Drop-off Locker

User Manual and Design Schematic

Soft Development

Open Source Projects

By Jerry Zaatri

Index

About Drop-off Locker……. 1

General use guide……. 1

Administrator guide……. 2

Troubleshoot and repair guide…. 3

Circuit Design…….. 4

Source Code……. 5

Fabrication Notes……. TBD

Upgrade Notes……. TBD

Contact Info……. TBD

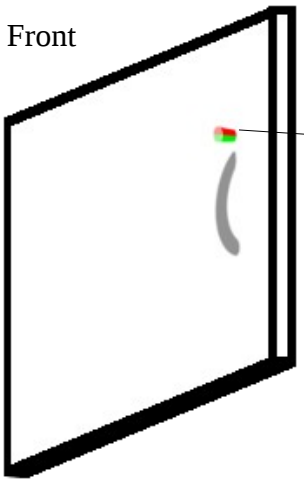
About:

Drop-off Locker is designed to combat package theft. The cabinet is designed to allow delivery personnel to place packages in and lock it without the need of a key. Obtaining packages is accomplished with access to the administrator control box to unlock doors.

General Use:

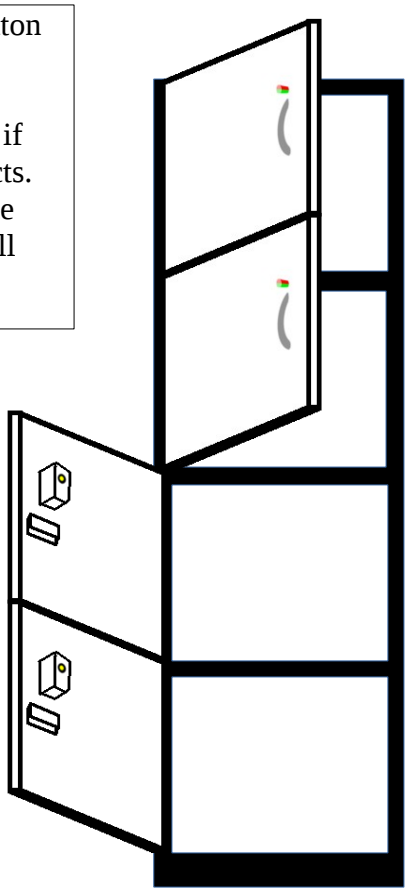
Availability of the locker is indicated by the clear acrylic push button. When lit green, the locker is available and when red, occupied.

Front

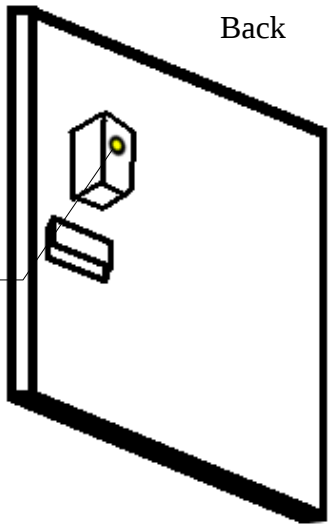


When available, pressing the button will release the lock.

<u>NOTE:</u> solenoid locks can jam if door is pulled before lock retracts. It's recommended to leave a note for delivery personnel to not pull until solenoid retracts.



