions (such as mc_do_control) use the state set by mc_start_frame to perform their computa-
tions.

Parameters:
   left Left motor encoder count
   right Right motor encoder count

Returns:
   < 0 on failure, >= 0 otherwise

7.43.4 Variable Documentation

7.43.4.1 encoder_val_t enc_correct[2]

target encoder counts

7.43.4.2 encoder_val_t enc_current[2]

current encoder counts

7.43.4.3 float mc_dlt

time since last frame

7.43.4.4 pwm_val_t pwm_current[2]

current pwm counts

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.43.4.5 vel_val_t vel_correct[2]

target velocities

7.43.4.6 vel_val_t vel_current[2]

current velocities

7.43.4.7 vel_val_t vel_current_vw[2]

current trans/rot velocities
7.44 mc/mc_common.c File Reference

Implements the functions defined in mc.h that are common to all motor controllers.
#include "mc.h"
#include <robot/time.h>
#include <math.h>

Include dependency graph for mc_common.c:

```
+-----+                  +------------------+
|     |                  |                  |
|     +------------------+ mc_common.c         |
|      +------------------+
|      |                   |
| robot/config.h           |
| robot/types.h            |
| robot/constants.h        |
| inttypes.h               |
| sys/types.h              |
| robot/time.h             |
| mc.h                     |
```

Defines

- `#define TWO_PI_L (2 * M_PI * ROBOT_WHEEL_RADIUS_L)
- `#define TWO_PI_R (2 * M_PI * ROBOT_WHEEL_RADIUS_R)
- `#define STEPS_MULT (ROBOT_ENCODER_STEPS_PER_REV * mc.dt)

Functions

- `int mc_start_frame (encoder_val_t left, encoder_val_t right)`
  
  *Called at the start of each "motor control frame."*

- `int mc_init_common (uint32_t flags)`

  *Perform internal initialization common to most motor controllers.*

- `void mc_shutdown (void)`

  *Shut down the controller and perform any necessary de-initialization.*

- `int mc_set_velocity (vel_val_t v, vel_val_t w)`

  *Set the target velocities for the motor controller.*

- `void mc_get_velocity (vel_val_t *v, vel_val_t *w)`

  *Get the current velocity (based on encoder counts for the current frame).*

- `void mc_set_odometry (odom_val_t *x, odom_val_t *y, odom_val_t *theta)`

  *Update the odometry values (passed as arguments) based on the current frame’s encoder counts.*
Variables

- vel_val_t vel_correct [2]
- vel_val_t vel_current [2]
- vel_val_t vel_current_vw [2]
- encoder_val_t enc_correct [2]
- encoder_val_t enc_current [2]
- pwm_val_t pwm_current [2]
- float mc_dt

7.44.1 Detailed Description

Implements the functions defined in mc.h that are common to all motor controllers.

Author:
  Christopher Chiaverini

Version:
  mc_common.c,v 1.8 2003/07/24 20:20:14 beevek Exp

7.44.2 Define Documentation

7.44.2.1 #define STEPS_MULT (ROBOT_ENCODER_STEPS_PER_REV * mc_dt)

7.44.2.2 #define TWO_PI_R_L (2 * M_PI * ROBOT_WHEEL_RADIUS_L)

7.44.2.3 #define TWO_PI_R_R (2 * M_PI * ROBOT_WHEEL_RADIUS_R)

7.44.3 Function Documentation

7.44.3.1 void mc_get_velocity (vel_val_t * v, vel_val_t * w)

Get the current velocity (based on encoder counts for the current frame).

This function should use internal state set by the mc_start_frame function to calculate current translational and rotational velocities.

Parameters:
  v Pointer to a location to store the current translational velocity
  w Pointer to a location to store the current rotational velocity

7.44.3.2 int mc_init_common (uint32_t flags)

Perform internal initialization common to most motor controllers.

In general, motor controllers should call this at the top of their own mc_init function.

Parameters:
  flags Currently ignored

Returns:
  < 0 on failure, >= 0 on success
7.44.3.3 void mc_set_odometry (odom_val_t * x, odom_val_t * y, odom_val_t * theta)

Update the odometry values (passed as arguments) based on the current frame’s encoder counts.
This function should ADD or SUBTRACT from the current values, but not overwrite them.

Parameters:
  x Pointer to current x-component of odometry
  y Pointer to current y-component of odometry
  theta Pointer to current theta-component of odometry

7.44.3.4 int mc_set_velocity (vel_val_t v, vel_val_t w)

Set the target velocities for the motor controller.
Called whenever a high-level program sets the desired translational and rotational velocities for
the robot. This should set internal variables as appropriate.

Parameters:
  v Target translational velocity
  w Target rotational velocity

Returns:
  < 0 on failure, >= 0 otherwise

7.44.3.5 void mc_shutdown (void)

Shut down the controller and perform any necessary de-initialization.

7.44.3.6 int mc_start_frame (encoder_val_t left, encoder_val_t right)

Called at the start of each "motor control frame."
In other words, this function is called before any other each time new encoder data becomes
available. It should perform any per-frame calculations and store their results internally. Other
functions (such as mc_do_control) use the state set by mc_start_frame to perform their computations.

Parameters:
  left Left motor encoder count
  right Right motor encoder count

Returns:
  < 0 on failure, >= 0 otherwise

7.44.4 Variable Documentation

7.44.4.1 encoder_val_t enc_correct[2]

target encoder counts
7.44.4.2 encoder_val_t enc_current[2]
current encoder counts

7.44.4.3 float mcflt
time since last frame

7.44.4.4 pwm_val_t pwm_current[2]
current pwm counts

7.44.4.5 vel_val_t vel_correct[2]
target velocities

7.44.4.6 vel_val_t vel_current[2]
current velocities

7.44.4.7 vel_val_t vel_current_vw[2]
current trans/rot velocities
7.45 mc/mc_pid.c File Reference

Implements the mc_do_control and mc_init functions for a PID type controller.
#include <math.h>
#include "mc.h"

Include dependency graph for mc_pid.c:

![Dependency Graph]

Defines

- #define NUM_SAMPLES 25
- #define clamp(v, i, a)

Functions

- void mc_get_left_kp (float *k)
- void mc_get_left_ki (float *k)
- void mc_get_left_kd (float *k)
- void mc_get_right_kp (float *k)
- void mc_get_right_ki (float *k)
- void mc_get_right_kd (float *k)
- void mc_set_left_kp (float k)
- void mc_set_left_ki (float k)
- void mc_set_left_kd (float k)
- void mc_set_right_kp (float k)
- void mc_set_right_ki (float k)
- void mc_set_right_kd (float k)
- int mc_init (uint32_t flags)

  Initialize the motor controller.

- int mc_do_control (pwm_val_t *left_pwm, pwm_val_t *right_pwm)

  Calculate PWM counts to send to the motors based on internal state (target velocity and the encoder counts for the current frame).
7.45.1 Detailed Description

Implements the mc.do.control and mc.init functions for a PID type controller.

**Author:**
Christopher Chiaverini

**Version:**
mc.pid.c,v 1.6 2003/07/18 07:08:45 beevek Exp

**Todo**
FIXME: remove functions to get/set gains

7.45.2 Define Documentation

7.45.2.1 \#define clamp(v, i, a)

**Value:**

```c
do { 
  if(v < i) 
    v = i; 
  else if(v > a) 
    v = a; 
}while(0)
```

7.45.2.2 \#define NUM_SAMPLES 25

Number of integration steps to store

7.45.3 Function Documentation

7.45.3.1 int mc.do.control (pwm.val.*left_pwm, pwm.val.*right_pwm)

Calculate PWM counts to send to the motors based on internal state (target velocity and the encoder counts for the current frame).

This is the primary function that will be re-implemented by new motor controllers.

**Parameters:**

- `left_pwm` Pointer to location to store PWM count for the left motor
- `right_pwm` Pointer to location to store PWM count for the right motor

**Returns:**

- `< 0 on failure, >= 0 on success`
7.45.3.2 void mc_get_left_kd (float * k)
7.45.3.3 void mc_get_left_ki (float * k)
7.45.3.4 void mc_get_left_kp (float * k)
7.45.3.5 void mc_get_right_kd (float * k)
7.45.3.6 void mc_get_right_ki (float * k)
7.45.3.7 void mc_get_right_kp (float * k)
7.45.3.8 int mc_init (uint32_t flags)

Initialize the motor controller.

Parameters:
flags Currently unused

Returns:
< 0 on failure, >= 0 otherwise

7.45.3.9 void mc_set_left_kd (float k)
7.45.3.10 void mc_set_left_ki (float k)
7.45.3.11 void mc_set_left_kp (float k)
7.45.3.12 void mc_set_right_kd (float k)
7.45.3.13 void mc_set_right_ki (float k)
7.45.3.14 void mc_set_right_kp (float k)
7.46 misc/logger/logger.cpp File Reference

Log data from /dev/robot entries, extensively.

```c
#include <robot.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/select.h>
#include <list>
#include "writer.h"
```

Include dependency graph for logger.cpp:

![Dependency Graph]

Enumerations

- enum { OPT_TYPE_STREAM = (1 << 0), OPT_TYPE_CHANGE = (1 << 1),
  OPT_TYPE_CTL = (1 << 2), OPT_RUN_BG = (1 << 3) }

  command line options

Functions

- int parse_hwid_list (char *arg, std::list<uint8_t> *hwids)
- void quit (int q)
  
  Properly handle signals and exit.

- int parse_command_line (int argc, char **argv)
  
  Parse the logger command line.

- void add_dir (devfs_set_t *s, fd_set *fds, int *max)
- void add_ctl (devfs_set_t *s, fd_set *fds, int *max)
- void init_poll_list (fd_set *fds, int *max)
  
  Initialize the list of file descriptors to read from and log.

- void read_and_log (devfs_set_t *set, devfs_type_t type)
  
  Read from a file with waiting data and log it.

- void log_forever (void)
  
  Loop forever, reading data and logging it when it becomes available.

- int main (int argc, char **argv)
Variables

- `char * lhelp`
- `log_writer_t robot_log`
  
  *output log*

- `log_entry_t entry`
  
  *buffer for log entries*

- `char * filename = 0`
- `int opt = 0`
- `std::list< uint8_t > hwids`
  
  *hwids we will log from*

7.46.1 Detailed Description

Log data from /dev/robot entries, extensively.

Author:

Kris Beever (beevek@cs.rpi.edu)

Version:

`logger.cpp` v 1.10 2003/08/19 15:14:57 beevek Exp

7.46.2 Enumeration Type Documentation

7.46.2.1 anonymous enum

command line options

Enumeration values:

- `OPT_TYPE_STREAM`
- `OPT_TYPE_CHANGE`
- `OPT_TYPE_CTL`
- `OPT_RUN_BG`

7.46.3 Function Documentation

7.46.3.1 `void add_ctl (devfs_set_t * s, fd_set * fds, int * max)` [inline]

7.46.3.2 `void add_dir (devfs_set_t * s, fd_set * fds, int * max)` [inline]

7.46.3.3 `void init_poll_list (fd_set * fds, int * max)`

Initialize the list of file descriptors to read from and log.

Parameters:

- `fds` A pointer to an fd_set to fill in
- `max` Will be set to the maximum valued file descriptor added to fds
7.46.3.4 void log_forever (void)

Loop forever, reading data and logging it when it becomes available.
This function never returns.

7.46.3.5 int main (int argc, char ** argv)

7.46.3.6 int parse_command_line (int argc, char ** argv)

Parse the logger command line.
Print usage information if necessary.

Returns:
-1 on failure, 0 on success

Parameters:
    argc main’s argc
    argv main’s argv

7.46.3.7 int parse_hwid_list (char * argv, std::list< uint8_t > * hwids)

Simple routine for parsing a string of the form blah,blah,blah,blah... and extracting hwids from it. Used as part of logger and logtool.

Returns:
-1 on failure, 0 on success

Parameters:
    argv string to parse
    hwids pointer to an STL list of hardware ids from hwid.h that will be filled in with the requested ids

7.46.3.8 void quit (int q)

Properly handle signals and exit.

7.46.3.9 void read_and_log (devfs_set_t * set, devfs_type_t type)

Read from a file with waiting data and log it.

Parameters:
    set A devfs_set_t from devfs_fds for the current robot
    type Which DEV_TYPE_* has waiting data

7.46.4 Variable Documentation

7.46.4.1 log_entry_t entry

buffer for log entries
7.46.4.2 char* filename = 0

7.46.4.3 std::list<uint8_t> hwids

hwids we will log from

7.46.4.4 char * lhelp

Initial value:

"\n"
"-d run as a daemon\n"
"\n"
"-s log DEV_TYPE_STREAM entries\n"
"-c log DEV_TYPE_CHANGE entries\n"
"-t log DEV_TYPE_CTL entries\n"
" if none of the above are specified, -ct is used by default\n"
"\n"
"-l <file> log to <file> (default " ROBOT_DEFAULT_LOG_FILE ")\n"
"\n"
"-h <list> log only the data for the specified HWIDs, comma separated (no spaces)\n"
" valid arguments for this are:\n"
"\n"
" sonar all sonar readings and ctl\n"
" ir all ir readings and ctl\n"
" bump all bump readings\n"
" vel all velocity readings and ctl\n"
" odom all odometry updates and ctl\n"
" encoder all encoder counts\n"
" pvm all pvm counts\n"
" {O-HW_MAX} (digits): specified HWID\n"
"\n"
" by default all data is logged\n"

7.46.4.5 int opt = 0

7.46.4.6 log_writer_t robot_log

output log
7.47 misc/logger/logger.h File Reference

Interface for reading from robot log files.
This graph shows which files directly or indirectly include this file:

![Dependency Graph]

7.47.1 Detailed Description

Interface for reading from robot log files.

**Author:**
Kris Beevers (beevek@cs.rpi.edu)

**Version:**
logger.h,v 1.7 2003/07/18 07:08:45 beevek Exp
7.48 misc/logger/logtool.cpp File Reference

Tool for printing information from robot log files.

#include <robot.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <list>
#include "logger.h"

Include dependency graph for logtool.cpp:

Enumerations

- enum { OPT_TYPE_STREAM = (1 << 0), OPT_TYPE_CHANGE = (1 << 1),
          OPT_TYPE_CTL = (1 << 2) }

command line options

Functions

- int parse_hwid_list (char *arg, std::list< uint8_t > *hwids)
- int parse_command_line (int argc, char **argv)

Parse the logtool command line.

- void print_entry_common (void)
- void print_ctl_entry (void)

Print stuff specific to a CTL-type entry.

- void print_data_entry (void)

Print stuff specific to a non-CTL-type entry (i.e. a STREAM or CHANGE entry).

- void quit (int q)

Handle signals gracefully and exit.

- int main (int argc, char **argv)

Variables

- char *lthelp
- log_reader: robot_log

log file to read from

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
• log_entry_t entry
  
  log entry buffer

• char * filename = 0
• robot_time_t begin = 0
• robot_time_t end = UINT64_MAX
• int opt = 0
• std::list< uint8_t > hwids

7.48.1 Detailed Description

Tool for printing information from robot log files.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
logtool.cpp,v 1.6 2003/07/18 07:08:45 beevek Exp

7.48.2 Enumeration Type Documentation

7.48.2.1 anonymous enum

command line options

Enumeration values:
  OPT_TYPE_STREAM
  OPT_TYPE_CHANGE
  OPT_TYPE_CTL

7.48.3 Function Documentation

7.48.3.1 int main (int argc, char ** argv)

7.48.3.2 int parse_command_line (int argc, char ** argv)

Parse the logtool command line.
Print usage information if necessary.

