These are the additional appendices of the MQP paper “Campus Safety System” by James Beaulieu, Natasha Bonina, Lauren Lewis and Phyo Thinzar. They are supplementary material not meant for public disclosure.

# Appendix C: Microcontroller Code

## C.1 UART Communication between Microcontrollers

```
#include <msp430.h>
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P1OUT = 0x00;                             // P1.0/6 setup for LED output
  P1DIR = BIT0 + BIT6;
  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2;
  P1SEL &= ~BIT3;
  P1DIR &= ~BIT3;
  P1REN |= BIT3;
  P1OUT |= BIT3;
  UCA0CTL1 |= UCSSEL_2;                     // CLK = ACLK
  UCA0BR0 = 104;                            // 32kHz/9600 = 3.41
  UCA0BR1 = 0x00;
  UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 3
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE + UCA0TXIE;               // Enable USCI_A0 TX/RX interrupt

  __bis_SR_register(LPM3_bits + GIE);       // Enter LPM3 w/ interrupts enabled
}

// USCI A0/B0 Transmit ISR
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
  unsigned char TxByte=0;
  if (P1IN & BIT3)
    TxByte |= BIT6;
  UCA0TXBUF = TxByte;                       // Read, justify, and transmit
}

// USCI A0/B0 Receive ISR
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
        if (UCA0RXBUF & BIT6) P1OUT |= BIT6;                    // Display RX'ed charater
        else P1OUT &= ~BIT6;
}
```

## C.2 UART Communication with Xbee that creates API packet

```
#include  "msp430g2553.h"
#include "xbee.h"
#include "mystring.h"

char header[] = {0x7E, 0x00, 0x16, 0x10, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFE, 0x00,
0x00};
char packet[27];
char buffer[8]= {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
unsigned int i;
unsigned int j=0;

char checkSum(void){
        int k;
        char checksum =0x00;
        for(k=3; k<sizeof (header); k++){
                checksum += header[k];
        }
        for(k=0; k<8; k++){
                checksum += buffer[k];
        }
        checksum = 0xFF - checksum;
        return checksum;

}

void createXbeePacket(void) {
        mystrncpy(packet, header, 17);
        mystrncat(packet, buffer, 8, 17);
        char checksum =  checkSum();
        mystrncat (packet,&checksum, 1, 25);

}

void main(void)

{
        WDTCTL = WDTPW + WDTHOLD;               // Stop WDT
        BCSCTL1 = CALBC1_1MHZ;                  // Set DCO
        DCOCTL = CALDCO_1MHZ;
        P1SEL = BIT1 + BIT2 ;                   // P1.1 = RXD, P1.2=TXD
        P1SEL2 = BIT1 + BIT2 ;                  // P1.1 = RXD, P1.2=TXD

        P1OUT |= BIT3;
        P1REN |= BIT3;

        UCA0CTL1 |= UCSSEL_2;                   // SMCLK
        UCA0BR0 = 104;                          // 1MHz 9600
        UCA0BR1 = 0;                            // 1MHz 9600
        UCA0MCTL = UCBRS0;                      // Modulation UCBRSx = 1
```

```
	UCA0CTL1 &= ~UCSWRST;		// **Initialize USCI state machine**
	IE2 |= UCA0RXIE;		// Enable USCI_A0 RX interrupt

	P1IE |= 0x08; // P1.3 interrupt enabled
	P1IFG &= ~0x08; // P1.3 IFG cleared

	__bis_SR_register(LPM0_bits + GIE);	// Enter LPM0, interrupts enabled

}
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)

{
	i =0;
	buffer[j]= UCA0RXBUF;
	j++;
	if (j == 8){
		createXbeePacket();
		UC0IE |= UCA0TXIE;
		UCA0TXBUF = packet[i++];
		for (j=8; j>0; j--){
			buffer[j-1] = 0x00;
		}
	}
}

#pragma vector=USCIAB0TX_VECTOR

__interrupt void USCI0TX_ISR(void)
{
	UCA0TXBUF = packet[i++]; //send next set of hex values over UART
	if (i == sizeof packet - 1) // checks to see if entire message is sent
		UC0IE &= ~UCA0TXIE; //disable tx interrupt
}
```

## C.3 UART Communication with GPS that echoes back GPS data

```
#include  "msp430g2553.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO
  DCOCTL = CALDCO_1MHZ;
  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
 UCA0CTL1 |= UCSSEL_2;                      // SMCLK
  UCA0BR0 = 104;                            // 1MHz 9600
  UCA0BR1 = 0;                              // 1MHz 9600
  UCA0MCTL = UCBRS0;                        // Modulation UCBRSx = 1
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, interrupts enabled
}

//  Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)

{
  UCA0TXBUF = UCA0RXBUF;                    // TX -> RXed character
}
```

## C.4 Creates Xbee API packet with relevant GPS data

```
#include  "msp430g2553.h"
#include "xbee.h"
#include "mystring.h"

char header[] = {0x7E, 0x00, 0x33, 0x10, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFE, 0x00,
0x00}; //header for Xbee packet with default coordinator address and length for GPS data
char packet[56]; //Xbee API packet being created to send over UART to Xbee
char message [38]; //contains GPS payload
char GPSbuffer[45]; //buffer for GPS data being collected over UART
char trash[7]; //for parsing
unsigned int packetcounter = 0; //counter for packet being created
unsigned int g =0, comma = 0; //GPS buffer counter and comma counter
unsigned int dollarflag = 0; //dollar sign flag for storing data

char checkSum(void){ //function for creating checksum for API Xbee packet
        int k;
        char checksum =0x00;
        for(k=3; k<sizeof (header); k++){ //adds up information from header that is part of checksum
                checksum += header[k];
        }
        for(k=0; k<37; k++){ //adds up information from message for checksum
                checksum += message[k];
        }
        checksum = 0xFF - checksum; //checksum is created from last 8 bits of added value subtracted from 0xFF
        return checksum; //returns checksum to be used in packet

}

//will only be called when buffer looks like
//$GPGGA,hhmmss.sss,ddmm.mmmm,N,dddmm.mmmm,E,1
void GPSparse(void) {
        int counter;
        for(counter = 0; counter < 37; counter++){ //takes relevant GPS data to be send over Xbee network
                message[counter]  = GPSbuffer[counter+7]; //GPS data will look like:
hhmmss.sss,ddmm.mmmm,N,dddmm.mmmm,E,1
        }
}

void createXbeeTransmitPacket(void) { //combines all of information for one packet to be sent to Xbee
        mystrncpy(packet, header, 18); //copies the header to the beginning of the packet
        mystrncat(packet, message, 38, 17); //copies the message to the end of the header
        char checksum =  checkSum(); //calls checksum and stores checksum
        mystrncat (packet,&checksum, 1, 54); //adds checksum into packet

}

void main(void)

{
```

```
	WDTCTL = WDTPW + WDTHOLD;		// Stop WDT
	BCSCTL1 = CALBC1_1MHZ;		// Set DCO
	DCOCTL = CALDCO_1MHZ;
	P1SEL = BIT1 + BIT2 ;		// P1.1 = RXD, P1.2=TXD
	P1SEL2 = BIT1 + BIT2 ;		// P1.1 = RXD, P1.2=TXD
	P2SEL = ~BIT5;			// set P2.5 as input to read in CTS pin (pin 12) from Xbee
	P2DIR = ~BIT5;
	P2SEL |= BIT3;			//sets up P2.3 as output to control sleep pin for Xbee Endpoint
	P2DIR |= BIT3;
	P2OUT |= BIT3;			//sets P2.3 as high to start

	UCA0CTL1 |= UCSSEL_2;		// SMCLK
	UCA0BR0 = 104;			// 1MHz 9600
	UCA0BR1 = 0;			// 1MHz 9600
	UCA0MCTL = UCBRS0;		// Modulation UCBRSx = 1
	UCA0CTL1 &= ~UCSWRST;		// **Initialize USCI state machine**
	IE2 |= UCA0RXIE;		// Enable USCI_A0 RX interrupt

	P1IE |= 0x08; // P1.3 interrupt enabled
	P1IFG &= ~0x08; // P1.3 IFG cleared

	__bis_SR_register(LPM0_bits + GIE);	// Enter LPM0, interrupts enabled

}
//RX vector for UART
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)

{
	if (UCA0RXBUF == '$' || dollarflag == 1) { //checks to see if 1st part of string is $ or if dollarflag is set, if either
are set, then the information is stored in GPSBuffer
		dollarflag = 1;
		GPSbuffer[g] = UCA0RXBUF; //store data from UART RX line to GPSbuffer
		g++;
	}
	if (GPSbuffer[3] == 'G' && GPSbuffer[4] == 'G' && GPSbuffer[5] == 'A') { //checks to see if the first header of
the buffer is $GPPGA which is the correct data for this project
		if (UCA0RXBUF == ',') comma++; //for each comma, this counter increases by one
		if (comma == 6 && UCA0RXBUF != ',') { //if the comma count is 6, then the data is ready to be sent
over UART
			UC0IE &= ~UCA0RXIE; //disables receiving
			GPSparse(); //parse relevant GPS data to pack
			createXbeeTransmitPacket(); //build the transmit packet for XBee
			P2OUT |= ~BIT3; //sets P2.3 as low to start data being sent from Xbee
			if (!(P2IN && BIT5)){ //checks CTS pin to determine if Xbee is ready to recieve data
				UC0IE |= UCA0TXIE; //Enable transmitting
				UCA0TXBUF = packet[packetcounter++]; //Send GPS packet via XBee
			}
			for (;g > 0; g--) { //reset g to 0 and buffer to NULLs
				GPSbuffer[g] = 0x00;
			}
			GPSbuffer[g] = 0x00; //sets GPSBuffer[0] back to null
			dollarflag = 0;
```

```
                        comma = 0;
                        packetcounter=0;
                        UC0IE |= UCA0RXIE; //re-enable receive
                }
        }
        else if (GPSbuffer[3] != 'G' && g == 4) { //if the header is not $GPPGA, then reset the buffer back to null
                dollarflag = 0;
                GPSbuffer[0] = '\0';
                GPSbuffer[1] = '\0';
                GPSbuffer[2] = '\0';
                GPSbuffer[3] = '\0';
                g = 0;
        }
        else if (GPSbuffer[4] != 'G' && g == 5) { //if the header is not $GPPGA, then reset the buffer back to null
                dollarflag = 0;
                GPSbuffer[0] = '\0';
                GPSbuffer[1] = '\0';
                GPSbuffer[2] = '\0';
                GPSbuffer[3] = '\0';
                GPSbuffer[4] = '\0';
                g = 0;
        }
}

#pragma vector=USCIAB0TX_VECTOR

__interrupt void USCI0TX_ISR(void)
{
        if (!(P2IN && BIT5)){ //check CTS line to determine if Xbee is ready to recieve UART
                UCA0TXBUF = packet[packetcounter++]; //send next set of hex values over UART
                if (packetcounter == sizeof packet - 1) // checks to see if entire message is sent
                        UC0IE &= ~UCA0TXIE; //disable tx interrupt
                        P2OUT |= BIT3; //sets P2.3 as high to put Xbee back in sleep
        }
}
```