Back

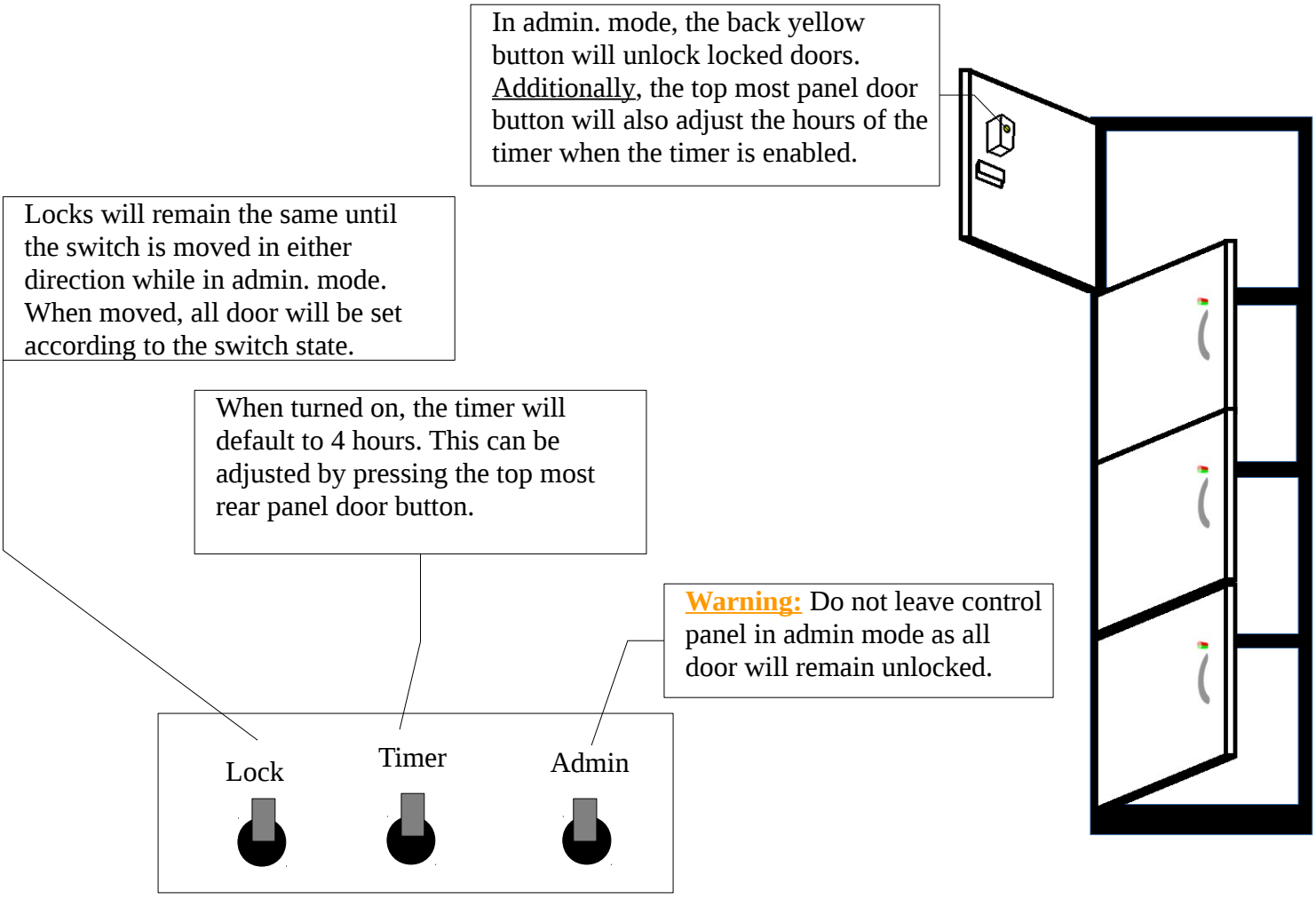


The rear yellow button will lock the door when pressed. Upon locking the front button will light up red.

<u>Recommendation:</u> if possible provide a phone number in the event someone accidentally locks something in the cabinet by accident.

Administrator Use:

As administrator, you will have the ability to lock and unlock all or individual doors as well as setup a timer until doors become locked. Within the control box are 3 switches: administrator mode, timer enable, and lock toggle. When in administrator mode, all doors are accessible and the lights will indicate the timer state instead of the lock state. When all red (no lights in control box) the timer is off, for each lit LED in the control box, an hour (or less) remains.



