|  |
| --- |
| UNIVERSITY OF NOTRE DAME-ELECTRICAL ENGINEERING |
| Final Report |
| Team AutoBev |
| **Elizabeth Clark, Lorena Garcia, Alex Macomber, Mark Pomerenke** |
| **5/6/2011** |

Contents

[1 Introduction: 4](#_Toc292362483)

[1.1 Problem Description: 4](#_Toc292362484)

[1.2 System Requirement: 4](#_Toc292362485)

[1.3 Overall System: 4](#_Toc292362486)

[1.4 Subsystem and Interface Requirements: 5](#_Toc292362487)

[**1.5** Future Enhancement Requirements: 6](#_Toc292362488)

[1.6 High Level Description: 8](#_Toc292362489)

[1.7 Expectations: 10](#_Toc292362490)

[2 Detailed Project Description: 10](#_Toc292362491)

[2.1 System Theory of operation: 10](#_Toc292362492)

[Power Supply: 11](#_Toc292362493)

[Microcontroller: 12](#_Toc292362494)

[Beverage Dispensing: 12](#_Toc292362495)

[User Interface: 12](#_Toc292362496)

[Bartender Interface: 12](#_Toc292362497)

[Interfaces: 12](#_Toc292362498)

[2.2 Detailed operation of Subsystems 13](#_Toc292362499)

[Detailed Operation of Card Scanner 13](#_Toc292362500)

[Detailed operation Android application 14](#_Toc292362501)

[Detailed operation of User Interface 16](#_Toc292362502)

[Detailed operation Bartender Interface 24](#_Toc292362503)

[Detailed operation of Microcontroller 24](#_Toc292362504)

[Detailed operation of Android application 26](#_Toc292362505)

[Detailed operation of Beverage Dispensing and Sensing 29](#_Toc292362506)

[2.3 Interfaces and Sensors: 30](#_Toc292362507)

[PC To Microcontroller Interface: 30](#_Toc292362508)

[Bartender To User Interface: 32](#_Toc292362509)

[3 System Integration Testing 34](#_Toc292362510)

[3.1 Describe how the integrated set of subsystems was tested. 34](#_Toc292362511)

[3.2 Show how the testing demonstrates that the overall system meets the design requirements 34](#_Toc292362512)

[4 Users Manual/Installation manual 34](#_Toc292362513)

[4.1 How to install your product 35](#_Toc292362514)

[4.2 How to use your product 35](#_Toc292362515)

[4.3 How the user can tell if the product is working 36](#_Toc292362516)

[5 To-Market Design Changes 38](#_Toc292362517)

[6 Conclusions 39](#_Toc292362518)

[7 Appendices 39](#_Toc292362519)

# Introduction:

## Problem Description:

One of the biggest challenges to both customers and owners of drinking establishments is that of lengthy wait times associated with the fulfillment of an order. In such an environment, profit for the owner, and happiness of the customer, is directly dependent upon efficiency of product deliveries. At many drinking establishments, customers waste many minutes each night waiting to be noticed by a bartender. Once noticed, the customer must then explain his or her order to the server. After making the order, the server then charges the customer. During this process, there is no guarantee that the server will hear the customer’s order correctly, allowing for the possibility of inaccurate order fulfillment. Furthermore, if the customer chooses to pay with a credit card, the procedure is made even longer, due to the fact that the credit card verification process is very time-consuming.

With the technology available in our current society, it is apparent that waiting times can and should be reduced. The crucial minutes wasted while the customer explains their order to the server and while the server charges the customer, coupled with the possibility of an inaccurate order, can be eliminated through the utilization of current technology. By implementing a computer system that provides a means to order, pay for, and pour a beverage, without dependence on a restaurant employee, the time between request and delivery of a final product can be greatly reduced and streamlined. Additionally, through the elimination of verbal ordering, the accuracy of orders can be increased. Using a digital ordering process would also enable establishments to serve customers in the order in which they requested a drink, instead of in a random fashion.

It is reasonable to say that an establishment can serve more drinks by reducing wait times. This would inevitably lead to more revenue. Furthermore, by serving customers in order, instead of in a random fashion, customer satisfaction would also increase. Through the automation of the order and payment processes, the workload of employees can be decreased while increasing order fulfillment accuracy.

## System Requirement:

Tables 1.1, 1.2, and 1.3 below show the requirements for the overall system, for the general subsystems, and for possible future enhancements as established at the time of the project proposal.

## Overall System:

|  |  |
| --- | --- |
| Overall System Requirements | |
| **General** | Must be capable of receiving and processing drink orders digitally  Must use simple user commands for navigation (single click)  Must have layer of protection between CPU and other sensitive electronics and users/ beverages |
| **Size** | Must fit onto a floor area no larger than 2’ x 2’ (not including any keg) |
| **Power** | Must be powered by wall plug in 120V AC |
| **Compatibility** | Must be able to connect to standard keg  Must be expandable to include more extensive drink orders  Must be in English  Must be in compliance with current health standards |

