The Personal Remote Display (PRD)

E90 Senior Project

Allison J. Barlow

Advisor Erik Cheever

May 8, 2008

Contents

Abstract

There are many relationships which require one individual to know the real-time status and availability of another individual relative to a specific location. This project seeks to provide a means of attaining that information. The Personal Remopte Display (PRD) is a device that will display the relevant information about its owner and be able to be updated remotely from the owner's personal computer. For example, the device could be placed on or next to the owner's office door and be updated wirelessly with the message that its owner is in a meeting and will be done by noon. This paper discusses the implementation of a robust prototype of this project and the future work that can be done.

1 Introduction

1.1 The Problem of Dynamic Scheduling Information

In the ideal world, everyone would keep to their schedules. Nobody would ever be late or change their plans, and individuals would always be available when they said they would be. Naturally, schedules would be shared online with the relevant people: students, teachers, coworkers, friends, and family. This calendar-sharing software is now available to everyone, whether it be through Google Calendar, Apple's iCal, or Microsoft's Outlook software. There are still some inter-software syncing issues, but the software is otherwise very robust.

In a world in which there is so much information available online, it can be surprisingly hard to find information as simple as the schedule or current location of an individual, despite the availability of calendaring software and ease of calendar sharing. A student may visit the office of a professor only to find that he or she needs to use a personal computer in order to look up a schedule. An employee may visit the office of his boss to find the door closed and wonder as to when he should return to ask for that well-earned raise. Both the professor and the boss are likely to have calendaring software or a PDA, yet the frequently changed information that the software contains is not always available to others, nor is it easy to udate as fast as the changes occur, even if it is shared.

The technology is available, and it is easy to use, but not *convenient* to use as frequently as it needs to be. Changes in schedules occur far to frequently to make entering every change into a calendar reasonable. If an employee needs to talk to a coworker about a senstive issue, it would be convient to be able to post a note saying that one is busy without entering it into one's calendar. Likewise, if a professor needs to help a student with a problem in the nearby computer lab, it would be nice to leave a note for other students that may come to find him or her. Cell-phones make it very easy to let specific people know when you are running behind or have had a change of plans, but there is no way to leave a note for a generic set of people, other than writing a note out by hand and sticking it to your door.

1.2 A Proposed Solution

A Personal Remote Display (PRD) is proposed to help solve this problem. The PRD is a two part device that on one end connects to the user's personal computer and on the other end connects to an LCD screen that can display the owner's status, availabilty and Calendar information. The two ends of the system will be connected wirelessly to enable easy installation and removal.

There are two target categories of users for this device: the owner, who installs the system in his or her office or work space, and the coworkers and collegues that query the device for the most current information about the owner. For clarity, the first category of users is termed *owners* and the second category of users is given the generic term of *users*.

The system is is intended to have the LCD compontent be mounted in a public space, right outside an office or dorm door, for example, and be updated with information about its owner from their PC within the office or dorm room. When the owner would need to update his or her status, they would merely type in the update through a web-based interface. The owner's calendar could also be linked to the system and if no status message is displayed, it could be set to display the current calendar event.

The users would aproach the mounted LCD component and touch the screen to wake it up from a sleeping or power-saving mode. The screen would light up and display the owner's current status. If the current status has not been updated recently, the users could toggle an option to view a calendar, in order to determine when the owner migth be in next, etc. After a set time, the screen would return to its sleeping mode and default back to displaying the owner's status.

1.3 Concerns

The best solution to a problem is not always in electrical gadgetry. The startup cost of the component parts and the ongoing costs of power consuption may outweigh the value of the use of a device, for either economic or environmental reasons. The development of the PRD was done with these concerns in mind: component parts were selected based on price and power consumption and software development sought to increase the value of the product by adding useful features. While the value of the product versus the cost of production and continual use is entirely personal prefence, it is the hope that the PRD is the most convinient solution to the problem presented, making the benifits worth the costs.

2 Overview of the PRD

The implementation of the PRD can be broken down into two main components: hardware and software. The hardware aspect if this project consists of purchasing the component parts and ensuring the connections between the parts function properly. The component parts of the PRD are the intelligent LCD and touchscreen, the PIC microcontroller, and the two wireless radio modules.

Figure 1, below, shows a block diagram of these component parts. The LCD screen (upper left) is connected via a standard serial connection to the PIC microcontroller prototyping board (left center). The XBee wireless modules (bottom) form a wirless connection that, once set up properly, acts as a standard serial connection. The PIC board controls the LCD screen based on input messages recieved from both the touchscreen on the LCD and the owner's PC (via the wireless Xbee modules). The touchscreen sends messages on the same serial connection as does the LCD. The connection between the PIC microcontroller and the XBee wireless module on the LCD side of the device is a modified serial connection, which will be discussed in more detail later.

The software aspect of the development consists of the implementation of the communication protocol between the PIC microcontroller and the intelligent LCD screen, the communciation protocol between the owner's personal computer and the PIC microcontroller, the PC owner interface and the LCD user interface.

3 Implementation Details

As mentioned previously, the PRD can be broken into two components for implementation: hardware and software.