Returns:
-1 on failure, 0 on success

Parameters:
  argc main’s argc
  argv main’s argv
7.48.3.3 int parse_hwid_list (char * arg, std::list< uint8_t > * hwids)

Simple routine for parsing a string of the form blah,blah,blah,blah... and extracting hwids from it. Used as part of logger and logtool.

Returns:
-1 on failure, 0 on success

Parameters:
  *arg string to parse
  *hwids pointer to an STL list of hardware ids from hwid.h that will be filled in with the requested ids

7.48.3.4 void print_ctl_entry (void)

Print stuff specific to a CTL-type entry.

7.48.3.5 void print_data_entry (void)

Print stuff specific to a non-CTL-type entry (i.e. a STREAM or CHANGE entry).

7.48.3.6 void print_entry_common (void)

7.48.3.7 void quit (int q)

Handle signals gracefully and exit.

7.48.4 Variable Documentation

7.48.4.1 robot_time_us_t begin = 0

7.48.4.2 robot_time_us_t end = UINT64_MAX

7.48.4.3 log_entry_t entry

log entry buffer

7.48.4.4 char* filename = 0

7.48.4.5 std::list< uint8_t > hwids

7.48.4.6 char * lthelp

Initial value:

"\n"
"-s show DEV_TYPE_STREAM entries\n"
"-c show DEV_TYPE_CHANGE entries\n"
"-t show DEV_TYPE_CTL entries\n"
" if none of the above is specified, -sct is used by default\n"
"\n"  " -l <file> open <file> as the log file (default " ROBOT_DEFAULT_LOG_FILE ")\n"  "\n"  " -b <time> begin printing at <time> (microseconds since epoch)\n"  " -e <time> end printing at <time>\n"  "\n"  " -h <list> print only the data for the specified HWIDs, comma separated (no spaces)\n"  " valid arguments for this are:\n"  "\n"  " sonar all sonar readings and ctl\n"  " ir all ir readings and ctl\n"  " bump all bump readings\n"  " vel all velocity readings and ctl\n"  " odom all odometry updates and ctl\n"  " encoder all encoder counts\n"  " pwm all pwm counts\n"  " {0-HW_MAX} (digits): specified HWID\n"  "\n"  " by default all data is printed\n"

7.48.4.7 int opt = 0

7.48.4.8 log_reader_t robot_log

log file to read from
7.49 misc/logger/parse_hwid_list.cpp File Reference

Parse a string of the form a,b,c,d.

```cpp
#include "logger.h"
#include <list>
```

Include dependency graph for parse_hwid_list.cpp:

![Dependency Graph](attachment://dependency_graph.png)

### Functions

- int `parse_hwid_list` (char *arg, std::list< uint8_t > *hwids)

7.49.1 Detailed Description

Parse a string of the form a,b,c,d.

**Author:**

Kris Beevers (beevek@cs.rpi.edu)

**Version:**

`parse_hwid_list.cpp`, v 1.3 2003/07/18 15:27:23 beevek Exp

7.49.2 Function Documentation

7.49.2.1 int `parse_hwid_list` (char * arg, std::list< uint8_t > * hwids)

Simple routine for parsing a string of the form blah,blah,blah,blah... and extracting hwids from it. Used as part of logger and logtool.

**Returns:**

-1 on failure, 0 on success

**Parameters:**

- `arg` string to parse
- `hwids` pointer to an STL list of hardware ids from `hwid.h` that will be filled in with the requested ids
7.50 misc/logger/reader.cpp File Reference

Implementation of the log_reader class.
#include "logger.h"
#include <string.h>

Include dependency graph for reader.cpp:

```
    logger.h  string.h
    -----------
    reader.cpp
```

7.50.1 Detailed Description

Implementation of the log_reader class.

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
reader.cpp, v 1.5 2003/07/18 01:13:13 beevek Exp
7.51 misc/logger/writer.cpp File Reference

Implementation of log_writer_t class.

#include <robot/util.h>
#include <string.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include "writer.h"

Include dependency graph for writer.cpp:

7.51.1 Detailed Description

Implementation of log_writer_t class.

Author:
Kris Beavers (beevek@cs.rpi.edu)

Version:
writer.cpp,v 1.6 2003/07/18 16:35:47 beevek Exp
7.52 misc/logger/writer.h File Reference

Interface for log writer class (only used internally).

#include "logger.h"

Include dependency graph for writer.h:

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

Compounds

- class log_writer_t

7.52.1 Detailed Description

Interface for log writer class (only used internally).

Author:
Kris Beevers (beevek@cs.rpi.edu)

Version:
writer.h,v 1.3 2003/07/18 16:35:47 beevek Exp
Remote-control server to be run on the robot.

```cpp
#include <robot.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <unistd.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

Include dependency graph for netrobotd.cpp:

---

**Defines**

- `#define printf(s...) do { printf(s); fflush(stdout); } while(0)`

**Enumerations**

- `enum { OPT_RUN_BG = (1 << 0), OPT_NO_UDP = (1 << 1), OPT_HAVE_LOG = (1 << 2) }
  command line options`

**Functions**

- `int net_write_msg (int sock, robot_net_msg_t *msg)`
  *Internal use only.*

- `int net_read_msg (int sock, robot_net_msg_t *msg)`
  *Internal use only.*

- `int parse_command_line (int argc, char **argv)`
  *Parse the netrobotd command line.*

- `int make_daemon (void)`

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
Become a daemon.

- void **setup_sockaddr** (struct sockaddr.in *sa)
- int **listen_init** (void)
  
  Initialize listening sockets.

- int **do_robot_init** (void)
- int **net_write_error** (int err)
- int **tcp_handle_msg** (void)
- int **tcp_handle** (void)
- int **udp_handle** (void)
- void **zombie** (int sig)
- int **main** (int argc, char **argv)

**Variables**

- int **errno**
- const **robot_net_msg_t** msg_error
- **robot_id_t** robot_id
  
  id of local robot

- const char * **robot_name**
  
  name of local robot

- int **robot_name_len**
- char * **nrdhelp**
- int **client**
  
  socket of connected client

- uint8_t **msg_buf**[256]
  
  data buffer for msg

- **robot_net_msg_t** msg
  
  global message buffer

### 7.53.1 Detailed Description

Remote-control server to be run on the robot.

**Author:**

Kris Beever (beevek@cs.rpi.edu)

**Version:**


Server that runs on the robot accepting commands from remote devices. Translates their commands into devfs.reads and devfs.writes locally. Must be running for the network functionality in librobot to work properly.
7.53.2 Define Documentation

7.53.2.1 #define printf(...) do { printf(); flush(stdout); } while(0)

7.53.3 Enumeration Type Documentation

7.53.3.1 anonymous enum

command line options

Enumeration values:
    OPT_RUN_BG
    OPT_NO_UDP
    OPT_HAVE_LOG

7.53.4 Function Documentation

7.53.4.1 int do_robot_init (void)

7.53.4.2 int listen_init (void)

Initialize listening sockets.
Creates and initializes both a TCP listening socket and a UDP one.

Returns:
    < 0 on failure, >= 0 on success

7.53.4.3 int main (int argc, char ** argv)

7.53.4.4 int make_daemon (void)

Become a daemon.
Close open files if necessary, or point them to some other place.

Returns:
    < 0 on failure, >= 0 on success

7.53.4.5 int net_read_msg (int sock, robot_net_msg_t * msg)

Internal use only.
Read a single robot message from the specified socket.

Parameters:
    sock Socket file descriptor to read from
    msg Pointer to message buffer to read into

Returns:
    < 0 on failure, >= 0 on success
7.53.4.6 int net_write_error (int err)

7.53.4.7 int net_write_msg (int sock, robot_net_msg_t * msg)

Internal use only.
Write a single robot message to the specified socket.

Parameters:
  sock Socket file descriptor to write to
  msg Pointer to message to send

Returns:
  < 0 on failure, >= 0 on success

7.53.4.8 int parse_command_line (int argc, char ** argv)

Parse the netrobotd command line.
Print usage information if necessary.

Returns:
  -1 on failure, 0 on success

Parameters:
  argc main’s argc
  argv main’s argv

7.53.4.9 void setup_sockaddr (struct sockaddr_in * sa) [inline]

7.53.4.10 int tcp_handle (void)

7.53.4.11 int tcp_handle_msg (void)

7.53.4.12 int udp_handle (void)

7.53.4.13 void zombie (int sig)

7.53.5 Variable Documentation

7.53.5.1 int client

socket of connected client

7.53.5.2 int errno

7.53.5.3 robot_net_msg_t msg

global message buffer
7.53.5.4  uint8_t  msg_buf[256]

data buffer for msg

7.53.5.5  const robot_net_msg_t  msg_error

7.53.5.6  char * nrdhelp

Initial value:

"\n"
"-d" run as daemon\n"
"-p <port> listen on <port> (both udp and tcp)\n"
"-u do not listen on udp (for \"find\" requests)\n"
"-l <file> print to this file instead of stdout\n"

7.53.5.7  robot_id_t robot_id

id of local robot

7.53.5.8  const char* robot_name

name of local robot

7.53.5.9  int robot_name_len
7.54 misc/netrobot/nrtool.cpp File Reference

A utility for controlling robots over the network.