## C.5 Final Trigger Code

```
#include  "msp430g2553.h"
#include "mystring.h"
//Xbee API packet defines
char check_parent[8] = {0x7E, 0x00, 0x04, 0x08, 0x52, 0x4D, 0x50, 0x08};
char parent_16[2] = {0xFF, 0xFE};
char parent_64[8] = {0x00, 0x13, 0xA2, 0x00, 0x40, 0xB0, 0xA0, 0x69};
char coord_64 [8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
char coord_16 [2] = {0xFF, 0xFE};
char packet[59]; //Xbee API packet being created to send over UART to Xbee
unsigned int packetcounter = 0; //counter for packet being created
char RX_packet[27]; //RX Buffer for Xbee packet
unsigned int rx_counter = 0;
char letters [5] = {0x41, 0x45, 0x4D, 0x53, 0x55};
unsigned int start = 0;
//GPS defines
unsigned int g =0, comma = 0; //GPS buffer counter and comma counter
unsigned int dollarflag = 0; //dollar sign flag for storing data
char GPS_payload [37]; //contains GPS payload
char GPSbuffer[45]; //buffer for GPS data being collected over UART
unsigned int E_flag = 0;
unsigned int timer_flag;
//will only be called when buffer looks like
//$GPGGA,hhmmss.sss,ddmm.mmmm,N,dddmm.mmmm,E,1
void GPSparse(void) {
        unsigned int counter;
        for(counter = 0; counter < 37; counter++){ //takes relevant GPS data to be send over Xbee network
                GPS_payload[counter]  = GPSbuffer[counter+7]; //GPS data will look like:
hhmmss.sss,ddmm.mmmm,N,dddmm.mmmm,E,1
        }
}
char checkSum (int p_length){
        unsigned int k;
        char checksum = 0x00;
        for(k=3; k<(20 + p_length); k++){
                checksum += packet[k];
        }
        checksum = 0xFF - checksum;
        return checksum;
}
void cleanpacket(void){
        unsigned int i;
        for (i = 0;i <= sizeof packet - 1; i++) { //reset g to 0 and buffer to NULLs
                packet[i] = 0x00;
        }
}
void createXbeeTransmitPacket(char *addr_64, char *addr_16, char *payload, int length, char len_val, unsigned int
l_val1, unsigned int l_val2) {
        packet[0] = 0x7E;
        packet[1] = 0x00;
        packet[2] = len_val;
        packet[3] = 0x10;
```

```
        packet[4] = 0x01;
        mystrncat(packet, addr_64, 8, 5);
        mystrncat(packet, addr_16, 2, 13);
        packet[15] = 0x00;
        packet[16] = 0x00;
        packet[17] = 0x25;
        packet[18] = letters[l_val1];
        packet[19] = letters[l_val2];
        mystrncat(packet, payload, length, 20);
        char checksum = checkSum(length-1);
        mystrncat(packet, &checksum, 1, (20 + (length-1)));
}
void runtimer (void){
        //Set up Timer for Checking Parent
        TACTL |= TASSEL_2 + MC_1;                       //use SMCLK, and count to CCR0
        CCR0 = 11111;                                   //count up value
        CCTL0 |= CCIE;                                  //enables interrupts for timer
        timer_flag = 1;
}
void stoptimer (void){
        TACTL |= MC_0;
        CCTL0 &= ~CCIE;
        timer_flag = 0;
}

void main(void)
{
        WDTCTL = WDTPW + WDTHOLD;           // Stop WDT
        BCSCTL1 = CALBC1_1MHZ;              // Set DCO
        DCOCTL = CALDCO_1MHZ;
        BCSCTL3 |= LFXT1S_2;            //set ACLK to internal low freq clock

        //Set up I/O
        P1SEL |= BIT1 + BIT2 ;              // P1.1 = RXD, P1.2=TXD
        P1SEL2 |= BIT1 + BIT2 ;             // P1.1 = RXD, P1.2=TXD

        P2SEL &= ~BIT5;                 // set P2.5 as input to read in CTS pin (pin 12) from Xbee
        P2DIR &= ~BIT5;

        P2SEL |= BIT3;                  /sets up P2.3 as output to control sleep pin for Xbee Endpoint
        P2DIR |= BIT3;
        P2OUT |= BIT3;                  //sets P2.3 as high to start

        P1OUT |= BIT3;                  //Sets up P1.3 to be input from button
        P1REN |= BIT3;                  //uses internal resistor for button on board

        //UART Config
        UCA0CTL1 |= UCSSEL_2;               // SMCLK
        UCA0BR0 = 104;                  // 1MHz 9600
        UCA0BR1 = 0;                    // 1MHz 9600
        UCA0MCTL = UCBRS0;                  // Modulation UCBRSx = 1
        UCA0CTL1 &= ~UCSWRST;               // **Initialize USCI state machine**
        IE2 |= UCA0RXIE;                // Enable USCI_A0 RX interrupt
```

```
	P1IE |= BIT3; // P1.3 interrupt enabled
	P1IFG &= ~BIT3; // P1.3 IFG cleared

	runtimer();
	__bis_SR_register(LPM3_bits + GIE);	// Enter LPM3, interrupts enabled

}
//RX vector for UART
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
	if(E_flag == 1){
		if (UCA0RXBUF == '$' || dollarflag == 1) { //checks to see if 1st part of string is $ or if dollarflag is
set, if either are set, then the information is stored in GPSBuffer
			dollarflag = 1;
			GPSbuffer[g] = UCA0RXBUF; //store data from UART RX line to GPSbuffer
			g++;
		}
		if (GPSbuffer[3] == 'G' && GPSbuffer[4] == 'G' && GPSbuffer[5] == 'A') { //checks to see if the first
header of the buffer is $GPPGA which is the correct data for this project
			if (UCA0RXBUF == ',') comma++; //for each comma, this counter increases by one
			if (comma == 6 && UCA0RXBUF != ',') { //if the comma count is 6, then the data is ready
to be sent over UART
				UC0IE &= ~UCA0RXIE; //disables receiving
				GPSparse(); //parse relevant GPS data to pack
				createXbeeTransmitPacket(coord_64, coord_16, GPS_payload, 36, 0x34, 1, 2);
//build the transmit packet for XBee
				packetcounter = 0;
				P2OUT &= ~BIT3; //sets P2.3 as low to start data being sent from Xbee
				if (!(P2IN && BIT5)){ //checks CTS pin to determine if Xbee is ready to recieve
data
					UC0IE |= UCA0TXIE; //Enable transmitting
					UCA0TXBUF = packet[packetcounter++]; //Send GPS packet via XBee
				}
				for (;g > 0; g--) { //reset g to 0 and buffer to NULLs
					GPSbuffer[g] = 0x00;
				}
				GPSbuffer[g] = 0x00; //sets GPSBuffer[0] back to null
				dollarflag = 0;
				comma = 0;
				UC0IE |= UCA0RXIE; //re-enable receive
			}
		}
		else if (GPSbuffer[3] != 'G' && g == 4) { //if the header is not $GPPGA, then reset the buffer back to
null
			dollarflag = 0;
			GPSbuffer[0] = '\0';
			GPSbuffer[1] = '\0';
			GPSbuffer[2] = '\0';
			GPSbuffer[3] = '\0';
			g = 0;
		}
```

```
                else if (GPSbuffer[4] != 'G' && g == 5) { //if the header is not $GPPGA, then reset the buffer back to
null
                        dollarflag = 0;
                        GPSbuffer[0] = '\0';
                        GPSbuffer[1] = '\0';
                        GPSbuffer[2] = '\0';
                        GPSbuffer[3] = '\0';
                        GPSbuffer[4] = '\0';
                        g = 0;
                }
        }
        else{
                stoptimer();
                if(UCA0RXBUF == 0x7E || start == 1){
                        start = 1;
                        RX_packet[rx_counter] = UCA0RXBUF;
                        rx_counter++;
                }
                if(RX_packet[3] == 0x88 && RX_packet[5] == 0x4D && RX_packet[6] == 0x50){
                        if(rx_counter == 10){
                                UC0IE &= ~UCA0RXIE; //disables receiving
                                if((RX_packet[8] == parent_16[0]) && (RX_packet[9] == parent_16[1])){
                                        for (;rx_counter > 0; rx_counter--) { //reset g to 0 and buffer to NULLs
                                                RX_packet[rx_counter] = 0x00;
                                        }
                                        RX_packet[rx_counter] = 0x00; //sets GPSBuffer[0] back to null
                                        rx_counter = 0;
                                        start = 0;
                                        UC0IE |= UCA0RXIE; //re-enable receive
                                }
                                else {
                                        parent_16[0] = RX_packet[8];
                                        parent_16[1] = RX_packet[9];
                                        createXbeeTransmitPacket(coord_64, coord_16, parent_16, 2, 0x13, 0,
3); //ask for new parent 64 bit
                                        packetcounter =0;
                                        P2OUT &= ~BIT3; //sets P2.3 as low to start data being sent from
Xbees
                                        if (!(P2IN && BIT5)){ //checks CTS pin to determine if Xbee is ready to
recieve data
                                                UC0IE |= UCA0TXIE; //Enable transmitting
                                                UCA0TXBUF = packet[packetcounter++]; //Send GPS packet
via XBee
                                        }
                                        for (;rx_counter > 0; rx_counter--) { //reset g to 0 and buffer to NULLs
                                                RX_packet[rx_counter] = 0x00;
                                        }
                                        RX_packet[rx_counter] = 0x00; //sets GPSBuffer[0] back to null
                                        rx_counter = 0;
                                        start = 0;
                                        UC0IE |= UCA0RXIE; //re-enable receive
                                }
                        }
                }
```

```
                else if(RX_packet[3] == 0x90 && RX_packet[15] == 0x25 && RX_packet[16] == 0x53 &&
RX_packet[17] == 0x53){
                        if(rx_counter == 26){
                                UC0IE &= ~UCA0RXIE; //disables receiving
                                parent_64[0] = RX_packet[18];
                                parent_64[1] = RX_packet[19];
                                parent_64[2] = RX_packet[20];
                                parent_64[3] = RX_packet[21];
                                parent_64[4] = RX_packet[22];
                                parent_64[5] = RX_packet[23];
                                parent_64[6] = RX_packet[24];
                                parent_64[7] = RX_packet[25];
                                for (;rx_counter > 0; rx_counter--) { //reset g to 0 and buffer to NULLs
                                        RX_packet[rx_counter] = 0x00;
                                }
                                RX_packet[rx_counter] = 0x00; //sets GPSBuffer[0] back to null
                                rx_counter = 0;
                                start = 0;
                                UC0IE |= UCA0RXIE; //re-enable receive
                        }
                }
                else if((RX_packet[3] != 0x90 && rx_counter == 5) && (RX_packet[3] != 0x88 && rx_counter == 5)){
                        for (;rx_counter > 0; rx_counter--) { //reset g to 0 and buffer to NULLs
                                RX_packet[rx_counter] = 0x00;
                        }
                        RX_packet[rx_counter] = 0x00; //sets GPSBuffer[0] back to null
                        rx_counter = 0;
                        start = 0;
                }
        runtimer();
        }
}

//TX ISR
#pragma vector=USCIAB0TX_VECTOR

__interrupt void USCI0TX_ISR(void)
{
        stoptimer();
        if (!(P2IN && BIT5)){ //check CTS line to determine if Xbee is ready to recieve UART
                UCA0TXBUF = packet[packetcounter++]; //send next set of hex values over UART
                if (packetcounter == sizeof packet - 1){ // checks to see if entire message is sent
                        UC0IE &= ~UCA0TXIE; //disable tx interrupt
                        P2OUT |= BIT3; //sets P2.3 as high to put Xbee back in sleep
                        packetcounter =0;
                        cleanpacket();
                        if(!E_flag){
                                runtimer();
                        }
                }
        }

}
```