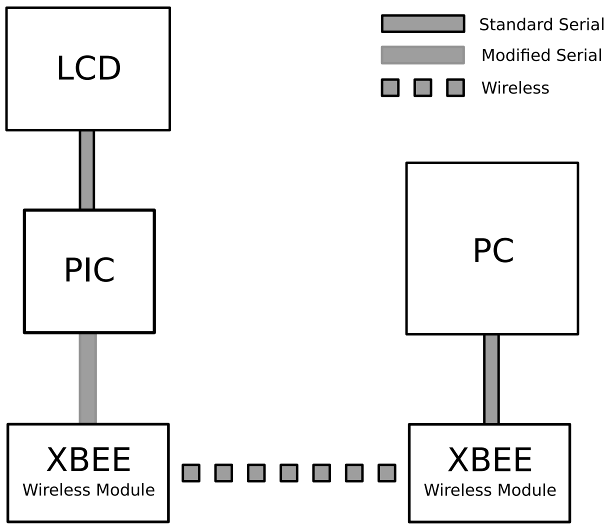


Figure 1: Block Diagram of the PRD

3.1 Hardware

The hardware of the PRD can be broken down into smaller component parts: an LCD/touchscreen, a PIC microcontroller, and two wireless modules. Specifically, the LCD and touchscreen package chosen was the ezLCD-002 Development Kit. The microcontroller chosen was the PIC18FJ10 on a development board, and the the Zigbee XBee chips were chosen for wireless communication.

Figure 2, below, is the hardware to be connected to an owner's PC. This hardware consists of a XBee module and its serial development board and a serial cable with which to connect to the PC. Figure 3 is the hardware to build the independent LCD-end of the PRD, without the power supply. It consists of the ezLCD-002 LCD and touch screen, the PIC18F67J10 development board, a Xbee chip, and the serial cable with which to connect the LCD screen and the PIC board. Again, this figure does not include the cables required to power the independent LCD-component.

3.1.1 LCD Screen

Since the main goal of this project to produce a product and not to develop specific hardware, the LCD screen was purchased in an off-the shelf package which included both an intelligent

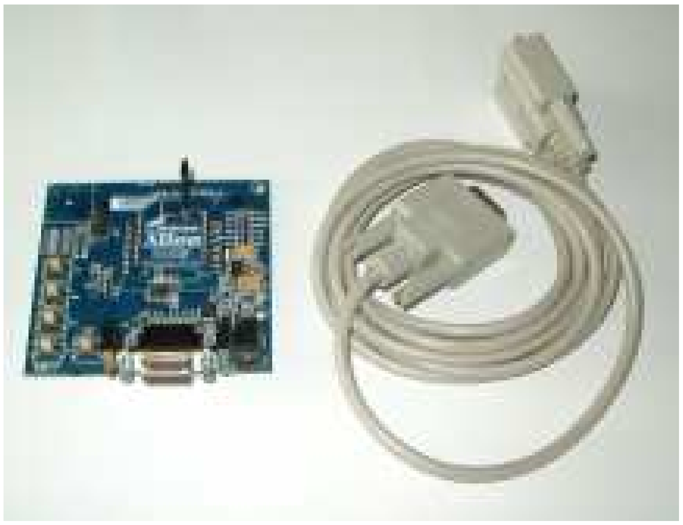


Figure 2: Hardware on the PC end

programmable LCD module and a touchscreen. It came with a software development kit, making using the display for both input and output fairly straightforward. The package chosen was the ezLCD-002 Development Kit, which provided a 2.7 inch display. The resolution on the screen is 160 by 240 pixels and can be seen in Figure 4 below. The touchscreen provided was broken down into an eight by six grid, outputting the coordinates of the pen location and a pen up message when the pen (or finger) is removed from the screen.

The kit provided several interfaces to choose from in controlling the display and recieving input from the touchscreen, including serial, USB, and parallel interfaces. The simplest to work with was the RS232 serial port, which could be directly connected to the PIC development board.

3.1.2 Zigbee XBee Wireless

In order for the display device to be updated, the owner's PC must be able to send wireless commands to the device. Zigbee RF (Radio Frequency) wireless provides a simple, serial interface in which to do this. The Zigbee protocol is similar to the more popular Bluetooth communication; both use radio to communicate wirelessly on a short range. The Zigbee XBee modules (Figure 5) are low-power and operate at a shorter range, which at an indoor range of 30 meters is more than adequate.

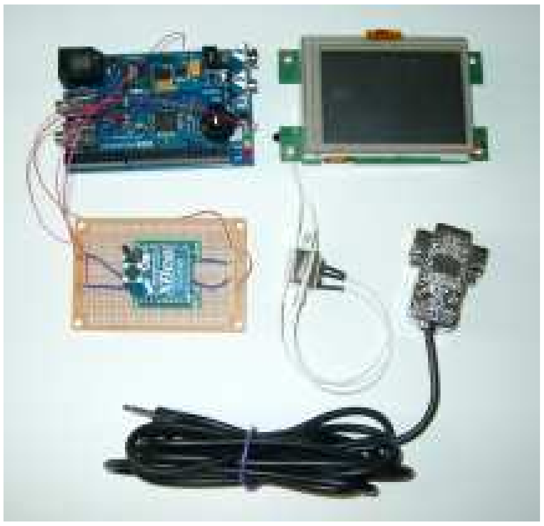


Figure 3: Hardware on the LCD end, without power supply

The Zigbee XBee module, like the LCD, also uses a serial interface to connect with the PIC. This module communicates with another XBee, which talks to the owner's PC. Since there are multiple projects which use the same Zigbee models, the two particular XBee chips that are being used must be set to communicate only with each other. Each chip has a SH (serial high) addess and SL (serial low) address, each 32 bits for a total of 64 bits. These addresses are factory set and read only. At over 10^{19} possible addresses, there is more than enough for each module produced to have a totally unique serial address. Each chip also has a DH (destination high) address and a DL (destination low) address, also 32 bits each. Depending on how these are set, a module can broadcast to a family of other modules or to a single module. In this case, a single point-to-point connection was desired, so the two modules were polled for their SH and SL addresses and their DH and DL addresses were set to their partner's serial addresses. After these changes were made and the modules connected, the communication between the PIC and the PC runs just as though there were a serial cable between the two.