```cpp
#include <robot.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <list>
#include <unistd.h>
#include <fcntl.h>
#include <sys/select.h>
#include <errno.h>
#include <robot/config.h>
```

Include dependency graph for nrtool.cpp:

![Dependency Graph]

Compounds

- struct `handler_t`

  specify which function to call when a command is run

Functions

- int `setup_robot()`

  initialize a remote robot

- int `cleanup_robot()`

  shutdown a remote robot and free local memory

- `robot_handle_t *find_robot` (const char *name)

  find a robot with the specified name in the robots array

- int `h_find` (int argc, char **argv)
- int `h_unset` (int argc, char **argv)
- int `h_name` (int argc, char **argv)
- int `h_ip` (int argc, char **argv)
- int `h_sasf` (int argc, char **argv)
- int h_ssf (int argc, char **argv)
- int h_saif (int argc, char **argv)
- int h_sif (int argc, char **argv)
- int h_stop (int argc, char **argv)
- int h_sv (int argc, char **argv)
- int h_stv (int argc, char **argv)
- int h_srv (int argc, char **argv)
- int h_translate (int argc, char **argv)
- int h_rotate (int argc, char **argv)
- int h_so (int argc, char **argv)
- int h_gv (int argc, char **argv)
- int h_go (int argc, char **argv)
- int h_gs (int argc, char **argv)
- int h_gi (int argc, char **argv)
- int h_gb (int argc, char **argv)
- int h_status (int argc, char **argv)
- void remove_comments (char *cmd)
  
  strip the comments from a command line

- int run (char *cmd)
  
  Actually can run a series of commands separated by ;'s.

- void console (void)
  
  Repeatedly read commands and execute them.

- int main (int argc, char **argv)

Variables

- int errno
- char * nrhelp
- const char * version = "Id: nrtool.cpp,v 1.9 2003/07/18 15:27:23 beevek Exp $"
- char * rip = 0
- robot_handle t * robot
  current robot handle

- robot_handle t * robots
  available robots reported by h_find

- int nrobots = 0
  size of robots array

- char * err_args = "error: wrong arguments"
- char * err_robot = "error: no robot to talk to (use 'name' or 'ip')"
- handler t handlers []
  list of handlers for commands
7.54.1 Detailed Description

A utility for controlling robots over the network.

**Author:**

Kris Beavers (beevek@cs.rpi.edu)

**Version:**

*nrtool.cpp,* v 1.11 2003/08/11 21:21:46 beevek Exp

Run "nrtool -h" for usage information.
The h_* functions are handlers for each command.

7.54.2 Function Documentation

7.54.2.1 int cleanup_robot ()

shutdown a remote robot and free local memory

7.54.2.2 void console (void)

Repeatedly read commands and execute them.
The function doesn't exit until the user types quit or exit, or does ctrl-c.

7.54.2.3 robot_handle_t* find_robot (const char * name)

find a robot with the specified name in the robots array
7.54.2.4  int h_find (int argc, char ** argv)

7.54.2.5  int h_gb (int argc, char ** argv)

7.54.2.6  int h_gi (int argc, char ** argv)

7.54.2.7  int h_go (int argc, char ** argv)

7.54.2.8  int h_gs (int argc, char ** argv)

7.54.2.9  int h_gv (int argc, char ** argv)

7.54.2.10 int h_ip (int argc, char ** argv)

7.54.2.11 int h_name (int argc, char ** argv)

7.54.2.12 int h_rotate (int argc, char ** argv)

7.54.2.13 int h_saif (int argc, char ** argv)

7.54.2.14 int h_sasf (int argc, char ** argv)

7.54.2.15 int h_sif (int argc, char ** argv)

7.54.2.16 int h_so (int argc, char ** argv)

7.54.2.17 int h_srv (int argc, char ** argv)

7.54.2.18 int hssf (int argc, char ** argv)

7.54.2.19 int h_status (int argc, char ** argv)

7.54.2.20 int h_stop (int argc, char ** argv)

7.54.2.21 int h_stv (int argc, char ** argv)

7.54.2.22 int h_sv (int argc, char ** argv)

7.54.2.23 int h Translate (int argc, char ** argv)

7.54.2.24 int h_unset (int argc, char ** argv)

7.54.2.25 int main (int argc, char ** argv)

7.54.2.26 void remove_comments (char * cmd)

strip the comments from a command line

7.54.2.27 int run (char * cmd)

Actually can run a series of commands separated by ;'s.
Strips comments from the end of the line. Separating commands with semicolons makes it so you can (essentially) script nrtool.

**Returns:**

< 0 on failure (cmd not found), >= 0 on success

7.54.2.28 int setup_robot ()

initialize a remote robot

7.54.3 Variable Documentation

7.54.3.1 char* err_args = "error: wrong arguments"

7.54.3.2 char* err_robot = "error: no robot to talk to (use 'name' or 'ip')\n"

7.54.3.3 int errno

7.54.3.4 handler_t handlers[]

list of handlers for commands

7.54.3.5 char * nrhelp

7.54.3.6 int nrobots = 0

size of robots array

7.54.3.7 char* rip = 0

7.54.3.8 robot_handle_t* robot

current robot handle

7.54.3.9 robot_handle_t* robots

available robots reported by h_find

7.54.3.10 const char* version = "Id: nrtool.cpp,v 1.9 2003/07/18 15:27:23 beevek
Exp $"
7.55 misc/simulator/bumpsensor.h File Reference

#include <dolt.h>

Include dependency graph for bumpsensor.h:

![Dependency Graph]

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

![Dependency Graph]

Compounds

- class BumpSensor
#include <robot.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <iostream>
#include <dolt.h>
#include <algorithm>
#include "world.h"
#include "simrobot.h"

Include dependency graph for robot_simulator.cpp:
Namespaces

- namespace dolt

Compounds

- struct clientInfo

Defines

- #define printf(s...) do { printf(s); fflush(stdout); } while(0)

Typedefs

- typedef list<pair<clientInfo, SimRobot *>, RobotList

Enumerations

- enum { OPT_RUN_BG = (1 << 0), OPT_NO_UDP = (1 << 1), OPT_HAVE_LOG = (1 << 2) }
Functions

- int net_write_msg (int sock, robot_net_msg_t *msg)
  
  *Internal use only.*

- int net_read_msg (int sock, robot_net_msg_t *msg)
  
  *Internal use only.*

- void setup_sockaddr (struct sockaddr_in *sa)
- int listen_init (void)
- int net_write_error (RobotList::iterator i, int err)
- int set_velocity (RobotList::iterator i)
- int get_velocity (RobotList::iterator i)
- int set_odometry (RobotList::iterator i)
- int get_odometry (RobotList::iterator i)
- int set_all_sonar_frequencies (RobotList::iterator i)
- int get_all_sonar (RobotList::iterator i)
- int set_sonar_frequency (RobotList::iterator i, int sensor)
- int get_sonar (RobotList::iterator i, int sensor)
- int set_all_ir_frequencies (RobotList::iterator i)
- int get_all_ir (RobotList::iterator i)
- int set_ir_frequency (RobotList::iterator i, int sensor)
- int get_ir (RobotList::iterator i, int sensor)
- int get_all_bump (RobotList::iterator i)
- int get_bump (RobotList::iterator i, int sensor)
- int tcp_handle_msg (RobotList::iterator robot)
- void look_for_change (RobotList::iterator robot)
- int tcp_handle (int sock, SimRobot *rob)
- void do_physics ()
- void keyboardHandler (unsigned char key)
- int main (int argc, char **argv)

Variables

- int errno
- const robot_net_msg_t msg_error
- robot_id_t robot_id
- const char *robot_name
- int robot_name_len
- uint8_t msg_buf [256]
- timeval last present tv
- double elapsed
- double total_elapsed
- Graphics G
- mpProblem *P
- objectList workspace
- float acceleration
- bool displayPaths
- bool displaySonars
- bool displayIRs
• bool displayBumps

• bool waitForChange

• RobotList robots

• list< RobotList::iterator > toDelete

• list< pair< int, SimRobot * > > socks

7.56.1 Define Documentation

7.56.1.1 #define printf(...) do { printf(s); fflush(stdout); } while(0)

7.56.2 Tyedef Documentation

7.56.2.1 typedef list< pair<clientInfo, SimRobot *> > RobotList

7.56.3 Enumeration Type Documentation

7.56.3.1 anonymous enum

Enumeration values:

OPT_RUN_BG

OPT_NO_UDP

OPT_HAVE_LOG
7.56.4 Function Documentation

7.56.4.1 void do_phrase ()

7.56.4.2 int get_all_bump (RobotList::iterator i)

7.56.4.3 int get_all_ir (RobotList::iterator i)

7.56.4.4 int get_all_sonar (RobotList::iterator i)

7.56.4.5 int get_bump (RobotList::iterator i, int sensor)

7.56.4.6 int get_ir (RobotList::iterator i, int sensor)

7.56.4.7 int get_odometry (RobotList::iterator i)

7.56.4.8 int get_sonar (RobotList::iterator i, int sensor)

7.56.4.9 int get_velocity (RobotList::iterator i)

7.56.4.10 void keyboardHandler (unsigned char key)

7.56.4.11 int listen_init (void)

7.56.4.12 void look_for_change (RobotList::iterator robot)

7.56.4.13 int main (int argc, char ** argv)

7.56.4.14 int net_read_msg (int sock, robot_net_msg_t * msg)

Internal use only.
Read a single robot message from the specified socket.

Parameters:
sock Socket file descriptor to read from
msg Pointer to message buffer to read into

Returns:
< 0 on failure, >= 0 on success

7.56.4.15 int net_write_error (RobotList::iterator i, int err)

7.56.4.16 int net_write_msg (int sock, robot_net_msg_t * msg)

Internal use only.
Write a single robot message to the specified socket.

Parameters:
sock Socket file descriptor to write to
msg Pointer to message to send
Returns:
< 0 on failure, >= 0 on success
7.56.4.17 int set_all_ir_frequencies (RobotList::iterator i)
7.56.4.18 int set_all_sonar_frequencies (RobotList::iterator i)
7.56.4.19 int set_ir_frequency (RobotList::iterator i, int sensor)
7.56.4.20 int set_odometry (RobotList::iterator i)
7.56.4.21 int set_sonar_frequency (RobotList::iterator i, int sensor)
7.56.4.22 int set_velocity (RobotList::iterator i)
7.56.4.23 void setup_sockaddr (struct sockaddr_in * sa) [inline]
7.56.4.24 int tcp_handle (int sock, SimRobot * rob)
7.56.4.25 int tcp_handle_msg (RobotList::iterator robot)

7.56.5 Variable Documentation

7.56.5.1 float acceleration
7.56.5.2 bool displayBumps
7.56.5.3 bool displayIRs
7.56.5.4 bool displayPaths
7.56.5.5 bool displaySonars
7.56.5.6 double elapsed
7.56.5.7 int errno
7.56.5.8 Graphics G
7.56.5.9 uint8_t msg_buf[256]
7.56.5.10 const robot_net_msg_t msg_error
7.56.5.11 mpProblem* P
7.56.5.12 robot_id_t robot_id
7.56.5.13 const char* robot_name
7.56.5.14 int robot_name_len
7.56.5.15 RobotList robots
7.56.5.16 list< pair<int, SimRobot *> > socks
7.56.5.17 list<RobotList::iterator> toDelete
7.56.5.18 double total_elapsed
7.56.5.19 struct timeval last_present tv
7.56.5.20 bool waitForChange
7.56.5.21 objectList workspace
7.57 misc/simulator/sensorinfo.h File Reference

#include <dolt.h>

Include dependency graph for sensorinfo.h:

![Dependency Graph]

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

![Dependency Graph]

Compounds

- class SensorInfo
7.58 misc/simulator/simrobot.cpp File Reference

#include <filereader.h>
#include "world.h"
#include "simrobot.h"
#include <CGAL/intersections.h>

Include dependency graph for simrobot.cpp:

Namespaces

- namespace std

Functions

- double dist (const Point &p, const Point &q)

Variables

- mpProblem * P
- const long defaultSeed = 18846224
- long pmRandGenSeed = defaultSeed
- const long alpha = 16807
- const long m = 2147483647
- const long q = m / alpha
- const long r = m % alpha
- const long pmRandMax = m
7.58.1 Function Documentation

7.58.1.1 double dist (const Point & p, const Point & q) [inline]

7.58.2 Variable Documentation

7.58.2.1 const long alpha = 16807

7.58.2.2 const long defaultSeed = 18846224

7.58.2.3 const long m = 2147483647

7.58.2.4 mpProblem* P

7.58.2.5 long pmRandGenSeed = defaultSeed

7.58.2.6 const long pmRandMax = m

7.58.2.7 const long q = m / alpha

7.58.2.8 const long r = m % alpha
7.59 misc/simulator/simrobot.h File Reference

This file defines the Simrobot class.

```cpp
#include <list>
#include <string>
#include <iostream>
#include <dolt.h>
#include "sensorinfo.h"
#include "bumpsensor.h"
```

Include dependency graph for simrobot.h:

![Dependency Graph for simrobot.h]

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

![Dependency Graph for simrobot.h]

Compounds

- class **SimRobot**

  The *SimRobot* class is used to simulate a configurable differential drive robot.

7.59.1 Detailed Description

This file defines the Simrobot class.

**Author:**

Christopher Chiaverini (chiav@cs.rpi.edu)

**Version:**

```makefile
simrobot.h,v 1.24 2003/08/06 15:36:00 chiav Exp
```
7.60 misc/simulator/world.cpp File Reference

#include "world.h"
#include <iostream>
#include <fstream>
#include <filereader.h>
#include <math.h>

Include dependency graph for world.cpp:

Functions

- istream & operator>>(istream &in, Obstacle &obst)

7.60.1 Function Documentation

7.60.1.1 istream& operator>>(istream & in, Obstacle & obst)
7.61 misc/simulator/world.h File Reference

```
#include <list>
#include <string>
#include <iostream>
#include <dolt.h>
#include "simrobot.h"
#include "logger.h"
```

Include dependency graph for world.h:

![Dependency Graph for world.h](dependency_graph.png)

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

![Dependency Graph for Included Files](included_files_graph.png)

**Compounds**

- class mpProblem
- class Obstacle
- class World

**Variables**

- const unsigned int MAX_PROBLEM_NAME_LEN = 100

7.61.1 Variable Documentation

7.61.1.1 const unsigned int MAX_PROBLEM_NAME_LEN = 100
7.62 misc/tests/behavior.cpp File Reference

Test of the behavioral subsystem.
#include <robot.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

Include dependency graph for behavior.cpp:

```
+-----------------+
| robot.h | stdio.h | unistd.h | stdlib.h |
+-----------------+
| behavior.cpp    |
```

**Functions**

- int **main** (int argc, char **argv)

### 7.62.1 Detailed Description

Test of the behavioral subsystem.

**Author:**

Kris Beevers (beevek@cs.rpi.edu)

**Version:**

behavior.cpp, v 1.5 2003/08/20 18:28:27 beevek Exp

Load a behavior, sit around for a while and then quit (should automatically unload behavior).

### 7.62.2 Function Documentation

#### 7.62.2.1 int **main** (int *argc*, char **argv)

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.63 misc/tests/drive_in_circle.cpp File Reference

Simple test to make the robot drive in a circle.

```c
#include <robot.h>
#include <stdio.h>
#include <unistd.h>
```

Include dependency graph for drive_in_circle.cpp:

```
robot.h  stdio.h  unistd.h
```

Functions

- int `main` (int argc, char **argv)

7.63.1 Detailed Description

Simple test to make the robot drive in a circle.

Author:

Kris Beever (beevek@cs.rpi.edu)

Version:

`drive_in_circle.cpp`, v 1.5 2003/07/18 07:08:45 beevek Exp

7.63.2 Function Documentation

7.63.2.1 int `main` (int `argc`, char ** `argv`)
7.64  misc/tests/forward.cpp File Reference

Simple test to make the robot drive forward.

```c
#include <robot.h>
#include <stdio.h>
#include <unistd.h>
```

Include dependency graph for forward.cpp:

```
robot.h   stdio.h   unistd.h
```

Functions

- int **main** (int argc, char **argv)

7.64.1  Detailed Description

Simple test to make the robot drive forward.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

forward.cpp,v 1.4 2003/07/18 07:08:45 beevek Exp

7.64.2  Function Documentation

7.64.2.1  int main (int argc, char ** argv)
7.65 misc/tests/freq.cpp File Reference

Test setting sensor frequencies.
#include <robot.h>
#include <stdio.h>
#include <unistd.h>

Include dependency graph for freq.cpp:

```
robot.h  stdio.h  unistd.h
  freq.cpp
```

Functions

- int main (int argc, char **argv)

7.65.1 Detailed Description

Test setting sensor frequencies.

Author:
Kris Bevers (beevek@cs.rpi.edu)

Version:
freq.cpp,v 1.4 2003/07/18 07:08:45 beevek Exp

7.65.2 Function Documentation

7.65.2.1 int main (int argc, char ** argv)
7.66 misc/tests/lock.cpp File Reference

Test the locking of hardware device control.

```c
#include <robot.h>
#include <stdio.h>
#include <unistd.h>
```

Include dependency graph for lock.cpp:

```
robot.h  stdio.h  unistd.h

lock.cpp
```

Functions

- int **main** (int argc, char **argv)

7.66.1 Detailed Description

Test the locking of hardware device control.

**Author:**

Kris Beevers (beevek@cs.rpi.edu)

**Version:**

lock.cpp,v 1.1 2003/08/01 15:30:43 beevek Exp

7.66.2 Function Documentation

7.66.2.1 int main (int argc, char ** argv)
7.67 misc/tests/sonar.cpp File Reference

Test sonar sensors.

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

Include dependency graph for sonar.cpp:

Functions

- int `main` (int argc, char **argv)

7.67.1 Detailed Description

Test sonar sensors.

Author:

Kris Beevers (beevek@cs.rpi.edu)

Version:

```
sonar.cpp,v 1.4 2003/07/24 18:23:51 beevek Exp
```

7.67.2 Function Documentation

7.67.2.1 int `main` (int `argc`, char ** `argv`)
7.68  misc/tests/stop.cpp File Reference

Make the robot’s motors stop.

```c
#include <robot.h>
#include <stdio.h>
#include <unistd.h>
```

Include dependency graph for stop.cpp:

![Dependency Graph]

Functions

- int `main` (int argc, char **argv)

7.68.1  Detailed Description

Make the robot’s motors stop.

**Author:**

Kris Beavers (beevek@cs.rpi.edu)

**Version:**

```text
stop.cpp,v 1.4 2003/07/18 07:08:46 beevek Exp
```

7.68.2  Function Documentation

7.68.2.1  int `main` (int argc, char ** argv)
7.69  misc/tests/stop_sensors.cpp File Reference

Make the robot’s sensors stop firing.
#include <robot.h>
#include <stdio.h>
#include <unistd.h>

Include dependency graph for stop_sensors.cpp:

Functions

• int main (int argc, char **argv)

7.69.1  Detailed Description

Make the robot’s sensors stop firing.

Author:
  Kris Beevers (beevek@cs.rpi.edu)

Version:
  stop_sensors.cpp,v 1.3 2003/07/18 07:08:46 beevek Exp

7.69.2  Function Documentation

7.69.2.1  int main (int argc, char ** argv)
7.70  misc/tests/time.cpp File Reference

Test robot.time.us.t.
#include <robot.h>
#include <stdio.h>
#include <unistd.h>
Include dependency graph for time.cpp:

Functions

• int main (int argc, char **argv)

7.70.1 Detailed Description

Test robot.time.us.t.

Author:
Kris Beever (beevek@cs.rpi.edu)

Version:
time.cpp,v 1.2 2003/07/18 07:08:46 beevek Exp

7.70.2 Function Documentation

7.70.2.1 int main (int argc, char ** argv)
7.71 pda/mom/discoverydialog.cpp File Reference

#include "robot.h"
#include "discoverydialog.h"

Include dependency graph for discoverydialog.cpp:
7.72 pda/mom/discoverydialog.h File Reference

#include <list>
#include <qvbox.h>
#include <qlabel.h>
#include <qdialog.h>
#include <qlistbox.h>
#include <qpushbutton.h>
#include "robot.h"

Include dependency graph for discoverydialog.h:

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

Compounds

• class DiscoveryDialog
7.73  pda/mom/infodialog.cpp File Reference

#include "infodialog.h"

Include dependency graph for infodialog.cpp:
7.74 pda/mom/infodialog.h File Reference

#include <qdialog.h>
#include <qvbox.h>
#include <qlabel.h>
#include <qprogressbar.h>
#include "robot.h"
#include "robotinfo.h"

Include dependency graph for infodialog.h:

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

Compounds

- class InfoDialog
7.75  pda/mom/ipinputdialog.cpp File Reference

#include <fstream>
#include <string>
#include "ipinputdialog.h"

Include dependency graph for ipinputdialog.cpp:

```
#include <fstream>
#include <string>
#include "ipinputdialog.h"
```

![Dependency Graph Image]

---

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.76 pda/mom/ipinputdialog.h File Reference

#include <qvbox.h>
#include <qlabel.h>
#include <qdialog.h>
#include <qcombobox.h>
#include <qlineedit.h>

Include dependency graph for ipinputdialog.h:

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

Compounds

- class IpInputDialog
7.77 pda/mom/irdialog.cpp File Reference

#include "irdialog.h"
#include "simulator_constants.h"

Include dependency graph for irdialog.cpp:
#include <qdialog.h>
#include <qvbox.h>
#include <qcanvas.h>
#include <qpushbutton.h>
#include "robot.h"
#include "robotinfo.h"

Include dependency graph for irdialog.h:

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

Compounds

- class IRDialog
7.79 pda/mom/mom.cpp File Reference

#include <qpe/qpeapplication.h>
#include "momwindow.h"

Include dependency graph for mom.cpp:

Functions

- int main (int argc, char **argv)

7.79.1 Function Documentation

7.79.1.1 int main (int argc, char ** argv)
#include <qevent.h>
#include <qcanvas.h>
#include "momcanvasview.h"

Include dependency graph for momcanvasview.cpp:
7.81 pda/mom/momcanvasview.h File Reference

#include <qcanvas.h>
#include <qevent.h>

Include dependency graph for momcanvasview.h:

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

Compounds

- class MomCanvasView
#include <stdlib.h>
#include <string>
#include <iostream>
#include <fstream>
#include <qmainwindow.h>
#include <qapplication.h>
#include <qmessagebox.h>
#include <qevent.h>
#include <qcanvas.h>
#include <qcolor.h>
#include <qmenubar.h>
#include <qvbox.h>
#include <qhbox.h>
#include <qstring.h>
#include <qpen.h>
#include "momwindow.h"
#include "momcanvasview.h"
#include "ipinputdialog.h"
#include "discoverydialog.h"
#include "sonardialog.h"
#include "irdialog.h"
#include "infodialog.h"
#include "simulator_constants.h"

Include dependency graph for momwindow.cpp:
7.83  pda/mom/momwindow.h File Reference

#include <qmainwindow.h>
#include <qpopupmenu.h>
#include <qevent.h>
#include <qcanvas.h>
#include <qlabel.h>
#include <qvbox.h>
#include <qhbox.h>
#include <qtimer.h>
#include <qpushbutton.h>
#include "momcanvasview.h"
#include "robotinfo.h"
#include "robot.h"

Include dependency graph for momwindow.h:

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

Compounds

- class MomWindow

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
7.84  pda/mom/robotinfo.cpp File Reference

#include "fstream"
#include "robotinfo.h"
#include "simulator_constants.h"

Include dependency graph for robotinfo.cpp:
7.85  pda/mom/robotinfo.h File Reference

#include <qcanvas.h>

Include dependency graph for robotinfo.h:

```
\begin{center}
\begin{tikzpicture}
\t\node[draw,rectangle] (qc) {qcanvas.h};
\t\node[draw,rectangle,below of=qc] (ri) {robotinfo.h};
\t\draw[->] (qc) -- (ri);
\end{tikzpicture}
\end{center}
```

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

```
\begin{center}
\begin{tikzpicture}
\t\node[draw,rectangle] (ri) {robotinfo.cpp};
\t\node[draw,rectangle,below of=ri] (mo) {mom.cpp};
\t\node[draw,rectangle,below of=mo] (ir) {irdialog.cpp};
\t\node[draw,rectangle,below of=ir] (irh) {irdialog.h};
\t\node[draw,rectangle,left of=ri] (id) {infodialog.cpp};
\t\node[draw,rectangle,left of=id] (idh) {infodialog.h};
\t\node[draw,rectangle,below of=idh] (irid) {irdialog.cpp};
\t\node[draw,rectangle,below of=idh] (irhid) {irdialog.h};
\t\node[draw,rectangle,below of=mo] (mr) {momwindow.cpp};
\t\node[draw,rectangle,below of=mr] (mrh) {momwindow.h};
\t\node[draw,rectangle,below of=mrh] (momin) {robotinfo.cpp};
\t\node[draw,rectangle,below of=momin] (mominh) {robotinfo.h};
\t\draw[->] (ri) -- (mo);
\t\draw[->] (mo) -- (ir);
\t\draw[->] (ir) -- (irh);
\t\draw[->] (irh) -- (irid);
\t\draw[->] (irid) -- (irhid);
\t\draw[->] (irhid) -- (id);
\t\draw[->] (id) -- (ir);
\t\draw[->] (ir) -- (mr);
\t\draw[->] (mr) -- (mrh);
\t\draw[->] (mrh) -- (momin);
\t\draw[->] (momin) -- (mominh);
\end{tikzpicture}
\end{center}
```

Compounds

- class Point
- class PointTheta
- class RobotInfo
7.86  pda/mom/simulator_constants.h File Reference

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

```
+---------------------+  +---------------------+
| simulator_constants.h |  | simulator_constants.h |
|                     |  |                      |
|                      |  | sonardialog.cpp      |
|                      |  | robotinfo.cpp        |
| irdialog.cpp         |  | momwindow.cpp        |
```

Defines

- `#define xCenter 118`
- `#define yCenter 118`
- `#define HEIGHT 236`
- `#define WIDTH 236`

7.86.1  Define Documentation

7.86.1.1  `#define HEIGHT 236`

7.86.1.2  `#define WIDTH 236`

7.86.1.3  `#define xCenter 118`

7.86.1.4  `#define yCenter 118`
#include "sonardialog.h"
#include "simulator_constants.h"

Include dependency graph for sonardialog.cpp:
7.88 pda/mom/sonardialog.h File Reference

#include <qdialog.h>
#include <qvbox.h>
#include <qcanvas.h>
#include <qpushbutton.h>
#include "robot.h"
#include "robotinfo.h"

Include dependency graph for sonardialog.h:

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

Compounds

- class SonarDialog
Chapter 8

robots-all Page Documentation

8.1 Libbehavior Extra Documentation

Kris Beewer (beewer@cs.rpi.edu)
API documentation for libbehavior.
$Id: libbehavior_api.txt,v 1.1 2003/08/01 18:17:31 beewer Exp$

For source-level API documentation see behavior.h, or the doxygen documentation for behavior.h.

Libbehavior is a library for writing reactive behaviors that:

* are started and stopped by the sequencer (see sequencer.txt)
* can take inputs from other behaviors (or deliberative programs)
* can send outputs to other behaviors
* can be inhibited (paused) and uninhibited at any time
* can interact with the robot’s hardware using librobot (see librobot_api.txt)

Behaviors written using libbehavior are mostly event-driven. They provide a set of functions to be called when certain events take place. All of these functions are optional (if a behavior provides none of them, it will just start and sleep forever, doing nothing). Behaviors can also provide one function that is run only once, after initialization, that acts as a "main program." Unlike the other functions in a behavior, this one is not event driven and is only called once. Usually, it should loop forever, almost always while blocking and waiting for some sort of input or occurrence in each loop iteration.

All of the "administrative" work for writing a behavior is taken care of by libbehavior. In particular, it takes care of

* starting up properly and notifying the sequencer of successful initialization
* calling robot_init; the behavior itself should never call robot_init or robot戕down (see the discussion of bhv_init below)
* shutting down properly, either when told to by the sequencer or when the behavior’s main function exits for some reason
* pausing when told to inhibit itself
* queuing data arriving on inputs while inhibited, if requested
* unpauing when told to uninhibit itself
* sending output data to the right recipients
* getting input data and notifying the behavior, or queuing the data if requested
* doing nothing forever if the behavior does not provide its own
main function

A behavior written using libbehavior must:

* provide a bhv_init function corresponding to the prototype in behavior.h
* provide any other functions from bhv_t in behavior.h that it requires

The bhv_init function is libbehavior's "entry point" into the behavior-specific functionality. This function is called when the behavior first starts. It should:

* fill in function pointers in the bhv argument; if you will not be providing one of the functions, do not set its pointer to anything
* fill in robot_init flags and flags from bhv_flags_t in the bhv->flags variable as appropriate
* start any necessary sub-behaviors (and connect their inputs/outputs as necessary)
* return < 0 on failure, or >= 0 otherwise

bhv_init should not do anything intensive itself. Save this for the main function.

Functions that can be specified by the behavior to be called by libbehavior are:

* cleanup (called when the behavior is shutting down)
* on_inhibit (called just before the behavior is inhibited)
* on_uninhibit (called just after the behavior is uninhibited)
* bhv_main (if specified, called after initialization is complete; the behavior will exit as soon as this function exits)

In addition, the behavior can specify functions to be called when data arrives on one of its inputs (as many functions as desired for a single input). See the documentation for bhv_add_input for more information.

Libbehavior will also queue incoming data from inputs if requested. The program can access this data with bhv_input_pop.

Finally, behaviors should use libbehavior's bhv_printf and bhv_perror functions in place of their normal counterparts, to print extra information to the sequencer's log file. printf and perror will send their output to the same file, but bhv_printf and bhv_perror differentiate a behavior's output from that of the sequencer and other behaviors.

The best way to figure out how to write a behavior is to look at some that are already written. In the source tree, look in the arch/bhv directory for examples.
8.2 Sequencer Internals Documentation

Kris Bevers (bevek@cs.rpi.edu)
Discussion of the sequencer internals.
#id: sequencer.txt,v 1.1 2003/08/01 18:17:31 bevek Exp $

The sequencer is the central program in the reactive layer of our robot software "architecture." It is responsible mostly for loading and unloading behaviors that make use of libbehavior.

The sequencer is really a very simple program. It itself does not control robot hardware in any way (in fact, it does not even call librobot's robot_init). All it does is start and stop other programs (behaviors) that act in a specified way according to requests by separate processes.

IPC between other processes and the sequencer is done using Linux's kernel message queues (sys/ipc.h and sys/msg.h). This is a very simple mechanism for delivering distinct messages to a process and requires a lot less parsing than socket communication in most cases. Message queues reside within the Linux kernel and are accessible from anywhere in the system via a special key/queue id. The sequencer's message queue key is specified in seq.h (ROBOT_SEQ_QUEUE_KEY). It is the only pre-specified key in the reactive layer.

The sequencer has internal search paths that dictate where it will look for behaviors to run. If a requested behavior isn't found in these directories, or the cwd of the requesting process, no behavior will be loaded.

Any process talking to the sequencer must have its own message queue, whose key is its pid. This works because all communication is initiated by client processes talking to the sequencer (which make their pids available to it). For non-behaviors, this is handled by robot_init when the RL_USE_SEQUENCER flag is passed to it. For behaviors, this is handled internally by libbehavior.

If the sequencer is told to launch a behavior and it finds the behavior in its search paths, it launches the behavior by forking and calling execve with the behavior's path. It then waits for the behavior to send a special message to the sequencer's message queue indicating a successful initialization. If the behavior's process exits before sending this message, or if the message isn't received within one second, the loading of the behavior is determined to have failed and the requesting process is notified. If the message is received, a "behavior handle" is sent to the requesting process for use in talking directly to the behavior's message queue. This behavior handle is also used to tell the sequencer to unload behaviors. Note that, because the behavior must send this message to the sequencer's message queue, normal programs (i.e. those not written using libbehavior) cannot be launched by the sequencer.

The sequencer also provides functionality for "attaching" to an already-running behavior (started by some other process), given proper permissions. Most normal programs won't need to do this, but it can occasionally be useful for starting a behavior without the calling process continuing to run, and then stopping it again later. The bhvtl program does this.

Also in seq.h is functionality for talking directly to behaviors (once they have been loaded by the sequencer). Specifically, the following actions can be performed on behaviors:

* inhibit (pause) a behavior
* uninhibit a behavior
* send data to one of a behavior's inputs
* get data from one of a behavior's outputs

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
* connect one behavior's outputs to another's inputs
* disconnect one behavior's outputs from another's inputs

For more informations on how these actions affect behaviors, see libbehavior_api.txt.
8.3 Compilation Instructions

Robot source compilation/installation howto
Kris Beever (beever@cs.rpi.edu)
$Id: compiling.txt,v 1.6 2003/08/01 16:17:31 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/rc234.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.

Un-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.
8.4 Librobot Extra Documentation

Kris Bevans (bevek@cs.rpi.edu)
API documentation for librobot.
$Id: librobot_api.txt,v 1.2 2003/08/01 18:17:31 bevek Exp$

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)
LIBLIBS += $(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.
8.5 General Robot Software Specifications

Kris Beever (beevek@cs.rpi.edu)

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

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)
* netrobot (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:

* mrtool   (console interface for controlling robot over the
            network)
* logtool  (examine log files produced by logger)
* bhvtcl   (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_dataflow.txt and motor_control_interface.txt.
Information on librobot and netrobot 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, netrobot, logger,
sequencer, mrtool, logtool, bhvtcl) 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 nrobot remotely. All nrobot 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 nrobotd).

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)
8.6 Low Level Data Flow

Kris Beevers (beevek@cs.rpi.edu)
Some discussion of the low-level data flow/management on the CPU end of the robot stuff.
$Id: low_level_data_flow.txt,v 1.4 2003/06/01 18:17:31 beevek Exp$

Microcontrollers controlling the motors and sensors will (at least for now) be speaking through the serial port to the cpu. Sitting on and reading from the serial port will be a high-priority process, the "interpreter," whose responsibility is mainly to read data from the serial device and pass it on to the rest of the system, and to take data from the rest of the system and write it to the serial device.

Data from the microcontrollers to the computer will be of the format:

```
[hwid (1 byte)][data (1 to 4 bytes)]
```

and data from the computer back to the microcontrollers will follow the same format. The "hardware id" (hwid) will refer to a specific "hardware device" controlled by the microcontrollers (e.g. motors, individual sensors, etc). See hwid.h for the set of valid hwids. The "data" is device-specific and will be handled by the interpreter. Some hwids that are available to high-level components in the system are completely "virtual" and are handled internally by the interpreter.

The interpreter is responsible for performing fast motor control in response to encoder counts sent by the microcontrollers. It does this by loading a motor control shared library. For details on the library interface, see motor_control_interface.txt, mclib.h and the mc/ directory. A PID control shared library is provided as the default controller. All motor controllers generate PWM counts in response to encoder counts, which are sent back to the motors through the interpreter (as well as forwarded to the rest of the system as data).

All data (including encoder/pwm data) coming from the microcontrollers is written by the interpreter to a set of special character devices in the /dev/robot filesystem. These files are created by the robot.o kernel module (see the driver/ directory). In general, each component provides several methods of access from user-level programs through /dev/robot, each with different characteristics:

- **stream**: any time data is written to the /dev/robot/.../stream entry, this device reports new data as being available for reading. in other words, every bit of data that the interpreter writes to /dev/robot for a particular component can be read from its stream device file.

- **change**: this file acts similarly to stream. however, new data is only written to it if the data is different than the previous data that was written. for values like sonar/ir readings, this is based on a threshold difference (see constants.h).

- **current**: whenever data is written to stream files, it is also written to the corresponding current file. however, current never blocks on a read. it always returns whatever the current value of the data for its component is.

- **ctl**: this device file is special. user-level programs can write to it (rather than just read). the interpreter reads from it, and data sent to it is used to control the robot's hardware.

All of these files require the data buffers being read or written
(through the read/write system calls) to be of exactly the size they expect (e.g., setting the velocity requires exactly two floats). In general, user-level programs will use the librobot api rather than accessing the /dev/robot files directly, and will not need to worry about this. See the file devfs.h for data sizes for different /dev/robot entries.

Currently, device files supported by robot.o are as follows; for type information, see types.h and for constants (e.g. ROBOT_NUM_*) see constants.h:

/dev/robot/vel/{stream,change,current,ctl}
  get/set robot’s translational and rotational velocity
    read: two vel_val_t
    write (ctl): two vel_val_t

/dev/robot/odom/{stream,change,current,ctl}
  get/set robot’s odometric information (x, y, theta)
    read: three odom_val_t
    write (ctl): three odom_val_t

/dev/robot/encoder/{stream,change,current}
  get robot’s motor encoder counts (left, right)
    read: two encoder_val_t
    write (ctl): not available

/dev/robot/pwm/{stream,change,current}
  get motor controller’s pwm counts (left, right)
    read: two pwm_val_t
    write (ctl): not available

/dev/robot/sonar/{stream,change,current,ctl}
  get sonar range information for all sonars at once, or set firing frequencies for all sonars at once
    read: sonar_val_t * ROBOT_NUM_SONAR
    write (ctl): freq_req_val_t * ROBOT_NUM_SONAR

/dev/robot/sonar/{0-ROBOT_NUM_SONAR}/stream,change,current,ctl}
  get sonar range information or set firing frequency for an individual sonar sensor
    read: sonar_val_t
    write (ctl): freq_req_val_t

/dev/robot/ir/{stream,change,current,ctl}
  get ir range information for all ir sensors at once, or set firing frequencies for all ir sensors at once
    read: ir_val_t * ROBOT_NUM_IR
    write (ctl): freq_req_val_t * ROBOT_NUM_IR

/dev/robot/ir/{0-ROBOT_NUM_IR}/stream,change,current,ctl}
  get ir range information or set firing frequency for an individual ir sensor
    read: ir_val_t
    write (ctl): freq_req_val_t

/dev/robot/bump/{stream,change,current}
  get bump sensor toggle information for all bump sensors at once
    read: bump_val_t
    write (ctl): not available

/dev/robot/bump/{0-ROBOT_NUM_BUMP}
  get bump sensor toggle information for an individual sensor
    read: bump_val_t
    write (ctl): not available

The robot.o driver also provides special locking capabilities, only for ctl-type device files. This is through an ioctl call. The
locking is based on a priority mechanism. If no lock is currently held on a c'tl file, any lock request will be granted and no other process will be able to write to the file. If a lock is currently held, a lock request will be granted ONLY if the priority of the request is HIGHER than that of the currently held lock. Any process with a higher priority than the currently held lock (or the process that holds the lock) can unlock the c'tl file as well. For more information on this, the documentation for the functions devfs_lock_ctl and devfs_unlock_ctl from devfs.h.
8.7 Motor Controller Interface

Kris Beever (kbeever@ecs.rpi.edu)
Some initial motor controller thoughts/specs:
$id: motor_control_interface.txt,v 1.5 2003/06/27 14:25:40 kbeever Exp$

The motor controller workings have been almost completely overhauled since the original version of this document. The interface they present remains mostly similar, but internally they have been restructured significantly.

Motor controllers are shared libraries that provide a strict set of functionality to whoever loads them, via an interface described in include/robot/mclib.h. Programs use librobot's mclib functions to load a motor control library and call its functions. Note that in general, only the interpreter will be doing this but it is possible for any program on the robot to load and call the functions of a motor control shared library (this is particularly useful in debugging, or, say, simulation).

The rest of this document assumes the motor controllers have been loaded into the interpreter.

The interpreter speaks directly to the hardware for the robots (via serial). At some fixed frequency (currently 100Hz), the motors send it encoder counts; it must call the current motor controller's methods as appropriate and return pwm counts to the motors. The interpreter is also responsible for keeping track of odometry information and sends this, along with current velocity information and encoder counts (as well as information from other sensors) to the robot's /dev/robot filesystem for use by higher-level programs.

Because motor control needs to happen at such a fast rate, the motor control library's functions (in particular, mc_do_control) need to be very efficient.

Motor controllers cannot assume a fixed rate (i.e. they should not define FREQUENCY 100 or something of the sort) because the interpreter runs in user space. Instead the mc_start_frame function is called at the start of every "frame" of data to update timer information and calculate the amount of time that has passed since the last motor control update. This function is implemented in mc_common.c, though a particular motor controller may re-implement it if necessary.

Higher-level software in the system will tell set the motor controller's desired velocity (via the interpreter's /dev/robot interface). The motor controller is responsible for converting a $v, \psi$ pair into a $v_l, v_r$ pair. Generally, the functionality provided by mc_set_velocity in mc_common.c should be able to handle this without changes.

The interface to be provided by motor controllers has changed to reflect the fact that the motor controller should store state information internally. In particular, only mc_start_frame actually takes encoder counts as arguments (these should be stored internally by that function for the rest of the frame -- this is done in the default implementation). Storing state information internally leads to a much cleaner interface and reduced computation (especially for functions like mc_set_odometry and mc_get_velocity).

Following is the interface to be provided by motor controllers (i.e., they must implement the following functions). Most of the types involved are defined in include/robot/types.h. Constants useful in motor control (e.g. wheel radii, axle width, etc.) are defined in include/robot/constant.h.
int mc_init(uint32_t flags);

Perform any necessary initialization. flags is currently unused but might eventually be used to pass some sort of options. A common initialization function is provided in mc_common.c as mc_init_common. In general, most controller-specific mc_init functions should call mc_init_common at the top of their function.

Return < 0 on failure, >= 0 otherwise.

void mc_shutdown(void);

Shut down the motor controller. Free any used memory and do any necessary de-initialization of anything else internal to the controller. A common version of this function is provided in mc_common.c.

int mc_start_frame(encoder_val_t left, encoder_val_t right);

Must be called at the beginning of each "data frame" to set internal variables containing current encoder counts. This will be called before mc_get_velocity, mc_set_odometry and mc_do_control. It should also calculate the change in time since the last frame and use this in its calculations, rather than assume a fixed frequency. A common version of this function is provided in mc_common.c and should work fine for most controllers.

Return < 0 on failure, >= 0 otherwise.

int mc_set_velocity(vel_val_t v, vel_val_t w);

Set the desired translational and rotational velocities. The controller should handle converting these values into values for the velocities of the left and right motors, and store these internally. Either of these velocities can be negative (to indicate "reverse"). A common version of this function is provided in mc_common.c.

Return < 0 on failure, >= 0 otherwise.

void mc_get_velocity(vel_val_t *v, vel_val_t *w);

Use the encoder values for the current frame to calculate the current translational (v) and rotational (w) velocities, and fill in the arguments. A common version of this function is provided in mc_common.c.

void mc_set_odometry(odom_val_t *x,odom_val_t *y, odom_val_t *theta);

Use the encoder values for the current frame to update the previous odometry values stored in x, y and theta. This method should modify the odometry values (add to them), not replace them starting from zero. A common version of this function is provided in mc_common.c.

Return < 0 on failure, >= 0 otherwise.

int mc_do_control(pwm_val_t *left_pwm, pwm_val_t *right_pwm);

Use the encoder values for the current frame to calculate pwm counts to send to the motor (also using internally stored "correct" encoder counts set in mc_start_frame based on desired velocities set by mc_set_velocity). This function is definitely always specific to each type of motor controller.

Return < 0 on failure, >= 0 otherwise.
8.8 Data Logging

Kris Bevers (bevek@cs.rpi.edu)
Discussion of data logging mechanism for the robot

Often it will be convenient to have access to data generated by the robot’s hardware and software during the course of an experiment. We’d like to have a logging mechanism that can record such data so that we can "play it back" later, i.e. either just by looking at it manually or by running commands through a simulator.

Conveniently, we have the /dev/robot interface, a virtual filesystem through which almost all of the data we would be interested in passes. Consequently, we can just log data by reading these files as appropriate. A simple "logging daemon" that just opens all of the files for reading and writes to the log when new data is available will suffice.

There are a couple of issues. First, we want accurate timing information for each event. Internally, the robot uses 64-bit microsecond values for timing. We will make use of this mechanism. The time of each event will just be the number of microseconds since the Unix epoch (see time(2)).

Second, the robot will be generating a huge amount of data. In particular, encoder counts will be coming from the motors at up to 100 times per second, and pwm counts will be going back out at the same rate. Pwm counts are separate from encoder counts in /dev/robot, so they will be treated as separate events. So too will velocity and odometry updates. So, assuming a 64-bit timestamp, this is a minimum of:

\[
64/16 + 64 + 8 + 64 + 8 + 64 + 32 + 64 + 32 = 464 \text{ bits} = 58 \text{ bytes}
\]
at a rate of 100 per second, i.e. 5800 bytes per second. This seems bad at first, but consider that with 3600 seconds in an hour, this is only 20880000 bytes per hour (20.88 MB). Other data will be produced at a much less significant rate and will not significantly change this.

Still, the logger will rotate logs after they reach a certain size. A script should be run periodically on the robot to destroy out-of-date log files.

For the sake of simplicity, all data will be logged to one file (rather than, say, separate files for each type of data). The /dev/robot structure lends itself well to this. Every "event" in the log file will follow this format:

```
[HWD] [DEV_TYPE] [timestamp] [data]
```

HWD is a hardware id number from hwid.h. DEV_TYPE is from devfs_type_t in devfs.h. timestamp is just a robot_time_us_t from types.h. data is specific to HWD and is known by the logger. Note that no data from DEV_TYPE_CURRENT will ever be logged. In general, only data from DEV_TYPE_CHANGE and DEV_TYPE_CTL will be logged, but the logger will optionally log data from DEV_TYPE_STREAM. Also, only data from HW_ALL, (SONAR, IR, BUMP) will be logged (rather than from every individual hardware device). No data is lost by doing this. However, all frequency requests for each HW_ (SONAR, IR) will be logged.

So, for example, all velocity control from a user will be written to the log with entries like:

```
[HWD_VEL] [DEV_TYPE_CTL] [<time>] [v] [w]
```
In order to make it easier to read from to these logfiles, liblogger provides class log_reader_t (see logger.h). A tool, logtool, is provided that simply reads all entries from a logfile and prints them out in a human-readable form (that is also convenient for grepping).

Both the logger and logtool have a number of command-line options. Run them with --help to print these out.
8.9 Simulator Interface

Christopher Chiaverini (chiav@cs.rpi.edu)
Simulator usage instructions:
$Id: simulator_interface.txt,v 1.2 2003/06/05 20:56:41 chiav Exp$

The simulator is designed to simulate differential-drive robots of any size and shape in a configurable environment. The simulator can simultaneously simulate SONAR, infrared, and bump sensors to allow the robots to determine information about their environment.

In order to set up the environment for a simulation at least 3 files are needed:

1) A file that describes the world
2) One or more files that describe the robots to be used in the simulation
3) A problem file that identifies the world file as well as any robot files

The format of each of these files is as follows:

The World Definition File -- A world definition must give a name for the world, and defines the boundary, and any number of obstacles (including none) within the boundary. All distances are in meters!

A typical World file is as follows (minus the header and the footer):

--------- Begin World File ---------

#comment
(A single word that names the world)

#comment
(The number of world boundary points)
(The boundary points defined in Clockwise order)

#comment
(The name of the obstacle)
(The number of obstacle boundary points)
(The boundary points defined in Counter-Clockwise order)

(any additional obstacle definitions go here...)

--------- End World File ---------

The Robot Definition File(s) -- A robot definition file must give a name for the robot, define the robot's starting position/center of rotation, the radial distance between the wheels and the center of rotation, the initial orientation of the robot, and the boundary of the robot. Any number of SONAR, infrared, and bump sensors is optional. All distances are in meters!

The robot definition is meant to be as customizable as possible, but please take note of these things:

The CIR should define the point that the robot rotates around when spinning in place, and should be the point along the axis of the wheels that is exactly in the center.

The center of rotation defines the starting point of the robot when the simulation is run.

All points on the robot should be defined relative to the CIR. The points should define the robot at an orientation of 0 radians. The robot boundary should not be defined in absolute terms!

The simulator assumes that at an orientation of 0 radians, the forward direction of the robot lies along the x-axis.
All angles must be defined in radians, defining them in degrees will cause undesirable effects.

The definitions and orientations of all sensors should be defined as if the robot is at an orientation of 0 radians.

The definitions of all sensors should be defined in numerical order. It does not matter if this is clockwise, counter-clockwise, or in any arbitrary order.

The boundary of the bump sensors should be defined as if the sensor is at an orientation of 0 radians, is centered along the x-axis, and has its back edge aligned with the y-axis.

The defining point of a bump sensor is the origin above and bump sensors should be placed on the robot according to this point.

A typical Robot file is as follows (minus the header and the footer):