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
	E_flag = 1;
	UC0IE |= UCA0RXIE; //re-enable receive
	stoptimer();
	UC0IE &= ~UCA0RXIE; //disables receiving
	packetcounter = 0;
	createXbeeTransmitPacket(parent_64, parent_16, 0x00, 1, 0x11, 0, 0);  //create Xbee alarm packet
	if (!(P2IN && BIT5)){ //checks CTS pin to determine if Xbee is ready to recieve data
		UC0IE |= UCA0TXIE; //Enable transmitting
		UCA0TXBUF = packet[packetcounter++]; //Send alarm packet via XBee
	}
	UC0IE |= UCA0RXIE; //re-enable receive
	P1IE &= ~0x08; // P1.3 interrupt disabled
	P1IFG &= ~BIT3; // P1.3 IFG cleared //clear button isr register

}

//Timer Interrupt
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A(void){
	mystrncat(packet,check_parent,8,0);
	packetcounter = 0;
	P2OUT &= ~BIT3; //sets P2.3 as low to start data being sent from Xbee
	if (!(P2IN && BIT5)){ //checks CTS pin to determine if Xbee is ready to recieve data
		UC0IE |= UCA0TXIE; //Enable transmitting
		UCA0TXBUF = packet[packetcounter++]; //Send alarm packet via XBee
	}
	UC0IE |= UCA0RXIE; //re-enable receive
}
```

## C.6 Final Box Code

```
#include  "msp430g2553.h"
#include "mystring.h"

//Xbee API packet defines
//char check_16[8] = {0x7E, 0x00, 0x04, 0x08, 0x52, 0x4D, 0x59, 0xFF};
char box_16[2];
char coord_64 [8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
char coord_16 [2] = {0xFF, 0xFE};
char packet[25]; //Xbee API packet being created to send over UART to Xbee
unsigned int packetcounter = 0; //counter for packet being created
char RX_packet[27]; //RX Buffer for Xbee packet
unsigned int rx_counter = 0;
char letters [5] = {0x41, 0x45, 0x4D, 0x53, 0x55};
unsigned int start = 0;

unsigned int E_flag = 0;
void main(void)

{
	WDTCTL = WDTPW + WDTHOLD;		// Stop WDT
	BCSCTL1 = CALBC1_1MHZ;		// Set DCO
	DCOCTL = CALDCO_1MHZ;
	BCSCTL3 |= LFXT1S_2;		//set ACLK to internal low freq clock

	//Set up I/O
	P1SEL |= BIT1 + BIT2 ;		// P1.1 = RXD, P1.2=TXD
	P1SEL2 |= BIT1 + BIT2 ;		// P1.1 = RXD, P1.2=TXD

	P2SEL &= ~BIT0;		//sets up P2.0 as output to control power for alarm system
	P2DIR |= BIT0;		//sets P2.0 as high to start to keep alarm system off
	P2OUT |= BIT0;

	P2SEL &= ~BIT2;		///sets up P2.2 as output to control power for alarm siren
	P2DIR |= BIT2;		//sets P2.2 as high to start to keep alarm siren off
	P2OUT |= BIT2;

	//UART Config
	UCA0CTL1 |= UCSSEL_2;		// SMCLK
	UCA0BR0 = 104;		// 1MHz 9600
	UCA0BR1 = 0;		// 1MHz 9600
	UCA0MCTL = UCBRS0;		// Modulation UCBRSx = 1
	UCA0CTL1 &= ~UCSWRST;		// **Initialize USCI state machine**
	IE2 |= UCA0RXIE;		// Enable USCI_A0 RX interrupt

	__bis_SR_register(LPM3_bits + GIE);	// Enter LPM3, interrupts enabled

}
//RX vector for UART
#pragma vector=USCIAB0RX_VECTOR
```

```
__interrupt void USCI0RX_ISR(void) {
	if(UCA0RXBUF == 0x7E || start == 1){
		start = 1;
		RX_packet[rx_counter] = UCA0RXBUF;
		rx_counter++;
	}
	if(RX_packet[3] == 0x90 && RX_packet[15] == 0x25 && RX_packet[16] == 0x41 && RX_packet[17] ==
0x41){
		E_flag = 1;
		P2OUT &= ~BIT0;				//sets P2.0 as low to start alarm system
		P2OUT &= ~BIT2;				//sets P2.2 as low to start alarm siren
		for (;rx_counter > 0; rx_counter--) { //reset g to 0 and buffer to NULLs
			RX_packet[rx_counter] = 0x00;
		}
		RX_packet[rx_counter] = 0x00; //sets GPSBuffer[0] back to null
		rx_counter = 0;
		start = 0;
	}
	else if((RX_packet[3] != 0x90 && rx_counter == 5) && (RX_packet[3] != 0x88 && rx_counter == 5)){
		for (;rx_counter > 0; rx_counter--) { //reset g to 0 and buffer to NULLs
			RX_packet[rx_counter] = 0x00;
		}
		RX_packet[rx_counter] = 0x00; //sets GPSBuffer[0] back to null
		rx_counter = 0;
		start = 0;
	}
}

#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
	if (!(P2IN && BIT5)){ //check CTS line to determine if Xbee is ready to recieve UART
		UCA0TXBUF = packet[packetcounter++]; //send next set of hex values over UART
		if (packetcounter == (sizeof packet - 1)){ // checks to see if entire message is sent
			UC0IE &= ~UCA0TXIE; //disable tx interrupt
			packetcounter =0;
		}
	}
}
```

# Appendix D: Coordinator Code

## D.1 Inventory Control Program - MQPinventory.c

```
/* Inventory creation and management program
 * Part of Campus Safety System MQP
 * Author: James Beaulieu, Lauren Lewis
 * Start Date: 2/12/2014
 */

#include "MQPinventory.h"

//So the program flow should start by asking the user what inventory they want to work with (Box, Trigger, or Dispenser). Once they choose one, that .inv is fopened into FILE *inventory defined as a global. Now this can be passed around to the different functions. Then there should be the option to initialize the file (which basically wipes it clean), add a new item to the inventory, list all of the items in the inventory, delete an item in the inventory, and modify an item in the inventory. This program is designed for use by the administrators of the system. THIS IS NOT THE PROGRAM THE COORDINATOR USES TO FIND AND/OR MODIFY THE DATABASE! The program should double check every action with an extra "Are you sure?", and a password needs to run the program.

FILE *inventory;

char filename[14];

char *boxfile = "WPIboxfile.inv";
char *trgfile = "WPItrgfile.inv";
char *dspfile = "WPIdspfile.inv";

//void makelist(char *filename);
void makelist(void);
//void additem(char *filename);
void additem(void);
void listitems (void);
void deleteitem(void);
void modifyitem(void);
int password(void);

int main() {
	char input;
	printf("Database inventory program\n");
	//if (password() == -1) {
	//	return -1;
	//}
	printf("Which inventory do you need to manage?\n");
	printf("\t1-BOX\t2-TRIGGER\t3-DISPENSER\tQ-QUIT\n--> ");
	scanf("%s", &input);
```

```
	switch (input) {
		case '1':
			if ((inventory = fopen(boxfile, "r+")) == NULL) {
				printf("File could not be opened\n");
				return -1;
			}
			strcpy(filename, boxfile);
			break;
		case '2':
			if ((inventory = fopen(trgfile, "r+")) == NULL) {
				printf("File could not be opened\n");
				return -1;
			}
			strcpy(filename, trgfile);
			break;
		case '3':
			if ((inventory = fopen(dspfile, "r+")) == NULL) {
				printf("File could not be opened\n");
				return -1;
			}
			strcpy(filename, dspfile);
			break;
		case 'Q':
			return 0;
		case 'q':
			return 0;
		default:
			return 0;
	};
	while (1) {
		printf("What do you want to do in this inventory?\n");
		printf("\t1-INITIALIZE\t2-ADD ITEM\n\t3-LIST ITEMS\t4-DELETE ITEM\n\t5-MODIFY ITEM\tQ-
QUIT\n--> ");
		scanf("%s", &input);
		switch (input) {
			case '1':
				makelist();
				break;
			case '2':
				additem();
				break;
			case '3':
				listitems();
				break;
			case '4':
				deleteitem();
				break;
			case '5':
				//modifyitem();
```

```
                        break;
                case 'Q':
                        fclose(inventory);
                        return 0;
                case 'q':
                        fclose(inventory);
                        return 0;
                default:
                        fclose(inventory);
                        return 0;
        };
    }
    return 0;
}

void makelist(void) {
    int i;
    int check;
    printf("Are you sure you want to initialize? All stored data\nwill be erased from all entries in %s.
Continue?(y or n): ", filename);
    while (1) {
        check = fgetc(stdin);
        if (check == 'y') {
            printf("Initializing...\n");
            break;
        }
        else if (check == 'n') {
            printf("Initialization cancelled.\n");
            return;
        }
    }
    if (strcmp(filename, boxfile) == 0) {
        box blankitem;
        memset(&blankitem, 0, sizeof(box));
        for(i = 1; i<=100; i++){
            fseek(inventory, i * sizeof(box), SEEK_SET);
            fwrite(&blankitem, sizeof(box), 1, inventory);
        }
    } else if (strcmp(filename, trgfile) == 0) {
        trigger blankitem;
        memset(&blankitem, 0, sizeof(trigger));
        for(i = 1; i<=100; i++){
            fseek(inventory, i * sizeof(trigger), SEEK_SET);
            fwrite(&blankitem, sizeof(trigger), 1, inventory);
        }
    } else if (strcmp(filename, dspfile) == 0) {
        disp blankitem;
        memset(&blankitem, 0, sizeof(disp));
        for(i = 1; i<=100; i++){
```