3.1.3 PIC

The PIC18F67J10 development board, as seen if Figure 6, was selected for its two easily accessible serial ports. The microchip is programmed using a standard PICC compiler and ICD USB-interface.

Figure 4: ezLCD-002 Intelligent Programmable LCD with Touchscreen

The microprocessor operates on a 2.0V to 3.6V range and the whole board operates on a 9V range as some components require higher voltages. It was observed that in using the serial communications the PIC must ramp up signals sent to the Zigbee XBee to the standard serial voltage range (10V), even though the XBee operates on a 2.8V to 3.4V scale. In order to conserve power, however, the chip which performs this signal amplification had its power supply cut and the XBee module was connected directly to the development board, maintaining a constant voltage of about 3.3V from the PIC to the XBee.

3.2 Software

The software of the PRD can be broken into the communication between the PIC and the LCD/touchscreen, communication between the owner's PC and the PIC, the user interface for the LCD, and the owner interface on the PC.

3.2.1 PIC to LCD Communication

The software on the PIC must send commands to the LCD in order for the user interfaces to be displayed. The LCD screen is intelligent and has built in drawing and text functions, making this fairly easy. For example, to draw a green circle in the center of the screen, one first sends a byte with the set-x and y-coordinates command, followed by a byte for

Figure 5: A Zigbee XBee Wireless Module

the x coord and a byte for the y coord, then a set color command byte followed by a byte representing the color, and finally a draw circle command followed by the desired radius. This same operation is outlined below as well.

```
// decimal values to send via serial port
// objective: draw a green circle with radius of 60 px and center at (120,80)

36   // set color
56   // green

37   // set cursor position
120  // x coordinate
80   // y coordinate

41   // draw circle
60   // radius
```

Repeating these commands can be very tedious if one wants to draw multiple circles, so these simple functions were abstracted into higher level functions in PICC which allow a programmer to draw complicated interfaces on the screen with minimal effort. The PICC code for the above functionality would look as follows. Note that all but a single line implement the higher functionallity, so drawing multiple circles would only require duplicating the *draw_circle* command.

```
#define GREEN 56
```

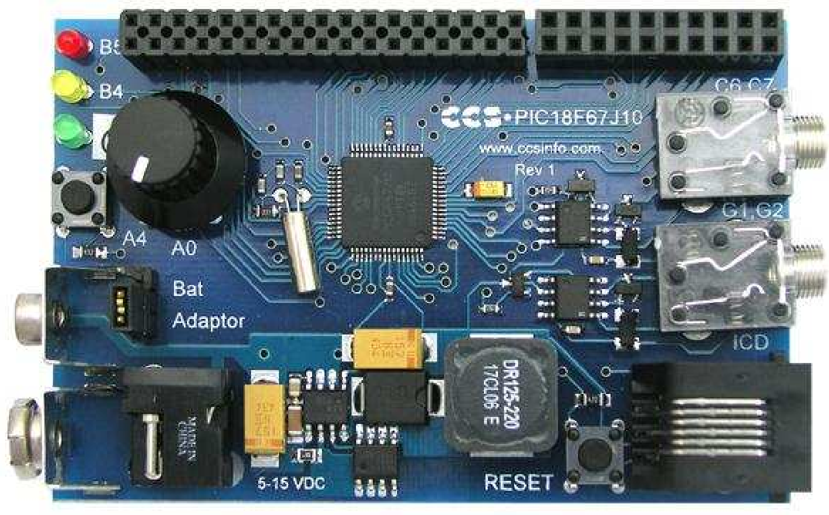


Figure 6: The PIC18F67J10 Development Board

```
void set_color(int color);
void set_xy(int x, int y);
void draw_circle(int radius);
void draw_circle(int radius, int x, int y, int color);


//draw_circle(radius, x-coord, y-coord, color);
draw_circle(60, 120, 80, GREEN);

void set_color(int color) {
   printf("\$");
   printf("\%c",color);
}

void set_xy(int x, int y) {
   printf("%c",37);
   printf("%c",x);
   printf("%c",y);
}

void draw_circle(int radius) {
   printf(")");
   printf("%c",radius);
}

void draw_circle(int radius, int x, int y, int color) {
   set_color(color);
   set_xy(x,y);
   draw_circle(radius);
}
```

Other functions are implemented in a similar way. The built in function range from drawing arcs, lines, and boxes to adding text. These functions were abstracted into PICC as needed in order to draw the user interface.

The touch screen interaction is also fairly simple; the screen is broken down into a 6 by 8 grid, and when it is touched, the coordinate of contact is sent through the same LCD-PIC serial cable, which is waiting for the coordinate with a looped *getc()* command. From there, the PIC updates the LCD appropriately. A "pen up" character is sent when the pen is removed from the screen. The functionality required by the design of the LCD user interface only required knowing when the screen was touched, not where it was touched. To this end, the software only looks for the "pen up" character. If the software accepted any touchscreen output character (coordinates of pen location) then a strobe effect would occur, even if delays were added.

3.2.2 PC to PIC Communication

ActiveComport is a proprietary serial communication tool used for accessing the serial port in the Windows operating system. At its heart is a DLL (Dynamic-Link Library) file, which is a shared library. DLL files have the same formating as the commonly known EXE files. ActiveComport was used to send serial communications from the PC. The package comes with examples in HTML/Javascript, which was desirable for expansion to remote access of the project. ActiveComport is accessed through a Javascript object and sends a serial message with a simple write command.

The characters sent are recieved on the PIC as character bytes. In order to interpret the sent characters correctly, the messages were tagged as shown below. In the following example, a status message is sent: first the *m* character as the tag for a status message update, then the status message itself, followed finally by a timestamp. This message from the PC to the PIC would update the status message and the timestamp of when the last status message was recieved.