```
---------- Begin Robot File ----------
#comment
(A single word that names the robot)
#comment
(A point that defines the CUR of the robot)
#comment
(The radial distance from the CUR to one of the wheels)
#comment
(The initial orientation of the robot)
#comment
(The acceleration of the robot)
#comment
(The number of robot boundary points)
(The boundary points defined in Counter-Clockwise order)
#comment
(The amount of noise in translational motion)
#comment
(The amount of noise in rotational motion)
#comment
(The number of SONAR sensors)
(A point that describes the placement of this sensor)
(The orientation of this sensor)
(any additional SONAR sensor definitions go here...)
#comment
(The angular width of the SONAR beam)
#comment
(The maximum distance of SONAR data)
#comment
(The amount of noise in SONAR measurements)
#comment
(The number of IR sensors)
(A point that describes the placement of this sensor)
(The orientation of this sensor)
(any additional IR sensor definitions go here...)
```
#comment
(The maximum distance of ir data)

#comment
(The amount of noise in infrared measurements)

#comment
(The number of boundary points for a bump sensor)
(The boundary points defined in Counter-Clockwise order)

#comment
(The number of bump sensors)
(A point that describes the placement of this sensor)
(The orientation of this sensor)
(any additional bump sensor definitions go here...)

----------- End Robot File  -----------

The Problem File -- The problem file is given on the command line, and defines
the name of the world file and robot files to use for the simulation. It also
defines whether or not log playback should occur.

A typical Problem file is as follows (minus the header and the footer):

----------- Begin Problem File  -----------

#comment
(The name of the world file)

#comment
(The number of robots in the simulation)

#comment
(The name of the robot definition file)
(any additional robot definition files go here...)

#comment
(The number of log files that should be read from)

#comment
(The name of the log file to read from)
(Any additional log files)

----------- End Problem File  -----------

The simulator now uses the same API as the robots do. Programs can be tested
with the simulator, then run on the robot with no code changes or recompilation
necessary. See the documentation on the robot interface for more information
on the API.

New Features:

-----------

Robots do not appear in the environment until a client connects to it. At this
point, the robot will appear at the starting point specified in the log file.
When a program finishes and robot_shutdown is called, the robot will disappear
from the environment.

As a result of the above change, programs will not be able to use the "search"
functionality to connect to the simulator. All simulator connections must be
made by manually giving an IP address. The simulator will not respond to ping
commands.

The acceleration of the robot is now specifiable.
Infrared range is now supported. This allows the infrared sensors to double as laser rangefinders if so desired.

SONAR and infrared frequencies are now supported, and as a result this information will no longer be generated on the fly. In order to better simulate real robots, the values of these sensors will be generated in the order they are specified, at the frequency specified by the user. This means that immediate changes in distance may not be available (i.e. passing a doorway) and depending on the sensor frequency specified, they may not be noticed at all.

Sensor noise and odometry noise has been implemented. The amount of noise can be specified in the description file for each robot. If no noise is wanted, these should be set to 0.

Robot actions can be played back from a log file generated by the robot. The log file is simulated from beginning to end. It is suggested that the user extract the portion of the log file that is of interest.
8.10 Todo List

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

**File motors.h** FIXME: acceleration?

**Member robot.get_id(void)** FIXME implement

**Member robot.get_name(void)** FIXME implement

**Member ihelp** FIXME: mc path

**Member DEFAULT_MC_LIB** FIXME

**File mc_pid.c** FIXME: remove functions to get/set gains
Index

~DiscoveryDialog
   DiscoveryDialog, 21
~IRDialog
   IRDialog, 25
~InfoDialog
   InfoDialog, 23
~IpInputDialog
   IpInputDialog, 24
~MomCanvasView
   MomCanvasView, 29
~MomWindow
   MomWindow, 31
~RobotInfo
   RobotInfo, 42
~SonarDialog
   SonarDialog, 56
~log_writer
   log_writer, 26

about
   MomWindow, 31
acceleration
   robot_simulator.cpp, 260
accept
   IpInputDialog, 24
add_ctl
   logger.cpp, 231
add_dir
   logger.cpp, 231
add_lwidi
   devfs_net.c, 186
alpha
   simrobot.cpp, 263
arch/bhv/test.c, 59
arch/bhv/testsub.c, 61
arch/libbehavior/bhv_main.c, 63
arch/seq/bhvctcpp, 68
arch/seq/sequencer.cpp, 72
assert
   util.h, 149
bchelp
   bhvctcpp, 70
begin
   logtool.cpp, 237
behavior.cpp
   main, 267
behavior.h
   bhv_add_input, 93
   bhv_block_forever, 94
   bhv_cleanup_func, 92
   bhv_data_func, 92
   bhv_flags, 93
   bhv_inhibit_func, 92
   bhv_init, 94
   bhv_input_pop, 94
   bhv_main_func, 93
   bhv_name, 95
   bhv_output, 94
   bhv_error, 92
   bhv_printf, 92
   bhv_self, 95
   bhv_shutdown, 95
   bhv_uninhibit_func, 93
   NO_robot_init, 93
   QUEUE_WHEN_INHIBITED, 93
behaviors
   sequencer.cpp, 77
bhv_add_input
   behavior.h, 93
   bhv_main.c, 65
bhv_block_forever
   behavior.h, 94
   bhv_main.c, 65
bhv_cleanup_func
   behavior.h, 92
BHv_CONNECT
   seq.h, 132
bhv_connection
   from, 13
   from_key, 13
to, 13
to_key, 13
BHv_DATA
   seq.h, 132
bhv_data_func
   behavior.h, 92
bhv_data
   data, 15
   key, 15
devfs_set_t, 20
clamp
mc_pid.c, 228
cleanup
bhv_t, 17
cleanup_devices
devices.c, 80
robotdrv.c, 82
cleanup_module
robotdrv.c, 82
cleanup_robot
nrtool.cpp, 250
client
netrobotd.cpp, 246
close
log_writer_t, 26
cmd
seq_msgbuf_t, 44
console
nrtool.cpp, 250
constants.h
ROBOT_AXLE_WIDTH, 97
ROBOT_DEFAULT_IR_FREQ, 97
ROBOT_DEFAULT_NET_PORT, 97
ROBOT_DEFAULT_NET_TIMEOUT, 97
ROBOT_DEFAULT_SONAR_FREQ, 97
ROBOT_DEFAULT_VEL_V, 97
ROBOT_DEFAULT_VEL_W, 97
ROBOT_ENCODER_STEPS_PER_REV, 97
ROBOT_HW_TTY, 97
ROBOT_IR_MULTIPLIER, 97
ROBOT_MAX_NAME_LEN, 97
ROBOT_MAX_VEL_V, 97
ROBOT_MAX_VEL_W, 97
ROBOT_NUM_BUMP, 98
ROBOT_NUM_IR, 98
ROBOT_NUM_SONAR, 98
ROBOT_SONAR_MULTIPLIER, 98
ROBOT_THRESH_IR, 98
ROBOT_THRESH_ODOM_THETA, 98
ROBOT_THRESH_ODOM_X, 98
ROBOT_THRESH_ODOM_Y, 98
ROBOT_THRESH_ROTATE, 98
ROBOT_THRESH_SONAR, 98
ROBOT_THRESH_TRANSLATE, 98
ROBOT_THRESH_VEL_V, 98
ROBOT_THRESH_VEL_W, 98
ROBOT_WHEEL_RADIUS, 99
ROBOT_WHEEL_RADIUS_L, 99
ROBOT_WHEEL_RADIUS_R, 99
contentsMouseMoveEvent
MomCanvasView, 29
contentMouseReleaseEvent
MomCanvasView, 29
count
robot_net_msg_t, 41
CTL
robotdrv.h, 84
cnt
devfs_set_t, 20
cnt_read_and_dispatch
interp.c, 158
CTL_SIZE_BUMP
devfs.h, 103
CTL_SIZE_ENCODER
devfs.h, 103
CTL_SIZE_IR
devfs.h, 103
CTL_SIZE_ODOM
devfs.h, 103
CTL_SIZE_PWM
devfs.h, 103
CTL_SIZE_SONAR
devfs.h, 103
data
bhv_data_t, 15
robot_dev_t, 36
seq_msgbuf_t, 44
serial_cmd_t, 45
data_len_t, 19
from_dev, 19
to_dev, 19
data_lengths
data_lengths_x.c, 152
interp.h, 164
data_lengths.c
data_lengths, 152
data_lengths_init, 151
data_lengths_init
data_lengths_x.c, 151
interp.h, 162
data_size
robot_dev_t, 36
DATA_SIZE_BUMP
devfs.h, 103
DATA_SIZE_ENCODER
devfs.h, 103
DATA_SIZE, 103
DATA_SIZE_OBJ
devfs.h, 103
DATA_SIZE_ODOM
devfs.h, 103
DATA_SIZE_PWM
devfs.h, 103
DATA_SIZE_HYSTERESIS
devfs.h, 103
dcnt
robot_dev.t, 36
robot_info.t, 38
DEFAULT_MC_LIB
interph, 162
defaultSeed
simrobot.cpp, 263
de
robot_info.t, 38
DEV_TYPE_CHANGE
devfs.h, 105
DEV_TYPE_CHANGE_STR
devfs.h, 103
DEV_TYPE_CTL
devfs.h, 105
DEV_TYPE_CTL_STR
devfs.h, 103
DEV_TYPE_CURRENT
devfs.h, 105
DEV_TYPE_CURRENT_STR
devfs.h, 103
DEV_TYPE_STREAM
devfs.h, 105
DEV_TYPE_STREAM_STR
devfs.h, 103
devfs.c
devfs_cleanup, 177
devfs_find, 177
devfs_get_data_size, 178
devfs_get_lock_owner, 178
devfs_init, 178
devfs_loc_cleanup, 178
devfs_loc_get_lock_owner, 179
devfs_loc_init, 179
devfs_loc_lock_ctl, 179
devfs_loc_read, 179
devfs_loc_unlock_ctl, 179
devfs_loc_wait_for_change, 179
devfs_loc_write, 179
devfs_lock_ctl, 179
devfs_net_cleanup, 179
devfs_net_get_lock_owner, 179
devfs_net_init, 179
devfs_net_lock_ctl, 179
devfs_net_read, 179
devfs_net_unlock_ctl, 179
devfs_net_wait_for_change, 179
devfs_net_write, 179
devfs_read, 179
devfs_unlock_ctl, 180
devfs_wait_for_change, 180
devfs_write, 180
errno, 181
devfs
CTL_SIZE_BUMP, 103
CTL_SIZE_ENCODER, 103
CTL_SIZE_JR, 103
CTL_SIZE_ODOM, 103
CTL_SIZE_pwm, 103
CTL_SIZE_SONAR, 103
CTL_SIZE_VEL, 103
DATA_SIZE_BUMP, 103
DATA_SIZE_ENCODER, 103
DATA_SIZE_JR, 103
DATA_SIZE_ODOM, 103
DATA_SIZE_PWM, 103
DATA_SIZE_SONAR, 103
DATA_SIZE_VEL, 103
DEV_TYPE_CHANGE, 105
DEV_TYPE_CHANGE_STR, 103
DEV_TYPE_CTL, 105
DEV_TYPE_CTL_STR, 103
DEV_TYPE_CURRENT, 105
DEV_TYPE_CURRENT_STR, 103
DEV_TYPE_STREAM, 105
DEV_TYPE_STREAM_STR, 103
DEVFS_BUMP, 105
DEVFS_BUMP_SINGLE, 105
DEVFS_BUMP_STR, 103
devfs_cleanup, 105
DEVFS_CLIENT, 104
devfs_dir_type, 104
DEVFS_ENCODER, 105
DEVFS_ENCODER_STR, 103
devfs_find, 105
devfs_get_data_size, 105
devfs_get_lock_owner, 106
devfs_init, 106
DEVFS_IOC_GET_LOCK_OWNER, 103
DEVFS_IOCLOCKCTL, 103
DEVFS_IOC_UNLOCKCTL, 103
DEVFS_JR, 105
DEVFS_JR_SINGLE, 105
DEVFS_JR_STR, 104
devfs_lock_ctl, 106
DEVFS_ODOM, 105
DEVFS_ODOM_STR, 104
DEVFS_PRIO_CRITICAL, 104
DEVFS_PRIO_HIGH, 104
DEVFS_PRIO_LOW, 104
DEVFS_PRIO_NORMAL, 104
DEVFS_PRIO_SUPER, 104
DEVFS_PWM, 105
DEVFS_PWM_STR, 104
DEVFS_RDONLY, 104
devfs_read, 106
DEVFS_ROOT_STR, 104
DEVFS_SERVER, 104
DEVFS_SONAR, 105
DEVFS_SONAR_SINGLE, 105
DEVFS_SONAR_STR, 104
devfs_type, 105
devfs_unlock_ctl, 107
DEVFS_VEL, 105
DEVFS_VEL_STR, 104
devfs_wait_for_change, 107
devfs_write, 107
NUM_DEVICE_DIRS, 104
NUM_DEVICES, 104
DEVFS_BUMP
devfs, 105
DEVFS_BUMP_SINGLEDevfs, 105
DEVFS_BUMP_STRDevfs.h, 103
devfs_cleanup
devfs.c, 177
devfs.h, 105
DEVFS_CLIENTdevfs.h, 104
devfs_dir_type
devfs.h, 104
DEVFS_ENCODERdevfs.h, 105
DEVFS_ENCODER_STRDevfs.h, 103
devfs_initrobot_handler, 39
devfs_initdevfs.c, 177
devfs.h, 105
devfs_flagsrobot_handler, 39
devfs_get_data_sizedevfs.c, 178
devfs.h, 105
devfs_get_lock_ownerdevfs.c, 178
devfs.h, 106
devfs_initdevfs.c, 178
devfs.h, 106
DEVFS_IOC_GET_LOCK_OWNER
devfs.h, 103
DEVFS_IOC_CLOCKCTL
devfs.h, 103
DEVFS_IOC_UNLOCKCTL
devfs.h, 103
DEVFS_IR
devfs.h, 105
DEVFS_IR_SINGLEDevfs.h, 105
DEVFS_IR_STRDevfs.h, 104
devfs_loc_cleanup
devfs.c, 178
devfs_local.c, 184
devfs_loc_get_lock_owner
devfs.c, 179
devfs_local.c, 184
devfs_loc_init
devfs.c, 179
devfs_local.c, 184
devfs_loc_lock_ctl
devfs.c, 179
devfs_local.c, 184
devfs_loc_read
devfs.c, 179
devfs_local.c, 184
devfs_loc_unlock_ctl
devfs.c, 179
devfs_local.c, 184
devfs_loc_wait_for_change
devfs.c, 179
devfs_local.c, 184
devfs_loc_write
devfs.c, 179
devfs_local.c, 184
devfs_local.c
devfs_loc_cleanup, 184
devfs_loc_get_lock_owner, 184
devfs_loc_init, 184
devfs_loc_lock_ctl, 184
devfs_loc_read, 184
devfs_loc_unlock_ctl, 184
devfs_loc_wait_for_change, 184
devfs_loc_write, 184
DNFMT, 183
errno, 184
init_dir_fail, 183
set_dir_fail, 183
devfs_lock_ctl
devfs.c, 179
devfs.h, 106
devfs_net.c

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
add_hwid, 186
devfs_net_cleanup, 186
devfs_net_get_lock_owner, 186
devfs_net_init, 186
devfs_net_lock_ctl, 186
devfs_net_read, 186
devfs_net_unlock_ctl, 186
devfs_net_wait_for_change, 186
devfs_net_write, 186
erno, 187
msg_get_lock_owner, 187
msg_init, 187
msg_lock_ctl, 187
msg_shutdown, 187