```
                fseek(inventory, i * sizeof(disp), SEEK_SET);
                fwrite(&blankitem, sizeof(disp), 1, inventory);
            }
        }
}
void additem(void){
        //Add box item
        if (strcmp(filename, boxfile) == 0) {
            box newbox = {"",0,"","",1,""};
            char stream[102];
            char indexnum[6];
            int i;
            //Get BID
            fgetc(stdin);
            while (1) {
                printf("Please enter the Box ID: ");
                fgets(stream,15,stdin);
                strncpy(newbox.BID, stream, 15);
                if (stream[14] == '\0') {
                    for (i = 0; i < 6; i++) {
                        indexnum[i] = stream[i+8];
                    }
                    newbox.index = atoi(indexnum);
                    break;
                }
                else printf("Incorrect format, please re-enter\n");
            }
            //Get 64 bit
            fgetc(stdin);
            while(1) {
                printf("Please enter the 64 bit address: ");
                fgets(stream,17,stdin);
                strncpy(newbox.ADDR64, stream, 17);
                if (stream[16] != '\0') printf("Incorrect format, please re-enter\n");
                else break;
            }
            //Get 16 bit
            fgetc(stdin);
            while (1) {
                printf("Please enter the 16 bit address: ");
                fgets(stream,5,stdin);
                strncpy(newbox.ADDR16, stream, 5);
                if (stream[4] != '\0') printf("Incorrect format, please re-enter\n");
                else break;
            }
            printf("\nPlease ensure battery is charged, system assuming Acceptable charge.\n\n");
            printf("Finally, please enter up to 100 characters of metadata about the location\n\tof the box:\n-
-> ");
            fgetc(stdin);
```

```
            fgets(stream,101,stdin);
            stream[101] = '\0';
            strncpy(newbox.metadata, stream, 102);
            fseek(inventory, (newbox.index) * sizeof(box), SEEK_SET);
fwrite(&newbox, sizeof(box), 1, inventory);
    }
    //Add Trigger item
    if (strcmp(filename, trgfile) == 0) {
            trigger newtrg = {"",0,"","",""};
            char stream[18];
            char indexnum[9];
            int i;
            //Get TID
            fgetc(stdin);
            while (1) {
                    printf("Please enter the Trigger ID: ");
                    fgets(stream,18,stdin);
                    strncpy(newtrg.TID, stream, 18);
                    if (stream[17] == '\0') {
                            for (i = 0; i < 9; i++) {
                                    indexnum[i] = stream[i+8];
                            }
                            newtrg.index = atoi(indexnum);
                            break;
                    }
                    else printf("Incorrect format, please re-enter\n");
            }
            //Get 64 bit
            fgetc(stdin);
            while(1) {
                    printf("Please enter the 64 bit address: ");
                    fgets(stream,17,stdin);
                    strncpy(newtrg.ADDR64, stream, 17);
                    if (stream[16] != '\0') printf("Incorrect format, please re-enter\n");
                    else break;
            }
            //Get status
            fgetc(stdin);
            while (1) {
                    printf("Please enter the status (either DID or SID): ");
                    fgets(stream,12,stdin);
                    strncpy(newtrg.status, stream, 12);
                    if (stream[11] != '\0') printf("Incorrect format, please re-enter\n");
                    else break;
            }
            fseek(inventory, (newtrg.index) * sizeof(trigger), SEEK_SET);
fwrite(&newtrg, sizeof (trigger), 1, inventory);
    }
    //Add Dispenser item
```

```
        if (strcmp(filename, dspfile) == 0) {
                disp newdisp = {"",0,0,"","",""};
                char stream[13];
                char indexnum[3];
                int i;
                //Get DID
                fgetc(stdin);
                while (1) {
                        printf("Please enter the Dispenser ID: ");
                        fgets(stream,12,stdin);
                        strncpy(newdisp.DID, stream, 12);
                        if (stream[11] == '\0') {
                                for (i = 0; i < 3; i++) {
                                        indexnum[i] = stream[i+8];
                                }
                                newdisp.index = atoi(indexnum);
                                break;
                        }
                        else printf("Incorrect format, please re-enter\n");
                }
                //Get number of triggers
                //fgetc(stdin);
                printf("Please enter the number of triggers in the dispenser: ");
                scanf("%d", &newdisp.triggers);
                //Get MAC address
                fgetc(stdin);
                while(1) {
                        printf("Please enter the MAC address (no colons please): ");
                        fgets(stream,13,stdin);
                        strncpy(newdisp.MAC, stream, 13);
                        if (stream[12] != '\0') printf("Incorrect format, please re-enter\n");
                        else break;
                }
                //Get IP
                fgetc(stdin);
                while (1) {
                        printf("Please enter the IP address (no periods, please include 0's): ");
                        fgets(stream,13,stdin);
                        strncpy(newdisp.IP, stream, 13);
                        if (stream[12] != '\0') printf("Incorrect format, please re-enter\n");
                        else break;
                }
                fseek(inventory, (newdisp.index) * sizeof(disp), SEEK_SET);
    fwrite(&newdisp, sizeof (disp), 1, inventory);
        }
}
void listitems(void) {
        //List Box items
        if (strcmp(filename, boxfile) == 0) {
```

```
            box item;
            int i, j;
    printf("*** List of Items in the Inventory ***\n\n");
            for(i = 1; i<=100; i++){
                fseek(inventory, i * sizeof(box), SEEK_SET);
                j = fread(&item, sizeof(box), 1, inventory);
                if((j) && (item.index))
        printf("\tBID: %s\t64-bit: %s\n\t16-bit: %s\tBattery: %d(0 for Critical)\n\tMetadata: %s\n\n",item.BID,
item.ADDR64, item.ADDR16, item.battstatus, item.metadata);
            }
        }
        //List Trigger items
        else if (strcmp(filename, trgfile) == 0) {
            trigger item;
            int i, j;
            printf("*** List of Items in the Inventory ***\n\n");
            for(i = 1; i<=100; i++){
                fseek(inventory, i * sizeof(trigger), SEEK_SET);
                j = fread(&item, sizeof(trigger), 1, inventory);
                if((j) && (item.index))
        printf("\tTID: %s\t64-bit: %s\n\tUse Status: %s\n\n",item.TID, item.ADDR64, item.status);
            }
        }
        //List Dispenser items
        else if (strcmp(filename, dspfile) == 0) {
            disp item;
            int i, j;
            printf("*** List of Items in the Inventory ***\n\n");
            for(i = 1; i<=100; i++){
                fseek(inventory, i * sizeof(disp), SEEK_SET);
                j = fread(&item, sizeof(disp), 1, inventory);
                if((j) && (item.index))
        printf("\tDID: %s\tNumber of triggers stored: %d\n\tMAC address: %s\tIP address: %s\n\n",item.DID,
item.triggers, item.MAC, item.IP);
            }
        }
}
void deleteitem(void) {
        int i;
        int check;
        printf("Are you sure you want to delete an item? All stored data\nfor this item will be erased %s.
Continue?(y or n): ", filename);
        while (1) {
            check = fgetc(stdin);
            if (check == 'y') {
                printf("Initializing...\n");
                break;
            }
            else if (check == 'n') {
```

```
            printf("Initialization cancelled.\n");
            return;
        }
    }
    if (strcmp(filename, boxfile) == 0) {
        box blankitem;
        memset(&blankitem, 0, sizeof(box));
        char stream[15];
        char indexnum[6];
        //Get BID
        fgetc(stdin);
        while (1) {
            printf("Please enter the Box ID: ");
            fgets(stream,15,stdin);
            if (stream[14] == '\0') {
                for (i = 0; i < 6; i++) {
                    indexnum[i] = stream[i+8];
                }
                i = atoi(indexnum);
                break;
            }
            else printf("Incorrect format, please re-enter\n");
        }
        fseek(inventory, i * sizeof(box), SEEK_SET);
        fwrite(&blankitem, sizeof(box), 1, inventory);
    } else if (strcmp(filename, trgfile) == 0) {
        trigger blankitem;
        memset(&blankitem, 0, sizeof(trigger));
        char stream[18];
        char indexnum[9];
        //Get TID
        fgetc(stdin);
        while (1) {
            printf("Please enter the Trigger ID: ");
            fgets(stream,18,stdin);
            if (stream[17] == '\0') {
                for (i = 0; i < 9; i++) {
                    indexnum[i] = stream[i+8];
                }
                i = atoi(indexnum);
                break;
            }
            else printf("Incorrect format, please re-enter\n");
        }
        fseek(inventory, i * sizeof(trigger), SEEK_SET);
        fwrite(&blankitem, sizeof(trigger), 1, inventory);
    } else if (strcmp(filename, dspfile) == 0) {
        disp blankitem;
        memset(&blankitem, 0, sizeof(disp));
```

```
            char stream[13];
            char indexnum[3];
            //Get DID
            fgetc(stdin);
            while (1) {
                  printf("Please enter the Dispenser ID: ");
                  fgets(stream,12,stdin);
                  if (stream[11] == '\0') {
                        for (i = 0; i < 3; i++) {
                              indexnum[i] = stream[i+8];
                        }
                        i = atoi(indexnum);
                        break;
                  }
                  else printf("Incorrect format, please re-enter\n");
            }
            fseek(inventory, i * sizeof(disp), SEEK_SET);
            fwrite(&blankitem, sizeof(disp), 1, inventory);
      }
}
```

## D.2 Inventory Control Functions - coord_functions.c

```
/* Functions for coordinator program to call
 * to interact with the item inventory files
 * Author: James Beaulieu
 * Start Date: 2/20/2014
 */

#include "MQPinventory.h"
#include "coord_functions.h"

/* First function is to retrieve the information stored
 * as the 'Status' member of the trigger activated in
 * Emergency Mode. The parameter is the character string
 * which is the 64-bit address of the trigger, and the
 * function returns a character string containing the
 * 'Status' member of the trigger's struct in the database,
 * which should contain the student ID if it's been
 * activated.
*/
char * SID_from_ADDR64 (char * address) {
      //Open the trigger.inv file for traversal
      FILE *inventory;
      trigger *dummytrigger = malloc(sizeof(trigger));
      char * status = malloc(sizeof(char)*12);
      int i, j;

      if ((inventory = fopen("WPItrgfile.inv", "r+")) == NULL) {
```

```
            printf("File could not be opened\n");
            return "error";
        }

        //Use a for loop to read the trigger structs
            //into a dummy struct, check the ADDR64
            //member against the parameter address
        for (i = 1;i < 100;i++) {
            fseek(inventory, i * sizeof(trigger), SEEK_SET);
            j = fread(dummytrigger, sizeof(trigger), 1, inventory);
            if (j && strcmp(address, dummytrigger->ADDR64) == 0) {
                strncpy(status, dummytrigger->status, 12);
            }
        }

        //Return the 'Status' member for trigger
            //with address provided
        fclose(inventory);
        free(dummytrigger);
        return status;
}