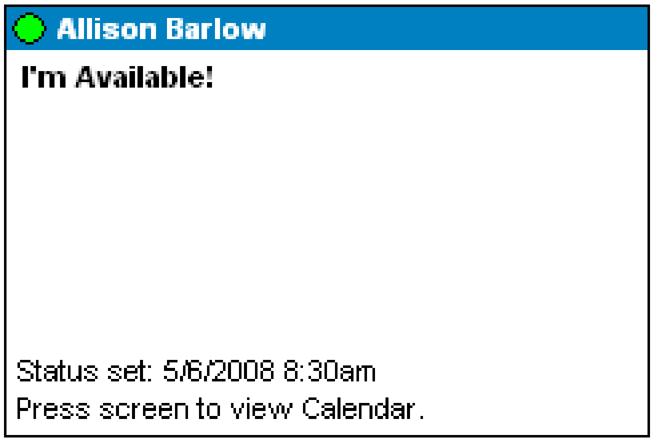
```
<m><I'm Available!><5/8/2008 12:00 PM>
```

For caledar item updates, a *c* character is sent followed by the number of the calendar item to update and then the text of the calendar item. The name of an owner can be set with the *o* character followed by the name of the owner. The < and > characters are sent in the messages as well as separators. This system can be expanded for greater functionality be adding new tags to the protocol system.

3.2.3 LCD User Interface

The design and implementation of the LCD user interface for this device is an essential component of the product, as it is key to the usability of the device. The PCI microchip is programmed with several "screens," or predetermined layouts for the LCD to display. These screens are written with the high level PICC display functions as described previously. The PIC waits for update commands from the PC and then stores them as strings which then get inserted into the expected layout. As a prototype, there are two LCD screens that are merely toggled between. The toggle is triggered upon the touch screen sending the pen lifted character.

The two screens implemented int eh prototype are the home screen, or default starting screen, and the calendar screen. An example of the home screen can be seen in Figure 7. The calendar screen is organized similarly to this screen with the word *Calendar* replacing the owner's name at the top and a list of text calendar items underneath, where the "I'm available!" message is displayed. Additionally, there is a similar use message about toggling ebtween the screens and no timestamp is displayed as calendar information is not as time sensitive.


Allison Barlow

I'm Available!

Status set: 5/6/2008 8:30am

Press screen to view Calendar.

Figure 7: An Example of the LCD User Interface

The code used to generate this example screen is shown below. Other screens could easily be written by generating code similar to this function. The full source code for the implementation of the PRD can be found in Appendix A.

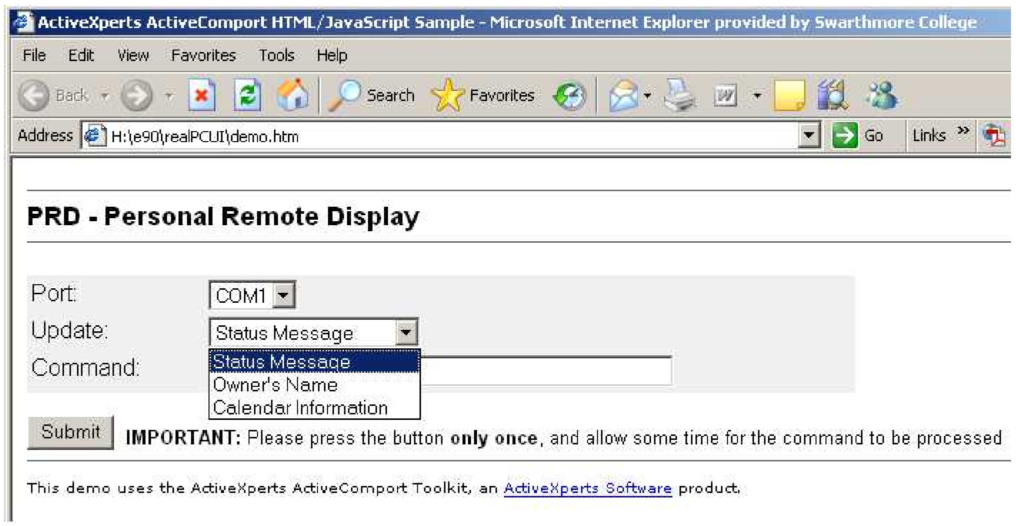
```
void mainscreen() {
    background(WHITE);
    draw_box_filled(0, 240, 0, 15, BLUE);
    print_string(username, usernamelength, WHITE, 2, 16, 1);
    draw_circle_filled(4, 7, 7, GREEN);
    draw_circle(4, 7, 7, BLACK);
    print_string(textbuffer,textbufferlength,BLACK, 2, 4, 17);

    print_string(timestamp, timestamplength, BLACK, 1, 4, 132);

    print_string(cm, cmlength, BLACK, 1, 4, 146); //cm: calendar message
}
```

3.2.4 PC "Owner" Interface

On the PC end of things, HTML, Javascript, and the proprietary ActiveCompot to develop a web-based GUI and communicate through the Zigbee XBee modules to the microchip. Since Javascript cannot inherently talk to the serial port on a Windows machine, the ActiveComport serial port Toolkit allows for this interaction to take place. HTML is used to specify the layout of the page and Javascript provides the functionality.

Two screenshots of the interface may be seen in Figures 8 and 9. The first is a demonstration of the options available for configuration: the status message, the owner's name, and the calendar information. Figure 9 more specifically demonstrates how to update calendar information. A second select menu appear when "Calendar Information" is selected from the Update menu, giving a option to add a new calendar item and the options to modify current calendar items if there are any. Using this system, owners can add, remove, and edit calendar items to be displayed on the LCD screen.

