```
// pid.hpp: Header file for the pid controller and all of it's assets
// Copyright (C) 2017 Ethan Wells
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

#ifndef PID_HPP
#define PID_HPP

#include "lift.hpp"

/** Consists of pid, and all subcomponents, etc */
namespace pid {
  /** Maximum value for the drive */
  static const int DRIVE_MAX = 127;
  /** Minimum value for the drive */
  static const int DRIVE_MIN = -127;
  /** Limit for the integral value */
  static const int INTEGRAL_LIMIT = 50;
  /** p value */
  extern float Kp;
  /** i value */
  extern float Ki;
  /** d value */
  extern float Kd;
  /** Default precision for waiting on pid to reach value */
  extern unsigned int default_precision;
  /** Whether or not each side of the drive's pid is enabled, in the order of
   * left to right */
  extern bool enabled[2];
  /** A class for a single position on the drive */
  struct pos_t {
    long left;
    long right;
    void request(void);
    pos_t(long left, long right);
```

```
    bool operator=(pos_t pos);
    pos_t operator+(pos_t pos);
    pos_t operator-(pos_t pos);
  };

  /** Enables all pid */
  void enable(void);

  /** Disables all pid */
  void disable(void);

  /** Task to manage pid */
  void controller(void* none);

  /** Initialize pid. Call in initialize() */
  void init(void);

  /** Stops the pid task */
  void stop(void);

  /** (Re)starts the pid task */
  void go(void);

  /** Gets the current position */
  pos_t get(void);

  /** Requests values for the left and right side of the drive */
  void request(long l, long r);
  /** Requests a specific position */
  void request(pos_t pos);

  /** Wait until pid reaches specified precision, for no longer than the
   * specified
   * blockTime. If 0 is passed to blockTime, it will wait indefinately until
   * the requested values are met */
  void wait(unsigned long precision, unsigned long blockTime);
  /** TaskHandle for the pid task */
  extern TaskHandle pidHandle;
} // namespace pid

#endif /* end of include guard: PID_HPP */
```