/* This next function is for the trigger requesting it's
 * parent box's 64-bit address based on its 16-bit address.
 * It'll take in a character string that contains the 16-bit
 * address and returns a character string containing the
 * 64-bit address of the box.
 */
char * ADDR64_from_ADDR16 (char * shortaddress) {
        //Open the box.inv file for traversal
        FILE *inventory;
        box *dummybox = malloc(sizeof(box));
        char *longaddress = malloc(sizeof(char)*17);
        int i, j;

        if ((inventory = fopen("WPIboxfile.inv", "r+")) == NULL) {
            printf("File could not be opened\n");
            return "error";
        }

        //Use a for loop to read the box structs
            //into a dummy struct, check the ADDR16
            //member against the parameter address
        for (i = 1;i < 100;i++) {
            fseek(inventory, i * sizeof(box), SEEK_SET);
            j = fread(dummybox, sizeof(box), 1, inventory);
            if (j && strcmp(shortaddress, dummybox->ADDR16) == 0) {
                strncpy(longaddress, dummybox->ADDR64, 17);
```

```
            }
        }

        //Return the ADDR64 member for box
            //with address provided
        fclose(inventory);
        free(dummybox);
        return longaddress;
}
/* This function is for a Box to update the 16 bit
 * address stored in the inventory file. This would
 * occur if the box detects that it's 16-bit address
 * has been reassigned, which could happen if it is
 * on the fringes of the network and interferrence
 * a temporary lapse in connection.
 * The function will take in the Box's 64 bit address
 * as a char string to traverse the inventory, and
 * the char string 16-bit address to be stored into
 * inventory. It will return 0 if successful, -1 if
 * failed
 */
int box_ADDR16_update(char * address64, char * address16) {
        //Open the box.inv file for traversal
        FILE *inventory;
        box *dummybox = malloc(sizeof(box));
        int i, j;

        if ((inventory = fopen("WPIboxfile.inv", "r+")) == NULL) {
            printf("File could not be opened\n");
            return -1;
        }

        //Use a for loop to read the box structs
            //into a dummy struct, check the ADDR64
            //member against the parameter address64
        for (i = 1;i < 100;i++) {
            fseek(inventory, i * sizeof(box), SEEK_SET);
            j = fread(dummybox, sizeof(box), 1, inventory);
            if (j && strcmp(address64, dummybox->ADDR64) == 0) {
                //If entering this loop, this is the box to change
                strncpy(dummybox->ADDR16, address16, 4);
                fseek(inventory, (dummybox->index) * sizeof(box), SEEK_SET);
        fwrite(dummybox, sizeof(box), 1, inventory);
                break;
            }
        }

        //Return the ADDR64 member for box
```

```
		//with address provided
	fclose(inventory);
	free(dummybox);
	return 0;
}

/* Need two functions for trigger rental operations:
 * one for updating the inventory to reflect a trigger
 * has been checked out and one to update that a
 * trigger has been returned
 */

/* This function will be the checking out function. The
 * things that happen in this process are: Student scans
 * ID (dispenser has SID), dispenser offers trigger (has
 * TID), student leaves with trigger. Things that need to
 * happen in the inventory: trigger entry needs to change
 * the status field from the dispenser ID it was in to the
 * SID passed in, dispenser entry needs to be updated with
 * one fewer triggers stored. This will take in three
 * parameters: char * TID, char * SID, and char * DID. It
 * will return 0 if successful, -1 if error
 */
int trigger_checkout(char * TID_input, char * SID_input,
                                                        char * DID_input) {
	char *indexnum = malloc(sizeof(char)*9);
	memset(indexnum, '\0', sizeof(indexnum));
	int dispindex, trigindex, i, j;
	FILE *inventory1;
	FILE *inventory2;
	trigger *dummytrigger = malloc(sizeof(trigger));
	disp *dummydisp = malloc(sizeof(disp));

	for (i = 0; i < 3; i++) {
		indexnum[i] = DID_input[i+8];
	}
	dispindex = atoi(indexnum);
	for (i = 0; i < 9; i++) {
		indexnum[i] = TID_input[i+8];
	}
	trigindex = atoi(indexnum);

	free(indexnum);

	if ((inventory1 = fopen("WPItrgfile.inv", "r+")) == NULL) {
		printf("File could not be opened\n");
		return -1;
	}
```

```
        fseek(inventory1, (trigindex)*sizeof(trigger), SEEK_SET);
        j = fread(dummytrigger, sizeof(trigger), 1, inventory1);
        if((j) && (trigindex == dummytrigger->index)) {
                strncpy(dummytrigger->status, SID_input, 11);
                fseek(inventory1, (trigindex) * sizeof(trigger), SEEK_SET);
    fwrite(dummytrigger, sizeof (trigger), 1, inventory1);
        }

        fclose(inventory1);
        free(dummytrigger);

        if ((inventory2 = fopen("WPIdspfile.inv", "r+")) == NULL) {
                printf("File could not be opened\n");
                return -1;
        }

        fseek(inventory2, (dispindex)*sizeof(disp), SEEK_SET);
        j = fread(dummydisp, sizeof(disp), 1, inventory2);
        if((j) && (dispindex == dummydisp->index)) {
                dummydisp->triggers--;
                fseek(inventory2, (dispindex) * sizeof(disp), SEEK_SET);
    fwrite(dummydisp, sizeof (disp), 1, inventory2);
        }

        fclose(inventory2);
        free(dummydisp);

        return 0;
}

/* This is the function to check a trigger back in. It needs
 * to update the trigger's status with the DID and the
 * dispenser with a new trigger in inventory. This will take
 * two arguments, the TID and the DID, and return 0 if success,
 * -1 for failure.
 */

int trigger_checkin(char * TID_input, char * DID_input) {
        char *indexnum = malloc(sizeof(char)*9);
        memset(indexnum, '\0', sizeof(indexnum));
        int dispindex, trigindex, i, j;
        FILE *inventory1;
        FILE *inventory2;
        trigger *dummytrigger = malloc(sizeof(trigger));
        disp *dummydisp = malloc(sizeof(disp));

        for (i = 0; i < 3; i++) {
```

```
            indexnum[i] = DID_input[i+8];
        }
        dispindex = atoi(indexnum);
        for (i = 0; i < 9; i++) {
            indexnum[i] = TID_input[i+8];
        }
        trigindex = atoi(indexnum);

        free(indexnum);

        if ((inventory1 = fopen("WPItrgfile.inv", "r+")) == NULL) {
            printf("File could not be opened\n");
            return -1;
        }

        fseek(inventory1, (trigindex)*sizeof(trigger), SEEK_SET);
        j = fread(dummytrigger, sizeof(trigger), 1, inventory1);
        if((j) && (trigindex == dummytrigger->index)) {
            strncpy(dummytrigger->status, DID_input, 11);
            fseek(inventory1, (trigindex) * sizeof(trigger), SEEK_SET);
    fwrite(dummytrigger, sizeof (trigger), 1, inventory1);
        }

        fclose(inventory1);
        free(dummytrigger);

        if ((inventory2 = fopen("WPIdspfile.inv", "r+")) == NULL) {
            printf("File could not be opened\n");
            return -1;
        }

        fseek(inventory2, (dispindex)*sizeof(disp), SEEK_SET);
        j = fread(dummydisp, sizeof(disp), 1, inventory2);
        if((j) && (dispindex == dummydisp->index)) {
            dummydisp->triggers++;
            fseek(inventory2, (dispindex) * sizeof(disp), SEEK_SET);
    fwrite(dummydisp, sizeof (disp), 1, inventory2);
        }

        fclose(inventory2);
        free(dummydisp);

        return 0;
}
```

### D.3 Inventory Control Header - coord_functions.h

```
/* This is a header for the coord_functions
 * Part of the Campus Safety System MQP
```

```
 * Author: James Beaulieu
 * Start Date: 2/23/2014
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char * SID_from_ADDR64 (char * address);
char * ADDR64_from_ADDR16 (char * shortaddress);
int box_ADDR16_update(char * address64, char * address16);
int trigger_checkout(char * TID_input, char * SID_input, char * DID_input);
int trigger_checkin(char * TID_input, char * DID_input);
```

### D.4 Coordinator main program - coord_prog_0.c

```
/*
	libxbee - a C library to aid the use of Digi's XBee wireless modules
		running in API mode.

	Copyright (C) 2009 onwards  Attie Grande (attie@attie.co.uk)

	libxbee 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.

	libxbee 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 libxbee. If not, see <http://www.gnu.org/licenses/>.
*/

#include <xbee.h>
#include "coord_header.h"
int toggle;


#define MAX_SIZE 128

void * send_KML(void * data) {
	char ** args = *((char***)data);
	char * portnumber = args[0];
	char * hostname = args[1];
	int sock, numbytes;
```

```
	struct addrinfo hints, *res;
	int R;

	int port = atoi(portnumber);

	memset(&hints, 0, sizeof(hints));
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_STREAM;

	if((R = getaddrinfo(hostname, portnumber, &hints, &res)) != 0) { //get address
		fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(R));
	}

	if((sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) //open socket
				== -1) {
		perror("Client: socket");
		pthread_exit((void *)-1);
	}
	if(connect(sock, res->ai_addr, res->ai_addrlen) == -1) { //connect to address
		close(sock);
		perror("Client: connect");
		pthread_exit((void *)-1);
	}

	freeaddrinfo(res);

	 //send get request
char file[600];
	FILE * fp = fopen("testKML.kml", "r+");
	int x = 0;
	while(!(feof(fp))){
		x = fread(file+x, sizeof(char), 1000, fp);
		if(ferror(fp)) pthread_exit((void*)-1);
	}

printf("[INFO] REQUEST:\n%s\n", file);
int n = send(sock, file, strlen(file), 0); //send request
	if (n == -1) printf("Send failed!\n");

//get and print response
int numBytes, check;
	char *buf = malloc(sizeof(char) * 2);
printf("[INFO] RESPONSE: \n");
while ((numBytes = recv(sock, buf, 1, 0)) != 0) { //receive requested file
  buf[1] = '\0';
  printf("%s", buf); //print file
}
```