msg_unlock_ctl, 187
msg_wait_change, 187
net_connect_tcp, 186
net_read_msg, 187
net_write_msg, 187
devfs_net_cleanup
devfs.c, 179
devfs_net.c, 186
devfs_net_get_lock_owner
devfs.c, 179
devfs_net.c, 186
devfs_net_init
devfs.c, 179
devfs_net.c, 186
devfs_net_lock_ctl
devfs.c, 179
devfs_net.c, 186
devfs_net_read
devfs.c, 179
devfs_net.c, 186
devfs_net_unlock_ctl
devfs.c, 179
devfs_net.c, 186
devfs_net_wait_for_change
devfs.c, 179
devfs_net.c, 186
devfs_net_write
devfs.c, 179
devfs_net.c, 186
DEVFS_ODOM
devfs.h, 105
DEVFS_ODOM_STR
devfs.h, 104
DEVFS_PRIO_CRITICAL
devfs.h, 104
DEVFS_PRIO_HIGH
devfs.h, 104
DEVFS_PRIO_LOW
devfs.h, 104
DEVFS_PRIO_NORMAL
devfs.h, 104
DEVFS_PRIO_SUPER
devfs.h, 104
DEVFS_PWM
devfs.h, 105
DEVFS_PWM_STR
devfs.h, 104
DEVFS_RDONLY
devfs.h, 104
devfs_read
devfs.c, 179
devfs.h, 106
DEVFS_ROOT_STR
devfs.h, 104
DEVFS_SERVER
devfs.h, 104
devfs_set
change, 20
ctl, 20
current, 20
hwid, 20
stream, 20
DEVFS_SONAR
devfs.h, 105
DEVFS_SONAR_SINGLE
devfs.h, 105
DEVFS_SONAR_STR
devfs.h, 104
devfs_type
robot_net_msg.t, 41
devfs_type.t
devfs.h, 105
devfs_unlock_ctl
devfs.c, 180
devfs.h, 107
DEVFS_VEL
devfs.h, 105
DEVFS_VEL_STR
devfs.h, 104
devfs_wait_for_change
devfs.c, 180
devfs.h, 107
devfs_write
devfs.c, 180
devfs.h, 107
devices.c
cleanup_devfs, 80
init_devfs, 80
INIT_DIR, 80
NUM_DIRS, 80
robot_devices, 80
disconnect
MomWindow, 31
discover

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
freq_set
  freq.c, 155
  interp.h, 162
freq_to_hw_freq
  freq.c, 154
freq_val
  types.h, 146
from
  bhv_connection.h, 13
from dev
data_len.h, 19
from key
  bhv_connection.h, 13

G robot_simulator.cpp, 260
getAllBump
  robot_simulator.cpp, 258
getAllIr
  robot_simulator.cpp, 258
getAllIrData
  SimRobot, 48
getAllSonar
  robot_simulator.cpp, 258
getAllSonarData
  SimRobot, 48
getBump
  robot_simulator.cpp, 258
getEstimatedCor
  SimRobot, 49
getIp
  IpInputDialog, 24
getIr
  robot_simulator.cpp, 258
getIrData
  SimRobot, 49
getOdometry
  robot_simulator.cpp, 258
getPort
  IpInputDialog, 24
getSonar
  robot_simulator.cpp, 258
getSonarData
  SimRobot, 49
getTrueCor
  SimRobot, 49
getVelocity
  robot_simulator.cpp, 258
getA
  SimRobot, 50
getBoundaryIR
  RobotInfo, 42
getBoundarySonar
  RobotInfo, 42
getBumpData
  SimRobot, 50
getBumpObjects
  SimRobot, 50
getEstimatedPath
  SimRobot, 50
getHandle
  MomWindow, 31
getInfo
  IRDialog, 25
  SonarDialog, 56
getIRBeams
  SimRobot, 51
getIRLine
  RobotInfo, 42
getIRObjects
  SimRobot, 51
getName
  SimRobot, 51
getNextLog
  mpProblem, 32
getNextRobot
  mpProblem, 32
getNumBumps
  SimRobot, 51
getNumIRs
  SimRobot, 51
getNumSonars
  SimRobot, 51
getOdometry
  SimRobot, 52
getPath
  SimRobot, 52
getRInfo
  MomWindow, 31
getSelection
  DiscoveryDialog, 21
getSonarObjects
  SimRobot, 52
getSonarPolygon
  RobotInfo, 42
getSonarScan
  SimRobot, 52
getTargetV
  SimRobot, 52
getTargetW
  SimRobot, 52
getV
  SimRobot, 53
getW
  SimRobot, 53
h_find
  nrtool.cpp, 250
<table>
<thead>
<tr>
<th>Function</th>
<th>File(s)</th>
<th>Line(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>robot_set_handle</td>
<td>nrtool.cpp, 251</td>
<td>110</td>
</tr>
<tr>
<td>handle_detach</td>
<td>sequencer.cpp, 75</td>
<td></td>
</tr>
<tr>
<td>handle_is_connected</td>
<td>handle.h, 110</td>
<td></td>
</tr>
<tr>
<td>handle_is_loc</td>
<td>handle.h, 110</td>
<td></td>
</tr>
<tr>
<td>handle_is_net</td>
<td>handle.h, 110</td>
<td></td>
</tr>
<tr>
<td>handle_list</td>
<td>sequencer.cpp, 75</td>
<td></td>
</tr>
<tr>
<td>handle_load</td>
<td>sequencer.cpp, 75</td>
<td></td>
</tr>
<tr>
<td>handle_unload</td>
<td>sequencer.cpp, 75</td>
<td></td>
</tr>
<tr>
<td>handlers</td>
<td>nrtool.cpp, 252</td>
<td></td>
</tr>
<tr>
<td>HEIGHT</td>
<td>simulator_constants.h</td>
<td>292</td>
</tr>
<tr>
<td>HW_ALL_BUMP</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_ALL_IR</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_ALL_SONAR</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP01</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP02</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP03</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP04</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP05</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP06</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP07</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>HW_BUMP08</td>
<td>hwid.h, 116</td>
<td></td>
</tr>
<tr>
<td>hw_dispatch</td>
<td>interp.c, 158</td>
<td></td>
</tr>
<tr>
<td>HW_ENCODER</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>hw_freq_valt</td>
<td>types.h, 146</td>
<td></td>
</tr>
<tr>
<td>HW_IR01</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_IR02</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_IR03</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
<tr>
<td>HW_IR04</td>
<td>hwid.h, 115</td>
<td></td>
</tr>
</tbody>
</table>

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
| HW_BUMP02 | 116 | HW_SONAR12 | 115 |
| HW_BUMP03 | 116 | HW_SONAR13 | 115 |
| HW_BUMP04 | 116 | HW_SONAR14 | 115 |
| HW_BUMP05 | 116 | HW_SONAR15 | 115 |
| HW_BUMP06 | 116 | HW_SONAR16 | 115 |
| HW_BUMP07 | 116 | HW_VEL | 115 |

- hwidths
  - logger.cpp, 233
  - logtool.cpp, 237

- id
  - robot_handle, 39

- help
  - interp.c, 159

- in_file
  - interp.c, 159
  - include/robot.h, 88
  - include/robot/behavior.h, 90
  - include/robot/constants.h, 96
  - include/robot/devfs.h, 100
  - include/robot/handle.h, 109
  - include/robot/hwid.h, 112
  - include/robot/mclib.h, 117
  - include/robot/motors.h, 119
  - include/robot/net.h, 123
  - include/robot/sensors.h, 126
  - include/robot/seq.h, 130
  - include/robot/sys.h, 138
  - include/robot/time.h, 141
  - include/robot/types.h, 144
  - include/robot/util.h, 148

- InfoDialog, 23
  - InfoDialog, 23
  - ~InfoDialog, 23
  - InfoDialog, 23

- infraredStatus
  - MsgWindow, 31

- init_complete
  - robot_handle, 40

- init_devs
  - devices.c, 80
  - robotdrv.c, 82

- INIT_DIR
  - devices.c, 80

- init_dir_fail
  - devs_local.h, 183

- init_module
  - robotdrv.c, 82

- init_poll_list
  - logger.cpp, 231

- init_queue
  - sequencer.cpp, 75

- init_search_path
librobot/net.c, 196
librobot/sensors.c, 168
errno, 172
robot_force_reset_all_sensors, 169
robot_get_all_jump, 169
robot_get_all_lir, 169
robot_get_all_sonar, 170
robot_get_ljump, 170
robot_get_lir, 170
robot_get_sonar, 170
robot_set_all_lir_freq, 170
robot_set_all_sonar_freq, 171
robot_set_lir_freq, 171
robot_set_sonar_freq, 171
sensors_init, 171
sensors_shutdown, 171
librobot/seq.c, 202
librobot/sys.c, 210
librobot/time.c, 214
librobot/util.c, 216
listen_init
netrobotd.cpp, 245
robot_simulator.cpp, 258
load
medib.c, 190
load_behavior
sequencer.cpp, 76
load_required.bhv
sequencer.cpp, 76
local_robot
handle.c, 189
sys.c, 213
lock.cpp
main, 271
lockowner
robot_dev.J, 36
lockprio
robot_dev.J, 36
log_forever
logger.cpp, 231
log_writer.J, 26
~log_writer.J, 26
close, 26
log_writer.J, 26
open, 26
size, 27
write, 27
logger.cpp
add_ctrl, 231
add_dir, 231
entry, 232
filename, 232
hwids, 233
init_poll_list, 231
llhelp, 233
log_forever, 231
main, 232
opt, 233
OPT_RUN_BG, 231
OPT_TYPE_CHANGE, 231
OPT_TYPE_CTL, 231
OPT_TYPE_STREAM, 231
parse_command_line, 232
parse_hwid_list, 232
quit, 232
read_and_log, 232
robot_log, 233
logtool.cpp
begin, 237
der, 237
tentry, 237
filename, 237
hwids, 237
llhelp, 237
main, 236
opt, 238
OPT_TYPE_CHANGE, 236
OPT_TYPE_CTL, 236
OPT_TYPE_STREAM, 236
parse_command_line, 236
parse_hwid_list, 236
printCtlEntry, 237
printDataEntry, 237
printEntryCommon, 237
quit, 237
robot_log, 238
look_for_change
robot_simulator.cpp, 258
loop_forever
interp.c, 158
llhelp
logtool.cpp, 237

m
simrobot.cpp, 263
main
behavior.cpp, 267
bhv_main.c, 66
bhv.J, 17
bhvctl.cpp, 69
drive_in_circle.cpp, 268
forward.cpp, 269
freq.cpp, 270
interp.c, 158
lock.cpp, 271
logger.cpp, 232
logtool.cpp, 236
INDEX

mc_shutdown, 28
mc_start_frame, 28
mc_lib_unload
mc_lib.c, 191
mc_lib.h, 118
mc_pid.c
clamp, 228
mc_do_control, 228
mc_get_left_kd, 228
mc_get_left_ki, 229
mc_get_left_kp, 229
mc_get_right_kd, 229
mc_get_right_ki, 229
mc_get_right_kp, 229
mc_init, 229
mc_set_left_kd, 229
mc_set_left_ki, 229
mc_set_left_kp
mc_pid.c, 229
mc_set_odometry
mc.h, 220
mc_common.c, 224
mc_lib.c, 28
mc_set_odometry_func_t
mc_lib.h, 118
mc_set_right_kd
mc_pid.c, 229
mc_set_right_ki
mc_pid.c, 229
mc_set_right_kp
mc_pid.c, 229
mc_set_velocity
mc.h, 220
mc_common.c, 225
mc_lib.c, 28
mc_set_velocity_func_t
mc_lib.h, 118
mc_shutdown
mc.h, 221
mc_common.c, 225
mc_lib.c, 28
mc_shutdown_func_t
mc_lib.h, 118
mc_start_frame
mc.h, 221
mc_common.c, 225
mc_lib.c, 28
mc_start_frame_func_t
mc_lib.h, 118
mc_lib.c
load, 190
mc_lib_load, 191
mc_lib_unload, 191
mc_lib.h
mc_do_control_func_t, 118
mc_get_velocity_func_t, 118
mc_init_func_t, 118
mc_lib_load, 118
mc_lib_unload, 118
mc_set_odometry_func_t, 118
mc_set_velocity_func_t, 118
mc_shutdown_func_t, 118
mc_start_frame_func_t, 118
mid
robot_msg, 41
misc/logger/logger.c, 230
misc/logger/logger.h, 234
misc/logger/logtool.cpp, 235
misc/logger/pars_hwid_list.cpp, 239
misc/logger/reader.cpp, 240
misc/logger/writer.cpp, 241
misc/logger/writer.h, 242
misc/netrobot/netrobot.cpp, 243
misc/netrobot/nettool.cpp, 248
misc/simulator/bumpsensor.h, 253
misc/simulator/robot_simulator.cpp, 254
misc/simulator/sensorinfo.h, 261
misc/simulator/simrobot.cpp, 262
misc/simulator/simrobot.h, 264
misc/simulator/world.cpp, 265
misc/simulator/world.h, 266
misc/tests/behavior.cpp, 267
misc/tests/drive_in_circle.cpp, 268
misc/tests/forward.cpp, 269
misc/tests/freq.cpp, 270
misc/tests/lock.cpp, 271
misc/tests/sonar.cpp, 272
misc/tests/stop.cpp, 273
misc/tests/stop_sensors.cpp, 274
misc/tests/time.cpp, 275
MKUINT16
interp.c, 157
MODULE_AUTHOR
robotdrv.c, 82
MODULE_DESCRIPTION
robotdrv.c, 82
MODULE_LICENSE
robotdrv.c, 82
mom.cpp
main, 284
MomCanvasView, 29
   MomCanvasView, 29
MomCanvasView
   ~MomCanvasView, 29
   contentsMouseMoveEvent, 29
   contentsMousePressEvent, 29
MomCanvasView, 29
   positionChanged, 29
   xOffset, 29
   yOffset, 29
MomWindow, 30
   MomWindow, 31
MomWindow
   ~MomWindow, 31
   about, 31
   disconnect, 31
   discoveryWindow, 31
   getHandle, 31
   getRInfo, 31
   haltRobot, 31
   infraredStatus, 31
   ipWindow, 31
   MomWindow, 31
   moveRobot, 31
   robotStatus, 31
   sonarStatus, 31
   updateScreen, 31
   updateStatus, 31
motors.c
   errno, 195
   robot_get_odometry, 193
   robot_get_velocity, 193
   robot_lock_motors, 193
   robot_rotate, 193
   robot_set_odometry, 194
   robot_set_velocity, 194
   robot_translate, 194
   robot_unlock_motors, 194
motors.h
   robot_get_odometry, 120
   robot_get_velocity, 120
   robot_lock_motors, 120
   robot_rotate, 120
   robot_set_odometry, 121
   robot_set_velocity, 121
   robot_translate, 121
   robot_unlock_motors, 121
move
   SimRobot, 53
moveRobot
   MomWindow, 31
mpProblem, 32
   mpProblem, 32
mpProblem
   getNextLog, 32
   getNextRobot, 32
   mpProblem, 32
   problemName, 32
   w, 32
MSEC_PER_SEC
   time.h, 142
msg
   netrobotd.cpp, 246
msg_buf
   netrobotd.cpp, 246
   robot_simulator.cpp, 260
MSG_ERROR
   net.h, 124
msg_error
   net.h, 201
   netrobotd.cpp, 247
   robot_simulator.cpp, 260
MSG_GET_LOCK_OWNER
   net.h, 124
msg_get_lock_owner
   devs.net.c, 187
   net.h, 201
MSG_INIT
   net.h, 124
msg_init
   devs.net.c, 187
   net.h, 201
MSG_LOCK_CTL
   net.h, 124
msg_lock_ctl
   devs.net.c, 187
   net.h, 201
MSG_PING
   net.h, 124
msg_ping
   net.h, 201
MSG_SHUTDOWN
   net.h, 124
msg_shutdown