Troubleshoot and Repair:

TBD

Circuit Design:

TBD

Source Code:

```
#define LED_D 13 //bottom
#define LED_C 12
#define LED_B 11
#define LED_A 10 //top
#define B_EN_D 9
#define B_EN_C 8 //back button
#define B_EN_B 7 //front button
#define B_EN_A 6 //priority set(control panel)

#define LCK_D 5 //bottom
#define LCK_C 4
#define LCK_B 3
#define LCK_A 2 //top
#define B_IN_D 3
#define B_IN_C 2
#define B_IN_B 1
#define B_IN_A 0
#define LOCKED 0
#define UNLOCKED 1
#define TAKEN 1
#define OPEN 0
#define LOOPS_TO_HR 36000
#define WATHCHDOG_LIMIT 50 //5 seconds limit

#define ACTIVE_LOW_RELAY

struct QuadState
{
  bool s[4];
};
struct QuadCount
{
  int c[4];
};

//lock access (TAKEN/OPEN)
QuadState gLockLed;
//current state of lock(LOCKED/UNLOCKED)
QuadState gLock;
//button state (per poll per set)
QuadState gBtn;

//avoid holding solenoid lock if button jams in
QuadState gWatchDogHit;
QuadCount gWatchDogCnt;
```

```
//how many hours till locking
unsigned short gLkTmr;
//no available pins to allow for interupt based timer.
//keep track of hours by counting loops
unsigned int gHrTmr;
//perserve last state of button/switch to trigger event on changed state
bool gLastTToggle; //for changing time
bool gLastLToggle; //for locking/unlocking all in admin mode
bool gLastAMode; //check if admin mode was just entered

//TODO: add watchdog timer for held lock buttons

void setup()
{
  // Initialize the digital pin as an output.
  for(int i = 6; i < 14; i++)
  {
    pinMode(i, OUTPUT);
    digitalWrite(i, 0);
  }
  for(int i = 0; i < 4; i++)
  {
    pinMode(i+LCK_A, OUTPUT);
#ifdef ACTIVE_LOW_RELAY
    digitalWrite(i+LCK_A, 1);
#else
    digitalWrite(i+LCK_A, 0);
#endif
  }

  Serial.begin(9600);
  Serial.println("LOCKER STARTING UP....");

/*  Debug code
  //startup debug blink
  for(int d = 0; d < 4; d++)
  {
    for(int t = 0; t < 4; t++)
    {
      delay(500);
      for(int i = LED_A; i <= LED_D; i++) { digitalWrite(i, LOCKED); }
      delay(500);
      debugBtnShow(d);
      //for(int i = LED_A; i <= LED_D; i++) { digitalWrite(i, UNLOCKED); }
    }
  }
*/
```

```
  //set default values
  for(int i = 0; i < 4; i++)
  {
    gLock.s[i] = LOCKED;
    gLockLed.s[i] = TAKEN;
  }

  gLkTmr = 4;

  gLastTToggle = false;
  gLastLToggle = false;
  gLastAMode = false;

  gHrTmr = LOOPS_TO_HR;

  //start watchdog with clean slate (no open locks)
  watchdogTick();

  showLockState();
  Serial.println("SYSTEM INITIALIZED....");
}

void loop()
{
  bool adminMode = false;
  bool activeTimer = false;
  bool toggleLock = false;

//button routines================================================================
  //admin setup-----------------------------------------------------------------
  delay(25);
  readBtn(0);
  adminMode = gBtn.s[0];
  activeTimer = gBtn.s[1];
  toggleLock = gBtn.s[2];

  if(activeTimer && adminMode && (gLkTmr == 0))
  {
    gLkTmr = 4;
  }
  else if(!activeTimer)
  {
    gLkTmr = 0;
  }