```
        printf("\nClosing connection...\n");
        close(sock);
        pthread_exit((void*)0);
}

void * dispthread(void *data) {
        int fdconn = *((int*)data); //cast input as contents of data as int
        printf("Opening new thread...\n");

        char *buffer = malloc(sizeof(char) * 30); //create buffer to store GET
                                                                                //request
        int received;
        printf("Waiting for request...\n");
        received = recv(fdconn, buffer, 30, 0);//call recv() to recieve
                                                                                //request, returns zero when
no new data
        buffer[29] = '\0';
        printf("%s", buffer); //print the buffer

        char *file = malloc(sizeof(char)*10); //file name from request
        char *SID = malloc(sizeof(char)*10);
        sscanf(buffer, "%s %s", file, SID); //get file name

        if(strcmp(file, "checkout") == 0) {
                trigger_checkout("T.WPI.A.000000001", SID, "D.WPI.A.001");
        }
        else if(strcmp(file, "checkin") == 0) {
                trigger_checkin("T.WPI.A.000000001", "D.WPI.A.001");
        }

        free(buffer);
        free(file);
        free(SID);

        printf("\nClosing connection...\n");
        close(fdconn); //close this socket

} //End of thread


void *serverthread(void *data) {
        char *portnum = *((char**)data);
        int port = atoi(portnum);       //convert port number to integer
        int fd; //declare port to open

        struct sockaddr_in saddr, clientaddr; //define socket address struct

        printf("Creating socket...\n");
```

```
	fd = socket(AF_INET, SOCK_STREAM, 0); //socket setup returns socket
	//descriptor integer

	printf("Binding to port %d...\n", port);
	memset(&saddr, '\0', sizeof(saddr)); //zero structure out
	saddr.sin_family = AF_INET; //match the protocol in the socket() call
	saddr.sin_addr.s_addr = htonl(INADDR_ANY); //bind to local addr
	saddr.sin_port = htons(port); //specify port for listening
	bind(fd, (struct sockaddr*) &saddr, sizeof(saddr)); //bind to port

	printf("Listening to port %d...\n", port);
	listen(fd, 1); //responds to handshake for TCP initiated by client

	printf("Trying to accept...\n");
	while(1) {
		int fdconn;
		socklen_t clientaddrSize = sizeof(clientaddr); //define

		if ((fdconn = accept(fd, (struct sockaddr*) &clientaddr, &clientaddrSize)) >= 0) { //accept called
when TCP handshake complete,
					//returns new socket descriptor for client talk
			printf("ACCEPTED!\n");
			pthread_t thread;	//declare new thread
			if (pthread_create(&thread, NULL, dispthread, &fdconn) != 0)
					//create new thread once server accepts client
				perror("Thread create failed: \n");
		}
	}
	close(fd);
	return(0);
} //End of main thread

void print_packet_info(struct xbee_pkt **pkt) {
	//printf("atCmd: %s\n", (*pkt)->atCmd);
	printf("status: %d\n",(*pkt)->status);
	char temp[17];
	int i, j, k, z = 0;
	for (i = 0;i < 8;i++){
		j = ((*pkt)->address.addr64[i])/0x10;
		k = ((*pkt)->address.addr64[i])%0x10;
		if (j < 10) {
			temp[z] = j + '0';
		} else {
			temp[z] = (j - 10) + 'A';
		}
		if (k < 10) {
			temp[z+1] = k + '0';
		} else {
```

```
                temp[z+1] = (k - 10) + 'A';
            }
            z += 2;
        }
        temp[16] = '\0';
        printf("Addr64: %s\n",temp);
        char temp16[5];
        z = 0;
        for (i = 0;i < 2;i++){
            j = ((*pkt)->address.addr16[i])/0x10;
            k = ((*pkt)->address.addr16[i])%0x10;
            if (j < 10) {
                temp16[z] = j + '0';
            } else {
                temp16[z] = (j - 10) + 'A';
            }
            if (k < 10) {
                temp16[z+1] = k + '0';
            } else {
                temp16[z+1] = (k - 10) + 'A';
            }
            z += 2;
        }
        temp16[4] = '\0';
        printf("Addr16: %s\n",temp16);
        printf("rx: [%s]\n", (*pkt)->data);
}

void conversion(char * data, int index) {
        double tochange = atof(data);//should be xxxx.xxxx
        int bignums = tochange/100;
        tochange = fmod(tochange,100.0);
        double converted = tochange/60.0;
        if(converted >= 1) {
            bignums += (int)(converted);
            converted = fmod(converted,1.0);
        }
        //now converted should be .xxxxxx
        converted *= 1000000;
        int toascii = (int)converted;
        //now converted should be xxxxxx
        int jindex, i;
        char newchar;
        if(index == 2) jindex = 8;
        else jindex = 9;
        for(i = jindex; i > index; i--) {
            newchar = (toascii%10) + 48;
            data[i] = newchar;
            toascii = toascii/10;
```

```
        }
        data[index] = 0x2E;
        for(i = (index-1); i >= 0; i--) {
                newchar = (bignums%10) + 48;
                data[i] = newchar;
                bignums = bignums/10;
        }
}

void update_kml(char * data) {
//format of payload
//%EMhhmmss.sss,ddmm.mmmm,N,dddmm.mmmm,E,1
//time, Lat, Long
//Need to convert to +-ddd.mmmmmm,+-dd.mmmmmm
char *delim, *lat, *lngtd;
char *latdir, *lngdir, *output;
delim = malloc(sizeof(char)*15);
lat = malloc(sizeof(char)*15);
lngtd = malloc(sizeof(char)*15);
latdir = malloc(sizeof(char)*2);
lngdir = malloc(sizeof(char)*2);
output = malloc(sizeof(char)*23);
char * comma = ",";

lat = memset(lat, '\0', 10);
latdir = memset(latdir, '\0', 2);
lngtd = memset(lngtd, '\0', 11);
lngdir = memset(lngdir, '\0', 2);

delim = strtok(data, comma);
lat = strtok(NULL, comma);
latdir = strtok(NULL, comma);
lngtd = strtok(NULL, comma);
lngdir = strtok(NULL, comma);
if (lat==NULL || latdir==NULL || lngtd==NULL || lngdir == NULL) {
        printf("\tYou aren't receiving coordinates yet,\n\tunsafe to write to file! Returning\n");
        return;
}

conversion(lat, 2); conversion(lngtd, 3);
if ((strcmp(latdir,"N"))==0){
        strcpy(latdir, "");
} else if ((strcmp(latdir,"S"))==0){
        strcpy(latdir, "-");
}
if ((strcmp(lngdir,"E"))==0){
        strcpy(lngdir, "");
} else if ((strcmp(lngdir,"W"))==0){
```

```
        strcpy(lngdir, "-");
}
strcpy(output, lngdir);
strcat(output, lngtd);
strcat(output, ",");
strcat(output, latdir);
strcat(output, lat);
//printf("output to file is %s\n", output);

FILE * fp = fopen("testKML.kml", "r+");
fseek(fp, 295, SEEK_SET);
fputs(output, fp);
fclose(fp);

printf("end of update function\n");
}

void parse_stuff(struct xbee_pkt **pkt) {
        char *returnstuff = malloc(sizeof(char)*17);
        int returnstatus = -1;
        if((*pkt)->data[0] == '%') {
                if((*pkt)->data[1] == 'E' && (*pkt)->data[2] == 'M') {
                        //emergency mode
                        printf("went into emergency mode\n");
                        update_kml((*pkt)->data);
                        return;
                }
                if((*pkt)->data[1] == 'A' && (*pkt)->data[2] == 'S') {
                        //Trigger requesting 64 bit of parent
                        printf("went into 64 bit request\n");
                        //call function here
                        returnstuff = ADDR64_from_ADDR16((*pkt)->address.addr16);
                        return;
                }
                if((*pkt)->data[1] == 'U' && (*pkt)->data[2] == 'S') {
                        //Box needs to update its 16 bit address
                        printf("went into 16 bit request\n");
                        //call function here
                        box_ADDR16_update((*pkt)->address.addr64, (*pkt)->address.addr16);
                        return;
                }
        } else {
                printf("Packet not formatted. Exiting...\n");
        }
}

void myCB(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data) {
        if ((*pkt)->dataLen > 0) {
```

```
            if ((*pkt)->data[0] == '@') {
                xbee_conCallbackSet(con, NULL, NULL);
                printf("*** DISABLED CALLBACK... ***\n");
            }
            printf("Connected!\n");
            print_packet_info(pkt);
            if (toggle == 0) toggle = 1;
            else toggle = 0;
            printf("Toggle is %d\n", toggle);
            parse_stuff(pkt);
        }
}

int main(int argc, char *argv[]) {
        void *d;
        struct xbee *xbee;
        struct xbee_con *con;
        struct xbee_conAddress address;
        unsigned char retVal;
        xbee_err ret;
        toggle = 0;

        char *port_number = argv[1];
        pthread_t thread;       //declare new thread
        printf("wasn't the pthread declare\n");
        if (pthread_create(&thread, NULL, serverthread, &port_number) != 0)
            //create new thread once server accepts client
            perror("Thread create failed: \n");

        if ((ret = xbee_setup(&xbee, "xbeeZB", "/dev/ttyUSB2", 9600)) != XBEE_ENONE) {
            printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
            return ret;
        }

        printf("Connected xbee should be configured!\nSetting address of box xbee\n");

        memset(&address, 0, sizeof(address));
        address.addr64_enabled = 1;
        address.addr64[0] = 0x00;
        address.addr64[1] = 0x13;
        address.addr64[2] = 0xA2;
        address.addr64[3] = 0x00;
        address.addr64[4] = 0x40;
        address.addr64[5] = 0xA7;
        address.addr64[6] = 0x44;
        address.addr64[7] = 0x19;

        printf("Establishing connection...\n");
```

```
	if ((ret = xbee_conNew(xbee, &con, "Data", &address)) != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
		printf("There was an error Lauren!\n");
		return ret;
	}

	printf("Assigning callback function...\n");
	if ((ret = xbee_conDataSet(con, xbee, NULL)) != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conDataSet() returned: %d", ret);
		printf("There was an error Lauren!\n");
		return ret;
	}

	if ((ret = xbee_conCallbackSet(con, myCB, NULL)) != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conCallbackSet() returned: %d", ret);
		printf("There was an error Lauren!\n");
		return ret;
	}

	for (;;) {
		void *p;

		if ((ret = xbee_conCallbackGet(con, (xbee_t_conCallback*)&p)) != XBEE_ENONE) {
			xbee_log(xbee, -1, "xbee_conCallbackGet() returned: %d", ret);
			return ret;
		}

		if (p == NULL) break;

		usleep(1000000);
	}

	if ((ret = xbee_conEnd(con)) != XBEE_ENONE) {
		xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
		return ret;
	}

	xbee_shutdown(xbee);

	return 0;
}
```

# Appendix E: Dispenser Code

## E.1 Dispenser main program - dispenser.py

```
#!/usr/bin/env python
#dispenser.py
```

```
#This program is for the Dispenser
#unit, it is part of the
#Campus Safety MQP
#
#Author: James Beaulieu
#Date: 4/18/14
#
#Requires LCD display,
#Numberpad, barcode scanner,
#and NFC reader

import sys
import display
import MFRC522
import time
import dispenserclient
import RPi.GPIO as GPIO

"""Program should do the following:
-initialize everything necessary
-display a message about scanning
ID to start
-start listening for input
-if ID number comes in, display options
-if checking out:
  -turn on NFC
  -read in ID number
  -send to coordinator*
  -say thanks, go back to listening
-if returning:
  -same as checking out, but send
  different info to coordinator

"""
def main():
  display.lcd_initialize()
  display.print_first_line("Starting up...")
  print "Got here"
  nfc_reader = MFRC522.MFRC522()
  print "got here too"
  UID = []
  length_list = 0
  time.sleep(2)
  while(True):
    print "and here too!"
    length_list = 0
    UID[:] = []
    display.lcd_clear()
```