The full source code for the interface can be found in Appendix A. The section of the source code worth mentioning is how to send the commands. Upon the *Submit* button being pressed, a send function is triggered. Within that function, the Javascript determines how to tag the message and if any additional information should be sent, other than the tag and the message itself. An example of the most general tag and message is shown below.


ActiveXperts ActiveComport HTML/JavaScript Sample - Microsoft Internet Explorer provided by Swarthmore College
File Edit View Favorites Tools Help
Back Search Favorites
Address H:\e90\realPCUI\demo.htm
Go Links
Status Message
Owner's Name
Calendar Information
Submit

PRD - Personal Remote Display

Port: COM1

Update: Status Message

Command:

IMPORTANT: Please press the button **only once**, and allow some time for the command to be processed

This demo uses the ActiveXperts ActiveComport Toolkit, an ActiveXperts Software product.

Figure 8: An Example of the PC "Owner" Interface, Demonstrating the Update Select Menu

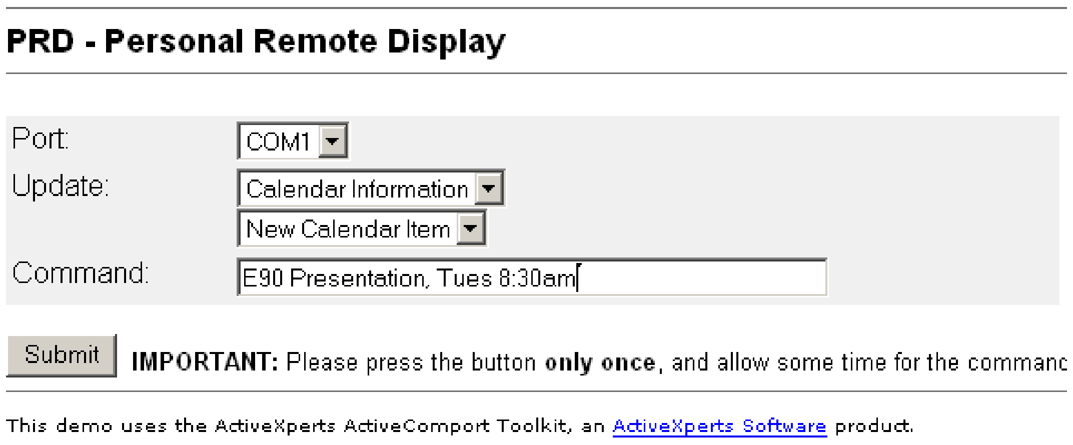
```
objComport.WriteString( "<"+updateOptionsTags[updateOptions.selectedIndex]+"><"+textCommand.value+">"
```

The full source iterates through all the peculiar tags and adds timestamps or extra specifier tags when needed. For example, if a calendar item is modified, a calendar item identity tag needs to be added in order to determine which calendar item to modify.

4 Final Product and Future Work

The project was completed as a robust protoype with many possible extensions. The user and owner interfaces function well and there are enough features to make this product easier to use than conventional methods of notifiying individuals of one's status and availability.

A high priority for future work would include building a container for the LCD parts, making the device mountable and portable. As a proof of concept implementation, this was trivial, but as a finished product, this is essential. Another high priority is allowing the information about the owner (the owner's name, calendar information, etc) to be stored on the hard drive of his or her personal computer as the information is updated or when the PC user interface program is closed. Power management concerns demand the implementation of a sleep mode.


PRD - Personal Remote Display

Port:	COM1
Update:	Calendar Information
	New Calendar Item
Command:	E90 Presentation, Tues 8:30am

IMPORTANT: Please press the button **only once**, and allow some time for the commanc

This demo uses the ActiveXperts ActiveComport Toolkit, an ActiveXperts Software product.

Figure 9: An Example of the PC “Owner” Interface, Demonstrating the Calendar Feature

Less urgent feature would include some sort of remote access, either by interfacing the device with Google Calendar (for calendar information) and Google Talk (for status and availability information) or by developing remote access software. Additional features could be added as well, as long as the device remained streamilined for use and does not become bloated with many specific features.

5 Conclusion

In the development of this product, much was learned about the nature of the interaction of hardware and software development and the dependencies between the two. Many precautions were taken to ensure robustness of the product and once the implementation moves past the protype stage and some of the features mentioned in the previous section are implemented, this product could possibly be produced for consumption, although specific hardware would need to be developed to reduce price, power consumption and size of the device.

Appendix A:

Source Code

Source code in C for the PIC18F67J10

```
#include "H:\e90\code\newpic.h"

#define MAXBUFF 32
#define MAXUNBUFF 32
#define MAXCALITEMS 5
#define MAXCALBUFF 32

#define GREEN 56
#define DARKGREEN 32
#define YELLOW 63
#define BLUE 192
#define RED 7
#define BLACK 0
#define WHITE 255
#define VIOLET 196

char textbuffer[MAXBUFF];
int textbufferlength;
int screen;
char username[MAXUNBUFF];
int usernamelength;
char timestamp[MAXUNBUFF];
int timestamplength;
int recentlyTouched;
int numCalItems;
int calItemslength[MAXCALITEMS];
char calItems[MAXCALITEMS][MAXCALBUFF];


void clear();
void background(int color);
void backlight_off();
void backlight_on();
void set_color(int color);
void set_xy(int x, int y);
void draw_box_filled(int x1, int x2, int y1, int y2, int color);
void draw_circle(int radius);
void draw_circle(int radius, int x, int y, int color);
void draw_circle_filled(int radius);
void draw_circle_filled(int radius, int x, int y, int color);
void draw_arc(int radius, int begin_angle, int end_angle);
void draw_arc(int radius, int x, int y, int color, int begin_angle, int end_angle);
void select_font(int font);
void print_string(char* mychars, int length, int color, int font, int x, int y);

void led_fanfare();
void bootscreen();
void mainscreen();
void calescreen();
int waitForMessage();
void getMessage();
void getTimestamp();
void getUsername();
void getCalItem();

void main(){
   char c;
   int strcounter;
   int i, mes;

   setup_adc_ports(NO_ANALOGS|VSS_VDD);
   setup_adc(ADC_OFF|ADC_TAD_MUL_0);
   setup_psp(PSP_DISABLED);
   setup_spi(SPI_SS_DISABLED);
   setup_wdt(WDT_OFF);
   setup_timer_0(RTCC_INTERNAL);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_timer_4(T4_DISABLED,0,1);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

   //boot
   i=0;
   while(i<10){
      fputc('c',XBEE);
      delay_ms(250);
      i++;
   }
   bootscreen();
   led_fanfare();
   screen=0;
   numCalItems = 0;
   for(i=0;i<=MAXBUFF;i++){textbuffer[i]=0;}
   username[0]='A';username[1]='l';username[2]='l';username[3]='i';username[4]='s';username[5]='o';
   username[6]='n';username[7]=' ';username[8]='B';username[9]='a';username[10]='r';username[11]='l';
   username[12]='o';username[13]='w';
   usernamelength=13;

   timestamp[0]='S';timestamp[1]='t';timestamp[2]='a';timestamp[3]='t';timestamp[4]='u';timestamp[5]='s';
   timestamp[6]=' ';timestamp[7]='s';timestamp[8]='e';timestamp[9]='t';timestamp[10]=':';timestamp[11]=' ';
   timestamp[12]='0';timestamp[13]='4';timestamp[14]='/';timestamp[15]='2';timestamp[16]='8';
   timestamp[17]='/';timestamp[18]='0';timestamp[19]='8';timestamp[20]=' ';timestamp[21]='3';
   timestamp[22]=':';timestamp[23]='1';timestamp[24]='7';timestamp[25]='p';timestamp[26]='m';
   timestamplength=26;

   recentlyTouched = 0;

   mainscreen();
   while(1){
      mes = waitForMessage();
      if (mes==1) { mainscreen();led_fanfare(); }
      else if (mes==2) {
         if (recentlyTouched) {
            recentlyTouched = 0;
         } else if (screen==0) {
            calescreen();
            screen =1;
         } else if (screen==1) {
            mainscreen();
            screen =0;
         }
      }
   }
}


void clear() {
   fputc('!',LCD);
}

void background(int color) {
   set_color(color);
   clear();
   fputc('!',LCD);
}
void backlight_off()
{
  // #use rs232(baud=115200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stop=1)
   fputc('#',LCD);
   //return 0;
}
void backlight_on() {
  // #use rs232(baud=115200,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stop=1)
  // output_high(pin_b0);
   fputc(34,LCD);
}
void set_color(int color) {
   fputc('$',LCD);
   fputc(color,LCD);
}
void set_xy(int x, int y) {
   fputc(37,LCD);
   fputc(x,LCD);
   fputc(y,LCD);
}

void draw_box_filled(int x1, int x2, int y1, int y2, int color) {
   set_color(color);
   set_xy(x1,y1);
   fputc(67,LCD);
   fputc(x2,LCD);
   fputc(y2,LCD);
}

void draw_circle(int radius) {
   fputc(')',LCD);
   fputc(radius,LCD);
}
void draw_circle(int radius, int x, int y, int color) {
   set_color(color);
   set_xy(x,y);
   draw_circle(radius);
}

void draw_circle_filled(int radius) {
   fputc(57,LCD);
   fputc(radius,LCD);
}
void draw_circle_filled(int radius, int x, int y, int color) {
   set_color(color);
   set_xy(x,y);
   draw_circle_filled(radius);
}

void draw_arc(int radius, int begin_angle, int end_angle) {
   printf("%c",47);
   printf("%c",radius);
   printf("%c",begin_angle);
   printf("%c",end_angle);
}

void draw_arc(int radius, int x, int y, int color, int begin_angle, int end_angle) {
   set_color(color);
   set_xy(x,y);
   draw_arc(radius,begin_angle,end_angle);
}

void select_font(int font) {
   fputc(43,LCD);
   fputc(font,LCD);
}

void print_string(char* mychars, int length, int color, int font, int x, int y) {
   int i;