   devs.net.c, 187
   net.h, 201
MSG_UNLOCK_CTL
   net.h, 124
msg_unlock_ctl
   devs.net.c, 187
   net.h, 201
msg_wait_change
   devs.net.c, 187
   net.h, 201
MSG_WAIT_FOR_CHANGE
   net.h, 124
MSG_BUF_BASE_SZ
INDEX

seq.h, 132
my_type
  seq_msgbuf.h, 44
my_gid
  bhv_main.c, 67
  bhv_tcp.c, 71
  seq.c, 209
myalarm
  sequencer.cpp, 77
name
  Obstacle, 33
  robot_handle.h, 40
  World, 57
net.c
  errno, 201
  find_list.h, 198
  msg_error, 201
  msg_get_lock_owner, 201
  msg_init, 201
  msg_lock_ctl, 201
  msg_ping, 201
  msg_shutdown, 201
  msg_unlock_ctl, 201
  msg_wait_change, 201
  net_connect_tcp, 198
  net_init, 199
  net_read_msg, 199
  net_shutdown, 199
  net_write_msg, 199
  robot_get_ip_str, 200
  robot_net_find, 200
  robot_net_set_timeout, 200
net.h
  MSG_ERROR, 124
  MSG_GET_LOCK_OWNER, 124
  MSG_INIT, 124
  MSG_LOCK_CTL, 124
  MSG_PING, 124
  MSG_SHUTDOWN, 124
  MSG_UNLOCK_CTL, 124
  MSG_WAIT_FOR_CHANGE, 124
  robot_get_ip_str, 124
  robot_net_find, 124
  robot_net_set_timeout, 125
net_connect_tcp
  devfs_net.c, 186
  net.c, 198
net.h
  bhv_main.c, 66
  net.c, 199
  sys.c, 211
net_read_msg
  devfs_net.c, 187
net.c, 199
netrobotd.cpp, 245
  robot_simulator.cpp, 258
net_shutdown
  net.c, 199
  sys.c, 211
net_write_error
  netrobotd.cpp, 245
  robot_simulator.cpp, 258
net_write_msg
  devfs_net.c, 187
  net.c, 199
  netrobotd.cpp, 246
  robot_simulator.cpp, 258
netrobotd.cpp
  client, 246
  do_robot_init, 245
  errno, 246
  listen_init, 245
  main, 245
  make_daemon, 245
  msg, 246
  msgbuf, 246
  msg_error, 247
  net_read_msg, 245
  net_write_error, 245
  net_write_msg, 246
  nr dhcp, 247
  OPT_HAVE_LOG, 245
  OPT_NO_UDP, 245
  OPT_RUN_BG, 245
  parse_command_line, 246
  printf, 245
  robot_id, 247
  robot_name, 247
  robot_name_len, 247
  setup_sockaddr, 246
  tcp_hand l e, 246
  tcp_hand le_msg, 246
  udp_han dle, 246
  zombie, 246
NO_ROBOT_INIT
  behavior.h, 93
nr dhcp
  netrobotd.cpp, 247
nr help
  nrtool.cpp, 252
nrrobots
  nrtool.cpp, 252
nrtool.cpp
  cleanup_robot, 250
  console, 250
  err_args, 252
  err_robot, 252
errno, 252
find_robot, 250
h_find, 250
h_gb, 251
h_gi, 251
h_go, 251
h_gs, 251
h_gv, 251
h_ip, 251
h_name, 251
h_rotate, 251
h_saif, 251
h_sasf, 251
h_sif, 251
h_so, 251
h_srv, 251
h_ssf, 251
h_status, 251
h_stop, 251
h_stv, 251
h_sv, 251
h_translate, 251
h_unset, 251
handlers, 252
main, 251
nhelp, 252
nrobots, 252
remove_comments, 251
rip, 252
robot, 252
robots, 252
run, 251
setup_robot, 252
version, 252
NUMDEVICE_DIRS
devfs.h, 104
NUM_DEVICES
devfs.h, 104
NUM_DIRS
devices.c, 80
NUM_SAMPLES
cp_2id.c, 228
Obstacle, 33
name, 33
Obstacle, 33
obstacles
World, 57
odom
interp.h, 164
interp/sensors,c, 167
odom_val

types.h, 147
odometryChanged

SimRobot, 53
on_inhibit
  bhv.h, 17
on_uninhibit
  bhv.h, 17
open
  log_writer.h, 26
operator>>
  world.cpp, 265
opt
  logger.cpp, 233
  logtool.cpp, 238
OPT_DEVFS
  interp.c, 158
OPT_HAVE_LOG
  interp.c, 158
  netrobotd.cpp, 245
  robot_simulator.cpp, 257
  sequencer.cpp, 75
OPT_HIGH_PRIOR
  interp.c, 158
OPT_HW
  interp.c, 158
OPT_NO_UDP
  netrobotd.cpp, 245
  robot_simulator.cpp, 257
OPT_RUN_BG
  interp.c, 158
  logger.cpp, 231
  netrobotd.cpp, 245
  robot_simulator.cpp, 257
  sequencer.cpp, 75
OPT_TYPE_CHANGE
  logger.cpp, 231
  logtool.cpp, 236
OPT_TYPE_CTL
  logger.cpp, 231
  logtool.cpp, 236
OPT_TYPE_STREAM
  logger.cpp, 231
  logtool.cpp, 236
out_cmd
  freq.c, 155
  interp.c, 159
output_t
  bhv_main.c, 65
outputs
  bhv_main.c, 67
owner
  freqreqval.h, 22
  ownerval.h
    types.h, 147
  P

Generated on Wed Aug 20 15:56:06 2008 for robots-all by Doxygen
acceleration, 260
displayBumps, 260
displayIRs, 260
displayPaths, 260
displaySonsars, 260
do-physics, 258
elapsed, 260
erino, 260
G, 260
get_all_bump, 258
get_all_IR, 258
get_all_sonar, 258
get_bump, 258
get_SR, 258
get_odometry, 258
get_sonar, 258
get_velocity, 258
keyboardHandler, 258
listen_init, 258
look_for_change, 258
main, 258
msg_buf, 260
msg_error, 260
net_read_msg, 258
net_write_error, 258
net_write_msg, 258
OPT_HAVE_LOG, 257
OPT_NO_UDP, 257
OPT_RUN_BG, 257
P, 260
printf, 257
robot_id, 260
robot_name, 260
robot_name_len, 260
RobotList, 257
robots, 260
set_all_IR_frequencies, 259
set_all_sonar_frequency, 260
set_IR_frequency, 260
set_odometry, 260
set_sonar_frequency, 260
set_velocity, 260
setup_sockaddr, 260
socks, 260
tcp_handle, 260
tcp_handle_msg, 260
toDelete, 260
total_elapsed, 260
tv, 260
waitForChange, 260
workspace, 260
robot_sleep

time.c, 215
time.h, 142

ROBOT_SONAR_MULTIPLIER
constants.h, 98
robot_sqdist
util.c, 216
util.h, 149
ROBOT_THRESH_IR
constants.h, 98
ROBOT_THRESH_ODOM_THETA
constants.h, 98
ROBOT_THRESH_ODOM_X
constants.h, 98
ROBOT_THRESH_ODOM_Y
constants.h, 98
ROBOT_THRESH_ROTATE
constants.h, 98
ROBOT_THRESH_SONAR
constants.h, 98
ROBOT_THRESH_TRANSLATE
constants.h, 98
ROBOT_THRESH_VEL_X
constants.h, 98
ROBOT_THRESH_VEL_W
constants.h, 98
robot_time_to_float_ms
time.h, 142
robot_time_to_float_sec
time.h, 142
robot_time_to_timeval
time.c, 215
robot_time_t
types.h, 147
robot_translate
motors.c, 194
motors.h, 121
robot_unlock_motors
motors.c, 194
motors.h, 121
ROBOT_WHEEL_RADIUS
constants.h, 99
ROBOT_WHEEL_RADIUS_L
constants.h, 99
ROBOT_WHEEL_RADIUS_R
constants.h, 99
robot_write
syscalls.c, 86
robotdrv.c
cleanup_levs, 82
cleanup_module, 82
init_levs, 82
init_module, 82
MODULE_AUTHOR, 82
MODULE_DESCRIPTION, 82
MODULE_LICENSE, 82
robotdrv.h
CHANGE, 84
CTRL, 84
CURRENT, 84
MAX_TYPE, 84
robot_devices, 84
robot_joys, 84
STREAM, 84
RobotInfo, 42
RobotInfo
~RobotInfo, 42
getBoundaryIR, 42
getBoundarySonar, 42
getIRLine, 42
getSonarPolygon, 42
RobotInfo, 42
RobotList
robot_simulator.cpp, 257
robots
nrtool.cpp, 252
robot_simulator.cpp, 260
robotStatus
MomWindow, 31
ROT
mc.h, 219
run
nrtool.cpp, 251
search_dirs
sequencer.cpp, 78
selectionChanged
DiscoveryDialog, 21
sem
robot_dev.h, 37
send_err
sequencer.cpp, 77
sens_init
interp.h, 162
interp/sensors.c, 166
sens_update_jump
interp.h, 162
interp/sensors.c, 166
sens_update_ir
interp.h, 163
interp/sensors.c, 166
sens_update_motors
interp.h, 163
interp/sensors.c, 166
sens_update_sonar
interp.h, 163
interp/sensors.c, 167
SensorInfo, 43
SensorInfo, 43
SensorInfo
SensorInfo, 43
sensors.h
robot_force_reset_all_sensors, 127
robot_get_all_jump, 127
robot_get_all_ir, 127
robot_get_all_sonar, 128
robot_get_jump, 128
robot_get_ir, 128
robot_get_sonar, 128
robot_set_all_ir_freq, 128
robot_set_all_sonar_freq, 129
robot_set_ir_freq, 129
robot_set_sonar_freq, 129
sensors_init
librobot/sensors.c, 171
sys.c, 213
sensors_shutdown
librobot/sensors.c, 171
sys.c, 213
seq.c
erro, 209
MAX_BHV, 204
my_pid, 209
seq_attach, 204
seq_cleanup, 204
seq_connect, 205
seq_disconnect, 205
seq_get, 205
seq_get_my_handle, 205
seq_inhibit, 206
seq_inhibit_all, 206
seq_init, 206
seq_load, 206
seq_load_args, 207
seq_load_net, 207
seq_pid, 209
seq_send, 207
seq_uninhibit, 208
seq_uninhibit_all, 208
seq_unload, 208
seq_unload_all, 209
seq.h
BHV_CONNECT, 132
BHV_DATA, 132
BHV_DISCONNECT, 132
BHV_INHIBIT, 132
BHV_INIT, 132
BHV_UNINHIBIT, 132
MSGBUF_BASE_SZ, 132
ROBOT_MAX_BHV_DATA_SIZE, 132
ROBOT_SEQ_QUEUE_KEY, 132
SEQ_ATTACH, 132

Generated on Wed Aug 20 15:56:06 2003 for robots-all by Doxygen
seq_attach, 132
seq_connect, 133
seq_disconnect, 133
SEQ_ERR, 132
seq_get, 133
seq_get_my_handle, 134
seq_inhibit, 134
seq_inhibit_all, 134
SEQ_LIST, 132
SEQ_LOAD, 132
seq_load, 134
seq_load_args, 135
seq_load_net, 135
seq_send, 135
seq_uninhibit, 136
seq_uninhibit_all, 136
SEQ_UNLOAD, 132
seq_unload, 136
seq_unload_all, 137
SEQ_ATTACH
seq.h, 132
seq_attach
seq.c, 204
seq.h, 132
seq_cleanup
bhvmain.c, 66
bhvctLcpp, 70
seq.c, 204
sys.c, 213
seq_connect
seq.c, 205
seq.h, 133
seq_disconnect
seq.c, 205
seq.h, 133
SEQ_ERR
seq.h, 132
seq_get
seq.c, 205
seq.h, 133
seq_get_my_handle
seq.c, 205
seq.h, 134
seq_inhibit
seq.c, 206
seq.h, 134
seq_inhibit_all
seq.c, 206
seq.h, 134
seq_init
bhvmain.c, 66
bhvctLcpp, 70
seq.c, 206
sys.c, 213

SEQ_LIST
seq.h, 132
SEQ_LOAD
seq.h, 132
seq_load
seq.c, 206
seq.h, 134
seq_load_args
seq.c, 207
seq.h, 135
seq_load_net
seq.c, 207
seq.h, 135
seq_msgbuf, 44
cmd, 44
data, 44
mtype, 44
seq_msgid
bhvmain.c, 67
bhvctLcpp, 71
seq.c, 209
seq_send
seq.c, 207
seq.h, 135
seq_uninhibit
seq.c, 208
seq.h, 136
seq_uninhibit_all
seq.c, 208
seq.h, 136
SEQ_UNLOAD
seq.h, 132
seq_unload
seq.c, 208
seq.h, 136
seq_unload_all
seq.c, 209
seq.h, 137
seqhelp
sequencer.cpp, 78
sequencer.cpp
behaviors, 77
erro, 77
find_and_load, 75
find_behavior, 75
handle_attach, 75
handle_list, 75
handle_load, 75
handle_unload, 75
init_queue, 75
init_search_path, 76
ip, 77
load_behavior, 76
load_required bhv, 76
null
setW, 55
SimRobot, 48
simrobot.cpp
alpha, 263
defaultSeed, 263
dist, 263
m, 263
P, 263
pmRandGenSeed, 263
pmRandMax, 263
q, 263
r, 263
simulator_constants.h
HEIGHT, 292
WIDTH, 292
xCenter, 292
yCenter, 292
size
log_writer, 27
sock
robot_handle, 40
sockaddr
robot_handle, 40
socks
robot_simulator.cpp, 260
sonar
interp/sensors.c, 167
sonar.cpp
main, 272
sonar_time_val, 147
types.h, 147
sonar_val, 147
SonarDialog, 56
SonarDialog, 56
SonarDialog
~SonarDialog, 56
getInfo, 56
SonarDialog, 56
sonarStatus
MomWindow, 31
start_bhv
bhvclient.cpp, 70
std, 12
STEPS_MULT
mc_common.c, 224
stop.cpp
main, 273
stop_sensors.cpp
main, 274
STREAM
robotdrv.h, 84
stream
devfs_set_t, 20
INDEX

343

test_cleanup
test.c, 60
test_inhibit
test.c, 60
test_main
test.c, 60
test_ondata
test.c, 60
test_uninhibit
test.c, 60
testsub.c
 blv_init, 61
test, 62
time.c
 robot_alarm, 215
 robot_get_time, 215
 robot_sleep, 215
 robot_time_to_timeval, 215
timeval_to_robot_time, 215
time.cpp
 main, 275
time.h
 MSEC_PER_SEC, 142
 robot_alarm, 142
 robot_get_time, 142
 robot_sleep, 142
 robot_time_to_float_ms, 142
 robot_time_to_float_sec, 142
 USEC_PER_MSEC, 142
 USEC_PER_SEC, 142
timeout
 robot_handle,, 40
timeval_to_robot_time
time.c, 215
to
 blv_connection,, 13
to_dev
data_len,, 19
to_key
 blv_connection,, 13
toDelete
 robot_simulator.cpp, 260
total_elapsed
 robot_simulator.cpp, 260
TRANS
 mc.h, 219
tv
 robot_simulator.cpp, 260
 TWOPLJR
 mc_common.c, 224
 TWOPLJR
 mc_common.c, 224
type
 robot_dev,, 37
types.h
 bump_bitfield_val,, 146
 bump_val,, 146
 encoder_val,, 146
 freq_val,, 146
 hw_freq_val,, 146
 ir_val,, 146
 ir_voltage_val,, 147
 odom_val,, 147
 owner_val,, 147
 pwm_val,, 147
 ROBOT_BYTE_ORDER, 146
 robot_id,, 147
 robot_time_usec,, 147
 sonar_time_val,, 147
 sonar_val,, 147
 vel_val,, 147
udp_handle
 netrobotd.cpp, 246
updateScreen
 MomWindow, 31
updateStatus
 MomWindow, 31
USEC_PER_MSEC
time.h, 142
USEC_PER_SEC
time.h, 142
util.c
 robot_dist, 216
 robot_sqdist, 216
util.h
 assert, 149
 robot_deg2rad, 149
 robot_dist, 149
 robot_dprintf, 149
 robot_fix_byte_order, 149
 robot_rad2deg, 149
 robot_sqdist, 149
vel
 interp/sensors.c, 167
vel_correct
 mc.h, 221
 mc_common.c, 226
vel_current
 mc.h, 222
 mc_common.c, 226
vel_current_xw
 mc.h, 222
 mc_common.c, 226
vel_val,, 147
 types.h, 147
version
nrtool.cpp, 252

w
  mpProblem, 32
  waitForChange
    robot_simulator.cpp, 260
  went
    robot_dev.t, 37
WIDTH
  simulator_constants.h, 292
workspace
  robot_simulator.cpp, 260
World, 57
  boundary, 57
  name, 57
  obstacles, 57
  World, 57
world.cpp
  operator>>, 265
world.h
  MAX_PROBLEM_NAME_LEN, 266
write
  log_writer.t, 27

x
  BumpSensor, 18
  Point, 34
  PointTheta, 35
xCenter
  simulator_constants.h, 292
xOffset
  MomCanvasView, 29

y
  BumpSensor, 18
  Point, 34
  PointTheta, 35
yCenter
  simulator_constants.h, 292
yOffset
  MomCanvasView, 29

zombie
  netrobotd.cpp, 246
  sequencer.cpp, 77