```
    print "and here!"
    display.print_first_line("Please scan ID")
    display.print_second_line("to begin")
    SID = raw_input()
    if len(SID) == 9: #Assume they entered an ID
      display.lcd_clear()
      time.sleep(3)
      display.print_first_line("Welcome user!")
      time.sleep(3)
      display.lcd_clear()
      display.print_first_line("What would you")
      display.print_second_line("like to do?")
      time.sleep(3)
      display.lcd_clear()
      display.print_first_line("Press enter to")
      display.print_second_line("confirm selection")
      time.sleep(2)
      display.lcd_clear()
      display.print_first_line("Check-out = 1")
      display.print_second_line("Return = 2")
      input = raw_input()
      if input == "1":
        while length_list == 0:
          UID = MFRC522.run_nfc(nfc_reader)
          length_list = len(UID)
        #send this to the coordinator
        print "about to call client"
        dispenserclient.talktocoord('0', SID,'12360')
        print "client called"
      elif input == "2":
        while length_list == 0:
          UID = MFRC522.run_nfc(nfc_reader)
          length_list = len(UID)
        #send this to the coordinator
        dispenserclient.talktocoord('1','','12360')
      else:
        display.print_first_line("Please try again")
        time.sleep(3)

if __name__ == '__main__':
  try:
    GPIO.cleanup()
    main()
  finally:
    GPIO.cleanup()
```

## E.2 Dispenser display functions - display.py

```
#!/usr/bin/python#
```

```
# HD44780 LCD Test Script for
# Raspberry Pi
#
# Author : Matt Hawkins
# Site   : http://www.raspberrypi-spy.co.uk
#
# Date   : 26/07/2012
#

# The wiring for the LCD is as follows:
# 1 : GND
# 2 : 5V
# 3 : Contrast (0-5V)*
# 4 : RS (Register Select)
# 5 : R/W (Read Write)       - GROUND THIS PIN
# 6 : Enable or Strobe
# 7 : Data Bit 0             - NOT USED
# 8 : Data Bit 1             - NOT USED
# 9 : Data Bit 2             - NOT USED
# 10: Data Bit 3             - NOT USED
# 11: Data Bit 4
# 12: Data Bit 5
# 13: Data Bit 6
# 14: Data Bit 7
# 15: LCD Backlight +5V**
# 16: LCD Backlight GND

#import
import RPi.GPIO as GPIO
import time

# Define GPIO to LCD mapping
"""LCD_RS = 22
LCD_E  = 17
LCD_D4 = 4
LCD_D5 = 24
LCD_D6 = 23
LCD_D7 = 18"""
LCD_RS = 15
LCD_E  = 11
LCD_D4 = 7
LCD_D5 = 18
LCD_D6 = 16
LCD_D7 = 12

# Define some device constants
LCD_WIDTH = 16    # Maximum characters per line
LCD_CHR = True
```

```
LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

# Timing constants
E_PULSE = 0.00005
E_DELAY = 0.00005

def main():
  # Main program block

  """GPIO.setmode(GPIO.BCM)       # Use BCM GPIO numbers
  GPIO.setup(LCD_E, GPIO.OUT)  # E
  GPIO.setup(LCD_RS, GPIO.OUT) # RS
  GPIO.setup(LCD_D4, GPIO.OUT) # DB4
  GPIO.setup(LCD_D5, GPIO.OUT) # DB5
  GPIO.setup(LCD_D6, GPIO.OUT) # DB6
  GPIO.setup(LCD_D7, GPIO.OUT) # DB7

  # Initialise display
  lcd_init()"""

  lcd_initialize()

  # Send some test
  lcd_byte(LCD_LINE_1, LCD_CMD)
  lcd_string("Test print")
  lcd_byte(LCD_LINE_2, LCD_CMD)
  lcd_string("Test print")
  time.sleep(3)
  lcd_clear()

def print_first_line(message):
  lcd_byte(LCD_LINE_1, LCD_CMD)
  lcd_string(message)

def print_second_line(message):
  lcd_byte(LCD_LINE_2, LCD_CMD)
  lcd_string(message)

def lcd_initialize():
  #GPIO.setmode(GPIO.BCM)       # Use BCM GPIO numbers
  GPIO.setmode(GPIO.BOARD)
  GPIO.setup(LCD_E, GPIO.OUT)  # E
  GPIO.setup(LCD_RS, GPIO.OUT) # RS
  GPIO.setup(LCD_D4, GPIO.OUT) # DB4
  GPIO.setup(LCD_D5, GPIO.OUT) # DB5
```

```
  GPIO.setup(LCD_D6, GPIO.OUT) # DB6
  GPIO.setup(LCD_D7, GPIO.OUT) # DB7

  # Initialise display
  lcd_init()

def lcd_clear():
  lcd_byte(LCD_LINE_1, LCD_CMD)
  lcd_string("                ")
  lcd_byte(LCD_LINE_2, LCD_CMD)
  lcd_string("                ")

def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD)
  lcd_byte(0x32,LCD_CMD)
  lcd_byte(0x28,LCD_CMD)
  lcd_byte(0x0C,LCD_CMD)
  lcd_byte(0x06,LCD_CMD)
  lcd_byte(0x01,LCD_CMD)

def lcd_string(message):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
```

```
  if bits&0x80==0x80:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

  # Low bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)

if __name__ == '__main__':
  try:
    main()
  finally:
    GPIO.cleanup()
```

### E.3 Dispenser NFC functions - MFRC522.py

```
import RPi.GPIO as GPIO
import spi
import signal

class MFRC522:
  NRSTPD = 22

  MAX_LEN = 16

  PCD_IDLE       = 0x00
```

```
PCD_AUTHENT    = 0x0E
PCD_RECEIVE    = 0x08
PCD_TRANSMIT   = 0x04
PCD_TRANSCEIVE = 0x0C
PCD_RESETPHASE = 0x0F
PCD_CALCCRC    = 0x03

PICC_REQIDL    = 0x26
PICC_REQALL    = 0x52
PICC_ANTICOLL  = 0x93
PICC_SElECTTAG = 0x93
PICC_AUTHENT1A = 0x60
PICC_AUTHENT1B = 0x61
PICC_READ      = 0x30
PICC_WRITE     = 0xA0
PICC_DECREMENT = 0xC0
PICC_INCREMENT = 0xC1
PICC_RESTORE   = 0xC2
PICC_TRANSFER  = 0xB0
PICC_HALT      = 0x50

MI_OK       = 0
MI_NOTAGERR = 1
MI_ERR      = 2

Reserved00     = 0x00
CommandReg     = 0x01
CommIEnReg     = 0x02
DivlEnReg      = 0x03
CommIrqReg     = 0x04
DivIrqReg      = 0x05
ErrorReg       = 0x06
Status1Reg     = 0x07
Status2Reg     = 0x08
FIFODataReg    = 0x09
FIFOLevelReg   = 0x0A
WaterLevelReg  = 0x0B
ControlReg     = 0x0C
BitFramingReg  = 0x0D
CollReg        = 0x0E
Reserved01     = 0x0F

Reserved10     = 0x10
ModeReg        = 0x11
TxModeReg      = 0x12
RxModeReg      = 0x13
TxControlReg   = 0x14
TxAutoReg      = 0x15
```

```
TxSelReg       = 0x16
RxSelReg       = 0x17
RxThresholdReg = 0x18
DemodReg       = 0x19
Reserved11     = 0x1A
Reserved12     = 0x1B
MifareReg      = 0x1C
Reserved13     = 0x1D
Reserved14     = 0x1E
SerialSpeedReg = 0x1F

Reserved20        = 0x20
CRCResultRegM     = 0x21
CRCResultRegL     = 0x22
Reserved21        = 0x23
ModWidthReg       = 0x24
Reserved22        = 0x25
RFCfgReg          = 0x26
GsNReg            = 0x27
CWGsPReg          = 0x28
ModGsPReg         = 0x29
TModeReg          = 0x2A
TPrescalerReg     = 0x2B
TReloadRegH       = 0x2C
TReloadRegL       = 0x2D
TCounterValueRegH = 0x2E
TCounterValueRegL = 0x2F

Reserved30      = 0x30
TestSel1Reg     = 0x31
TestSel2Reg     = 0x32
TestPinEnReg    = 0x33
TestPinValueReg = 0x34
TestBusReg      = 0x35
AutoTestReg     = 0x36
VersionReg      = 0x37
AnalogTestReg   = 0x38
TestDAC1Reg     = 0x39
TestDAC2Reg     = 0x3A
TestADCReg      = 0x3B
Reserved31      = 0x3C
Reserved32      = 0x3D
Reserved33      = 0x3E
Reserved34      = 0x3F

serNum = []

def __init__(self,spd=1000000):
```

```
  spi.openSPI(speed=1000000)
  GPIO.setmode(GPIO.BOARD)
  GPIO.setup(22, GPIO.OUT)
  GPIO.output(self.NRSTPD, 1)
  self.MFRC522_Init()

def MFRC522_Reset(self):
  self.Write_MFRC522(self.CommandReg, self.PCD_RESETPHASE)

def Write_MFRC522(self,addr,val):
  spi.transfer(((addr<<1)&0x7E,val))

def Read_MFRC522(self,addr):
  val = spi.transfer((((addr<<1)&0x7E) | 0x80,0))
  return val[1]

def SetBitMask(self, reg, mask):
  tmp = self.Read_MFRC522(reg)
  self.Write_MFRC522(reg, tmp | mask)

def ClearBitMask(self, reg, mask):
  tmp = self.Read_MFRC522(reg);
  self.Write_MFRC522(reg, tmp & (~mask))

def AntennaOn(self):
  temp = self.Read_MFRC522(self.TxControlReg)
  if(~(temp & 0x03)):
    self.SetBitMask(self.TxControlReg, 0x03)

def AntennaOff(self):
  self.ClearBitMask(self.TxControlReg, 0x03)

def MFRC522_ToCard(self,command,sendData):
  backData = []
  backLen = 0
  status = self.MI_ERR
  irqEn = 0x00
  waitIRq = 0x00
  lastBits = None
  n = 0
  i = 0

  if command == self.PCD_AUTHENT:
    irqEn = 0x12
    waitIRq = 0x10
  if command == self.PCD_TRANSCEIVE:
    irqEn = 0x77
    waitIRq = 0x30
```