   select_font(font);
   set_color(color);
   set_xy(x,y);
   fputc(45,LCD);
   //fputc(mychars[0],LCD);
   for(i=0;i<=length;i++) {
      fputc(mychars[i],LCD);
   }

   //fprintf(LCD,my_string);
   fputc(0,LCD);
}

void led_fanfare() {
   output_low(pin_B5);
   delay_ms(500);
   output_low(pin_B4);
   delay_ms(500);
   output_low(pin_C0);
   delay_ms(500);
   output_high(pin_B5);
   delay_ms(500);
   output_high(pin_B4);
   delay_ms(500);
   output_high(pin_C0);
}

void bootscreen() {
   backlight_on();
   background(BLUE);
   draw_circle_filled(30, 120, 80, BLACK);
   draw_circle_filled(28, 120, 80, WHITE);
   draw_circle_filled(26, 120, 80, BLACK);
   draw_circle_filled(24, 120, 80, WHITE);

   //print_string("Personal Remote Display", 23, VIOLET, 3, 20, 30);
   //print_string("by ajb", 6, VIOLET, 2, 20, 60);
}

void mainscreen() {
   char cm[31];//calendar message
   background(WHITE);
   draw_box_filled(0, 240, 0, 15, BLUE);
   print_string(username, usernamelength, WHITE, 2, 16, 1);
   draw_circle_filled(4, 7, 7, GREEN);
   draw_circle(4, 7, 7, BLACK);
   print_string(textbuffer,textbufferlength,BLACK, 2, 4, 17);


   print_string(timestamp, timestamplength, BLACK, 1, 4, 132);

   cm[0]='T';cm[1]='o';cm[2]='u';cm[3]='c';cm[4]='h';cm[5]=' ';cm[6]='s';cm[7]='c';cm[8]='r';
   cm[9]='e';cm[10]='e';cm[11]='n';cm[12]=' ';cm[13]='t';cm[14]='o';cm[15]=' ';cm[16]='v';
   cm[17]='i';cm[18]='e';cm[19]='w';cm[20]=' ';cm[21]='C';cm[22]='a';cm[23]='l';cm[24]='e';
   cm[25]='n';cm[26]='d';cm[27]='a';cm[28]='r';cm[29]='.';cm[30]='\0';
   print_string(cm, 30, BLACK, 1, 4, 146);
}

void calescreen() {
   char cm[31];//calendar message
   char cal[9];
   int i;

   cal[0]='C';cal[1]='a';cal[2]='l';cal[3]='e';cal[4]='n';cal[5]='d';cal[6]='a';cal[7]='r';cal[8]='\0';
   background(WHITE);
   draw_box_filled(0, 240, 0, 15, DARKGREEN);
   print_string(cal, 8, WHITE, 2, 16, 1);
   draw_circle_filled(4, 7, 7, YELLOW);
   draw_circle(4, 7, 7, BLACK);
   //print_string(textbuffer,textbufferlength,BLACK, 2, 4, 17);
   for(i=0;i<numCalItems;i++) {
      print_string(calItems[i],calItemslength[i],BLACK, 2, 4, 17+12*i);
   }


   cm[0]='T';cm[1]='o';cm[2]='u';cm[3]='c';cm[4]='h';cm[5]=' ';cm[6]='s';cm[7]='c';cm[8]='r';
   cm[9]='e';cm[10]='e';cm[11]='n';cm[12]=' ';cm[13]='t';cm[14]='o';cm[15]=' ';cm[16]='r';
   cm[17]='e';cm[18]='t';cm[19]='u';cm[20]='r';cm[21]='n';cm[22]=' ';cm[23]='H';cm[24]='o';
   cm[25]='m';cm[26]='e';cm[27]='.';cm[28]='\0';cm[29]='\0';cm[30]='\0';
   print_string(cm, 30, BLACK, 1, 4, 146);
}

int waitForMessage() {
   int retval,i,cx,cl,tag;
   cx=0;
   cl=0;
   tag=0;

   if (kbhit(XBEE)) { cx=fgetc(XBEE); }
   if (kbhit(LCD)) { cl=fgetc(LCD); }

   retval = 0;
   //figure out how this works!!!
  // if (cx!=0) {led_fanfare();}
   if (cx==60){ //2) { //start of text signal

      //cx=fgetc(XBEE); //no idea why i need this extra one
      while(tag==0) {tag=fgetc(XBEE);} //oh boy tag!
      cx=fgetc(XBEE);//">", at least it should be...
      if (tag=='m') {getMessage();}
      else if (tag=='t') {getTimestamp();}
      else if (tag=='o') {getUsername();}
      else if (tag=='c') {getCalItem();}
      retval = 1;
   }
   if (cl!='\0') {
      retval = 2;
   }
   return retval;
}

void getMessage() {
   int i,c;
   fgetc(XBEE);
   for(i=0;i<MAXBUFF;i++){
      c=fgetc(XBEE);
      textbuffer[i]=c;
      if (c==62||c==124) { //end of text buffer
         i--;
         break;
      }
   }
   textbufferlength = i;
   //getTimestamp();
   for(i=12;i<MAXUNBUFF;i++){
      c=fgetc(XBEE);
      timestamp[i]=c;
      if (c==62) { //end of text buffer
         i--;
         break;
      }
   }
   timestamplength = i;
}

void getTimestamp() {
   int i,c;
   fgetc(XBEE);
   for(i=12;i<MAXUNBUFF;i++){
      c=fgetc(XBEE);
      timestamp[i]=c;
      if (c==62) { //end of text buffer
         i--;
         break;
      }
   }
   timestamplength = i;
}

void getUsername() {
   int i,c;
   fgetc(XBEE);
   for(i=0;i<MAXUNBUFF;i++){
      c=fgetc(XBEE);
      username[i]=c;
      if (c==62) { //end of text buffer
         i--;
         break;
      }
   }
   usernamelength = i;
}

void getCalItem() {
   int i,c,j,k,n;

   fgetc(XBEE);
   n = fgetc(XBEE)-48;//cal item to modify
   fgetc(XBEE);fgetc(XBEE);

   if (n==0) {j=numCalItems++;} //add a new cal item
   else {j=n-1;}
   for(i=0;i<MAXCALBUFF;i++){
      c=fgetc(XBEE);
      calItems[j][i]=c;
      if (c==62) { //end of text buffer
         i--;
         break;
      }
   }
   calItemsLength[j] = i;

   //if its a null string, get rid of the cal item!
   if (calItemsLength[j]==255) {
      for(i=j;i<(numCalItems-1);i++){
         for(k=0;k<calItemslength[i+1];k++){
            calItems[i][k]=calItems[i+1][k];
            }
            calItemslength[i]=calItemslength[i+1];
        }
        numCalItems--;
    }
}
```