  if((adminMode) && (toggleLock != gLastLToggle) && gLastAMode)
  {
    setAllState(toggleLock ? TAKEN : OPEN);
  }
  gLastLToggle = toggleLock;
  gLastAMode =  adminMode;
```

```
  //front button-----------------------------------------------------------------
  delay(25);
  readBtn(1);
  if(adminMode)
  {
    for(int i = 0; i < 4; i++)
    {
      gLock.s[i] = gBtn.s[i];
    }
  }
  else
  {
    for(int i = 0; i < 4; i++)
    {
      gLock.s[i] = gBtn.s[i] && (gLockLed.s[i] == OPEN);
    }
  }

  //rear button------------------------------------------------------------------
  delay(25);
  readBtn(2);
  for(int i = 0; i < 4; i++)
  {
    if(adminMode) { gLockLed.s[i] &= !gBtn.s[i]; }
    else { gLockLed.s[i] |= gBtn.s[i]; }
  }
  if(adminMode)
  {
    if(activeTimer && !gLastTToggle && gBtn.s[0])
    {
      toggleTimer();
    }
    gLastTToggle = gBtn.s[0];
  }
  else { gLastTToggle = false; }

  //for future improvements/debug -----------------------------------------------
  delay(25);
  readBtn(3);
  //add code as needed

//show results ===================================================================
  //display state
  if(adminMode)
  {
    showTimerState();
  }
  else
  {
    showLockState();
  }

  watchdogTick();
```

```
  triggerLock();

  //check for timer event (reaching an hour)
  gHrTmr--;
  if(!gHrTmr)
  {
    hrHit();
  }
}


void watchdogTick()
{
  for(int i = 0; i < 4; i++)
  {
    if(gWatchDogHit.s[i])
    {
      if(!gLock.s[i]) { gWatchDogCnt.c[i] = 0; }
    }
    else
    {
      gWatchDogCnt.c[i] = gLock.s[i] ? gWatchDogCnt.c[i]+1 : 0;
    }

    gWatchDogHit.s[i] = (gWatchDogCnt.c[i] >= WATHCHDOG_LIMIT);
  }
}

void showTimerState()
{
  for(int i = 0; i < 4; i++)
  {
    digitalWrite(i+LED_A, gLkTmr > i ? LOW : HIGH);
  }
}

void showLockState()
{
  for(int i = 0; i < 4; i++)
  {
    digitalWrite(i+LED_A, gLockLed.s[i]);
  }
}

void debugBtnShow(unsigned int set)
{
  readBtn(set);
  gLockLed = gBtn;
  showLockState();
}
```

```
void triggerLock()
{
  for(int i = 0; i < 4; i++)
  {
#ifdef ACTIVE_LOW_RELAY
    digitalWrite(i+LCK_A, !(gLock.s[i] && !gWatchDogHit.s[i]));
#else
    digitalWrite(i+LCK_A, (gLock.s[i] && !gWatchDogHit.s[i]));
#endif
  }
}

void readBtn(unsigned int set)
{
  //disable all sets and reset state
  for(int i = 0; i < 4; i++)
  {
    digitalWrite(B_EN_A+i, 0);
    gBtn.s[i] = false;
  }

  //if valid set, read in buttons
  if(set < 4)
  {
    digitalWrite(B_EN_A+set, 1);
    for(int i = 0; i < 4; i++) { gBtn.s[i] = analogRead(i+B_IN_A) > 111; }
  }
}

void toggleTimer()
{
  gLkTmr = (gLkTmr + 4) % 5;
}

void enableTimer(bool en)
{
  if(en)
  {
    gLkTmr = 4;
  }
  else
  {
    gLkTmr = 0;
  }
}

void setAllState(bool takenState)
{
  for(int i = 0; i < 4; i++)
  {
    gLockLed.s[i] = takenState;
  }
}
```

```
void hrHit()
{
  gHrTmr = LOOPS_TO_HR;

  if(gLkTmr)
  {
    gLkTmr--;
    if(!gLkTmr)
    {
      setAllState(TAKEN);
    }
  }
}
```