```
self.Write_MFRC522(self.CommIEnReg, irqEn|0x80)
self.ClearBitMask(self.CommIrqReg, 0x80)
self.SetBitMask(self.FIFOLevelReg, 0x80)

self.Write_MFRC522(self.CommandReg, self.PCD_IDLE);

while(i<len(sendData)):
  self.Write_MFRC522(self.FIFODataReg, sendData[i])
  i = i+1

self.Write_MFRC522(self.CommandReg, command)

if command == self.PCD_TRANSCEIVE:
  self.SetBitMask(self.BitFramingReg, 0x80)

i = 2000
while True:
  n = self.Read_MFRC522(self.CommIrqReg)
  i = i - 1
  if ~((i!=0) and ~(n&0x01) and ~(n&waitIRq)):
    break

self.ClearBitMask(self.BitFramingReg, 0x80)

if i != 0:
  if (self.Read_MFRC522(self.ErrorReg) & 0x1B)==0x00:
    status = self.MI_OK

    if n & irqEn & 0x01:
      status = self.MI_NOTAGERR

    if command == self.PCD_TRANSCEIVE:
      n = self.Read_MFRC522(self.FIFOLevelReg)
      lastBits = self.Read_MFRC522(self.ControlReg) & 0x07
      if lastBits != 0:
        backLen = (n-1)*8 + lastBits
      else:
        backLen = n*8

      if n == 0:
        n = 1
      if n > self.MAX_LEN:
        n = self.MAX_LEN

      i = 0
      while i<n:
```

```
            backData.append(self.Read_MFRC522(self.FIFODataReg))
            i = i + 1;
        else:
          status = self.MI_ERR

      return (status,backData,backLen)


    def MFRC522_Request(self, reqMode):
      status = None
      backBits = None
      TagType = []

      self.Write_MFRC522(self.BitFramingReg, 0x07)

      TagType.append(reqMode);
      (status,backData,backBits) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE, TagType)

      if ((status != self.MI_OK) | (backBits != 0x10)):
        status = self.MI_ERR

      return (status,backBits)


    def MFRC522_Anticoll(self):
      backData = []
      serNumCheck = 0

      serNum = []

      self.Write_MFRC522(self.BitFramingReg, 0x00)

      serNum.append(self.PICC_ANTICOLL)
      serNum.append(0x20)

      (status,backData,backBits) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE,serNum)

      if(status == self.MI_OK):
        i = 0
        if len(backData)==5:
          while i<4:
            serNumCheck = serNumCheck ^ backData[i]
            i = i + 1
          if serNumCheck != backData[i]:
            status = self.MI_ERR
        else:
          status = self.MI_ERR
```

```
  return (status,backData)

def CalulateCRC(self, pIndata):
  self.ClearBitMask(self.DivIrqReg, 0x04)
  self.SetBitMask(self.FIFOLevelReg, 0x80);
  i = 0
  while i<len(pIndata):
    self.Write_MFRC522(self.FIFODataReg, pIndata[i])
    i = i + 1
  self.Write_MFRC522(self.CommandReg, self.PCD_CALCCRC)
  i = 0xFF
  while True:
    n = self.Read_MFRC522(self.DivIrqReg)
    i = i - 1
    if not ((i != 0) and not (n&0x04)):
      break
  pOutData = []
  pOutData.append(self.Read_MFRC522(self.CRCResultRegL))
  pOutData.append(self.Read_MFRC522(self.CRCResultRegM))
  return pOutData

def MFRC522_SelectTag(self, serNum):
  backData = []
  buf = []
  buf.append(self.PICC_SElECTTAG)
  buf.append(0x70)
  i = 0
  while i<5:
    buf.append(serNum[i])
    i = i + 1
  pOut = self.CalulateCRC(buf)
  buf.append(pOut[0])
  buf.append(pOut[1])
  (status, backData, backLen) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE, buf)

  if (status == self.MI_OK) and (backLen == 0x18):
    print "Size: " + str(backData[0])
    return  backData[0]
  else:
    return 0

def MFRC522_Auth(self, authMode, BlockAddr, Sectorkey, serNum):
  buff = []
  buff.append(authMode)
  buff.append(BlockAddr)
  i = 0
  while(i < len(Sectorkey)):
```

```
      buff.append(Sectorkey[i])
      i = i + 1
    i = 0
    while(i < len(serNum)):
      buff.append(serNum[i])
      i = i +1
    (status, backData, backLen) = self.MFRC522_ToCard(self.PCD_AUTHENT,buff)
    if not(status == self.MI_OK):
      print "AUTH ERROR!!"
    if not (self.Read_MFRC522(self.Status2Reg) & 0x08) != 0:
      print "AUTH ERROR(status2reg & 0x08) != 0"

    return status

  def MFRC522_Read(self, blockAddr):
    recvData = []
    recvData.append(self.PICC_READ)
    recvData.append(blockAddr)
    pOut = self.CalulateCRC(recvData)
    recvData.append(pOut[0])
    recvData.append(pOut[1])
    (status, backData, backLen) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE, recvData)
    if not(status == self.MI_OK):
      print "Error while reading!"

    print "Got data size: "+str(backLen)
    i = 0
    if len(backData) == 16:
      print "Sector "+str(blockAddr)+" "+str(backData)

  def MFRC522_Write(self, blockAddr, writeData):
    buff = []
    buff.append(self.PICC_WRITE)
    buff.append(blockAddr)
    crc = self.CalulateCRC(buff)
    buff.append(crc[0])
    buff.append(crc[1])
    (status, backData, backLen) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE, buff)
    if not(status == self.MI_OK) or not(backLen == 4) or not((backData[0] & 0x0F) == 0x0A):
      status = self.MI_ERR

    print str(backLen)+" backdata &0x0F == 0x0A "+str(backData[0]&0x0F)
    if status == self.MI_OK:
        i = 0
        buf = []
        while i < 16:
          buf.append(writeData[i])
          i = i + 1
```

```
        crc = self.CalulateCRC(buf)
        buf.append(crc[0])
        buf.append(crc[1])
        (status, backData, backLen) = self.MFRC522_ToCard(self.PCD_TRANSCEIVE,buf)
        if not(status == self.MI_OK) or not(backLen == 4) or not((backData[0] & 0x0F) == 0x0A):
          print "Error while writing"
        if status == self.MI_OK:
          print "Data writen"


    def MFRC522_Init(self):
      GPIO.output(self.NRSTPD, 1)

      self.MFRC522_Reset();



      self.Write_MFRC522(self.TModeReg, 0x8D)
      self.Write_MFRC522(self.TPrescalerReg, 0x3E)
      self.Write_MFRC522(self.TReloadRegL, 30)
      self.Write_MFRC522(self.TReloadRegH, 0)

      self.Write_MFRC522(self.TxAutoReg, 0x40)
      self.Write_MFRC522(self.ModeReg, 0x3D)
      self.AntennaOn()

#continue_reading = True
# Capture SIGINT
def end_read(signal,frame):
  global continue_reading
  print "Ctrl+C captured, ending read."
  continue_reading = False
  GPIO.cleanup() # Suggested by Marjan Trutschl

continue_reading = True
signal.signal(signal.SIGINT, end_read)
#MIFAREReader = MFRC522()

def run_nfc(reader):
  backData = []
  #signal.signal(signal.SIGINT, end_read)
  (status,TagType) = reader.MFRC522_Request(reader.PICC_REQIDL)

  if status == reader.MI_OK:
    print "Card detected"

  (status,backData) = reader.MFRC522_Anticoll()
  if status == reader.MI_OK:
```

```
    print "Card read UID:
"+str(backData[0])+","+str(backData[1])+","+str(backData[2])+","+str(backData[3])+","+str(backData[4])

    key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

    reader.MFRC522_SelectTag(backData)

    #status = reader.MFRC522_Auth(reader.PICC_AUTHENT1A, 11, key, backData)
    if status == reader.MI_OK:
      print "AUTH OK"
    else:
      print "AUTH ERROR"
  return backData
```

## E.4 Dispenser client extension module - dispenserclient.c

```
#include <Python.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define MAX 128

static PyObject *
talktocoord(PyObject *self, PyObject *args) {
        int sock;
        int zero = 0, one = 1;
        //char buf[MAX];
        struct addrinfo hints, *res;
        int R;
        char *input = malloc(sizeof(char));
        char *SID = malloc(sizeof(char)*9);
        char *portchar = malloc(sizeof(char)*5);
        PyArg_ParseTuple(args, "sss", &input, &SID, &portchar);
        int type = atoi(input);
        printf("input is %s, SID is %s, port is %s\n",input, SID, portchar);

        printf("Inside client function!\n");

        /*if (argc != 3) { //check for port number from user input
                printf("Need address and port number\n"); //print error
                return 1;
```

```
	}*/

	/*char *file;
	char *url = strtok_r(argv[1], "/", &file);
	char * portchar = argv[2];
	int port = atoi(argv[2]);*/

	char * hostname = "130.215.141.230";
	//char * portchar = "12346";
	//int port = atoi(portchar);

	memset(&hints, 0, sizeof(hints));
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_STREAM;

	if((R = getaddrinfo(hostname, portchar, &hints, &res)) != 0) { //get address
		fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(R));
	}

	if((int)(sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) //open socket
			== -1) {
		perror("Client: socket");
		//return Py_BuildValue("i", one);
	}
	if((int)(connect(sock, res->ai_addr, res->ai_addrlen)) == -1) { //connect to address
		close(sock);
		perror("Client: connect");
		//return Py_BuildValue("i", one);
	}

	freeaddrinfo(res);

	printf("opened socket fine!\n");

	/* //send get request
char request[128] = "GET /"; //form request
strcat(request, file);
strcat(request, " HTTP/1.1\r\nHost: ");
strcat(request, url);
strcat(request, "\r\nConnection: close\n\r\n");*/

	char request[30]; //form request
	memset(request, '\0', 30);
	printf("request is %s\n",request);
	if(type == 0){ //checkout
		strcat(request, "checkout ID");
		strcat(request, SID);
	}
```

```
        else if(type == 1){ //check in
                strcat(request, "checkin ID");
                strcat(request, "blank");
        }

  printf("[INFO] REQUEST:\n%s\n", request);
  int n = send(sock, request, strlen(request), 0); //send request
        if (n == -1) printf("Send failed!\n");

  /*//get and print response
  int numBytes, check;
        char *buf = malloc(sizeof(char) * 2);
  printf("[INFO] RESPONSE: \n");
  while ((numBytes = recv(sock, buf, 1, 0)) != 0) { //receive requested file
    // printf("\nNUMBER OF BYTES: %d\n", numBytes);
    buf[1] = '\0';
    printf("%s", buf); //print file
  }*/

        printf("\nClosing connection...\n");
        close(sock);
        return Py_BuildValue("i", zero);
}
static PyMethodDef
module_functions[] = {
        {"talktocoord", talktocoord, METH_VARARGS, "Transfer data"},
        {NULL}
};

void
initdispenserclient(void)
{
        Py_InitModule3("dispenserclient", module_functions, "Socket transfer module.");
}
```

### E.5 Dispenser client extension module setup script - setup.py

```
from distutils.core import setup, Extension

module1 = Extension("dispenserclient", sources=["dispenserclient.c"])

setup (
    name = 'Dispenser Client',
    author='James Beaulieu',
    version = '1.0',
    ext_modules = [module1]
)
```

### E.6 Dispenser script to run program on startup - rundispenser.sh

```
while [ 1=1 ];
do
        if [ ! "$(pgrep dispenser.py)" ];
        then
                cd /home/pi/ && sudo /home/pi/dispenser.py
        fi
        sleep 10
done
```