Source code int HTML/Javascript for Owner Interface on the PC

```
<html>
  <head>
    <title>ActiveXperts ActiveComport HTML/JavaScript Sample</title>
    <object codebase="acomport.cab" classid="CLSID:68A2C188-A606-4841-AE8A-D58C6F6BE583" ></object>

<script language="JavaScript">

var objComport = new ActiveXObject ( "ActiveXperts.ComPort" );
var timestamp = new Date();
var updateOptionsTags = new Array();
var numCalItems = 0;
var calItems = new Array();
updateOptionsTags[0]="m"; //message
updateOptionsTags[1]="o"; //owner
updateOptionsTags[2]="c"; //calendar

var hours;
var minutes;
var month;
var isAM;
isAM=true;

function Send (){
  objComport.Device              = comboDevice.options [ comboDevice.selectedIndex].text;
  objComport.BaudRate            = 9600;
  objComport.ComTimeout          = 2000;
  objComport.LogFile             = "C:\\ComLog.txt";
  objComport.HardwareFlowControl = objComport.asFLOWCONTROL_DEFAULT;

  objComport.Open ();

  if( objComport.IsOpened == -1 ) {
    timestamp = new Date();
    hours = timestamp.getHours()
    minutes = timestamp.getMinutes()
    month = timestamp.getMonth() + 1 ;
    if (minutes < 10) {minutes = "0" + minutes}
    if (hours>12) { hours = hours%12; isAM=false;} else {isAM=true;}

    if (updateOptionsTags[updateOptions.selectedIndex]=='o'){
      objComport.WriteString("<"+updateOptionsTags[updateOptions.selectedIndex]+"><"+
        textCommand.value+">");
    } else if (updateOptionsTags[updateOptions.selectedIndex]=='m'){
      //timestamp
      if(isAM){
        objComport.WriteString("<"+updateOptionsTags[updateOptions.selectedIndex]+"><"+
          textCommand.value + "|"+month+"/"+timestamp.getDate()+"/"+
          timestamp.getFullYear()+" "+hours + ":" + minutes + " AM>");
      } else{
        objComport.WriteString( "<"+updateOptionsTags[updateOptions.selectedIndex]+"><"+
          textCommand.value + "|"+month+"/"+timestamp.getDate()+"/"+timestamp.getFullYear()+
          " "+hours + ":" + minutes + " PM>");
      }
    } else if (updateOptionsTags[updateOptions.selectedIndex]=='c'){
      //send message to lcd
      objComport.WriteString( "<"+updateOptionsTags[updateOptions.selectedIndex]+"><"+
      calendarOptions.selectedIndex+"><"+textCommand.value+">");

      //update internal values
      if (calendarOptions.selectedIndex==0) {//new item
        calItems[numCalItems]=textCommand.value;
        calendarOptions.options[numCalItems+1]= new Option(textCommand.value);
        numCalItems++;
      } else if (textCommand.value=="") { //delete
        for (i=(calendarOptions.selectedIndex-1); i < numCalItems-2; i++) {
          calItems[i]=calItems[i+1];
        }
        calendarOptions.options[calendarOptions.selectedIndex]=null;
        numCalItems--;
      } else {
        calItems[calendarOptions.selectedIndex]=textCommand.value;
calendarOptions.options[calendarOptions.selectedIndex] = new Option(textCommand.value);
      }
    }

    calendarOptions.selectedIndex;
  }

objComport.Close ()
}

function ListDevices () {
  nCount  = objComport.GetDeviceCount ();

  for ( i = 0 ; i < nCount ; i++ )
  {
    comboDevice.options [ i ] = new Option ( objComport.GetDevice ( i ), "" );
  }

  for ( i = 1 ; i < 9 ; i++ )
  {
    comboDevice.options [ i + nCount - 1 ] = new Option ( "COM" + i , "" );
  }
}

function ListUpdateOptions () {
  updateOptions.options[0]= new Option("Status Message","");
  updateOptions.options[1]= new Option("Owner's Name","");
  updateOptions.options[2]= new Option("Calendar Information","");

  calendarOptions.style.display = 'none';
  calendarOptions.options[0]= new Option("New Calendar Item","");
}

function CheckForCal () {
  if (updateOptions.selectedIndex==2)//calendar was selected
  {
    calendarOptions.style.display = 'block';
  } else { calendarOptions.style.display = 'none'; }
}

</script>
</head>

<body onload="ListDevices();ListUpdateOptions()">
  <font face="sans-serif" size="2">
  <hr size="1" color="#707070">
  <b><font size="4">PRD - Personal Remote Display</font></b><br>
  <hr size="1" color="#707070">
  <br>
  <table border="0" bgcolor="#f0f0f0" >
    <tr>
      <td width="120" valign="top">Port:</td>
      <td width="450">
        <select size="1" id="comboDevice" >
       </select>
      </td>
    </tr>
<tr>
      <td width="120" valign="top">Update:</td>
      <td width="450">
        <select size="1" id="updateOptions" onChange="return CheckForCal()">
        </select>
        <select size="1" id="calendarOptions">
        </select>
      </td>
    </tr>
    <tr>
      <td valign="top">Command:</td>
      <td>
        <input size="50" type="text" id="textCommand" value="Available." ID="Text1">
        </td>
    </tr>
  </table>
  <br>
  <input type="button" onclick="Send()" value="Submit">&nbsp;&nbsp;<b>IMPORTANT:</b>
     Please press the button <b>only once</b>, and allow some time for the command to
      be processed </font>
  <br>
  <hr size="1" color="#707070">
  <font size="1" face="Verdana">This demo uses the ActiveXperts ActiveComport Toolkit,
    an <a href="http://www.activexperts.com">ActiveXperts Software</a> product.</font></b>
  </body>
</html>
```