Table 1.1 Overall System Requirements

## Subsystem and Interface Requirements:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Magnetic Card Reader | | | | |
| **General** | Must be able to scan credit cards and send information to the PC | | | |
| **Size** | Must not be bigger than 6”x6” | | | |
| **Power** | Must be powered through USB interface | | | |
| **PC Software** | Must be able to interface to a PC through USB  Must be able to emulate a USB Human Interface Device (HID) keyboard  Must turn on and off with PC | | | |
| User Interface | | | | |
| **General** | Must be implemented on a standard monitor | | | |
| **Power** | Screen must be powered by 120 V AC | | | |
| **Compatibility** | Must work on Windows PC  Will Be Programmed Using Microsoft Visual Studio  Will Program in C# Visual | | | |
| Bartender Interface | | | | |
| **PC Software** | | | Must be capable of running on a low performance windows machine  Must list drink items in order in a FIFO queue  Must assign item number to new drink items for claims purpose (will not need to issue ticket)  Must be capable of adding orders to the queue from remote client (customer interface)  Must be capable of deleting queue items from local user  Must be able to handle bi-directional communication via Local Area Ethernet  Must be compatible with UI program  Bartender Server must be multi-threaded for expandability | |
| Microcontroller | | | | |
| **General** | | | | Must handle at least 8 digital inputs \* Must handle at least 4 analog inputs\* Must supply at least 4 digital outputs \*  Must indicate the state of digital outputs with LED’s  Must be programmable with C software  Must have a universal asynchronous receiver transmitter (UART) Must be able to communicate serially with CPU over USB connection Must have non-volatile memory  Must be capable of 5V operation. Must have On/Off switch  Must have reset function  \*will only use 2/1/1 I/AI/O ports for demonstration, but built in for expandability |
| **Power** | | | | Powered by 12 V supply from wall converter |
| **Compatibility** | | | | Will receive signal from computer  Will receive signal from flow sensor |
| Beverage Dispensing and Sensing | | | | |
| **General** | | Must have keg to pour drink from with appropriate keg components (spigot, keg adapter, pressure valve, tubing, and gas tank). Must have a solenoid valve to only allow beverage to pour through if already paid for. Must have a flow sensor to keep track of how much beverage has been poured.  Must have Cup Senor to detect if cup is present  Must have emergency stop | | |
| **Power** | | Must be powered via plug into a 120 VAC outlet Must create intermediate voltage power supplies of 12 V and 5V regulated | | |
| **Valve Permission Solenoids** | | Solenoids must receive 12 V across the coil to activate  Must be able to control 12 V solenoid signal with a 5 V output  Amplifier FET must be able to handle 0.53 A, and a max Vdd of at least 20V  Must connect to 0.375” OD, .25” ID tubing | | |
| **Flow Sensing** | | Requires specified resistive network for proper signal output  Requires 5-24V power supply  Must connect to 0.375” OD, .25” ID tubing | | |
| **Cup Sensor** | | Must detect if cup is present underneath spigot  Must toggle input to microcontroller from 5 to 0 V | | |
| **Emergency Stop** | | Must allow user to stop beverage flow if anything goes wrong  Should not be software actuated | | |
| **Pressure System** | | Must create a regulated pressure of 12 psi  Must attach to keg input by coupling to connector | | |
| **Microcontroller Software** | | Must use less than 8K of program memory  Must be able to communicate with solenoid(open/close it).  Must be able to get input from flow sensor via interrupt  Must be able to communicate with User Interface via USB  Must operate with a baud rate of 9600 | | |
| PC to Microcontroller USB Interface | | | | |
| **General** | | Must have USB interface  User Interface PC must be able to communicate with microcontroller by sending bytes  Microcontroller must be able to communicate with  User Interface PC by sending bytes | | |
| Bartender/User PC UDP Interface | | | | |
| **General** | | Must connect using network UDP protocol  Must operate on hard line and wireless connections to the network  Bartender must be initialized before the User can access system  User connections must receive updated bartender data upon request  Must be send user data packets to the server with high reception certainty | | |

Table 1.2 Subsystem and Interface Requirements

## Future Enhancement Requirements:

|  |  |
| --- | --- |
| Future Enhancement Requirements | |
| **Online Database** | Must allow users to sign up to access online database that keeps track of past drink orders  Must show information such as time, quantity, order, and expense |
| **Charge Credit Cards** | Must be able to complete charging process  Upgrade from prototype that simply stores credit card information in a local database without charging the account |
| **Serve Mixed Drinks** | Must be able to properly make and serve “mixed drinks” in an automated manner |
| **Cognitive Testing** | Must be able to determine if customer is in suitable state for another drink |
| **Interconnect**  **Multiple Kiosks** | Must allow for multiple dispensing units to be interacting with a centralized bartender interface within a single establishment |

Table 1.3 Future Enhancement Requirements

Table 1.1 shows the overall system requirements necessary to solve the problem. The requirements in this table were all completed. Drink orders can currently be received and processed digitally through the use of a simple user interface. All sensitive electronics have been placed in encasings that provide protection from both users, and beverages. The unit is reasonably small enough to fit in a normal drinking establishment and can be powered through a normal wall outlet. It takes as an input, a standard keg, allows for the bartender to automatically update the drink list for the customers to view, and is implemented in English and in compliance with current health standards.

Table 1.2 shows the subsystem and interface requirements necessary to solve the problem.

All requirements for the magnetic card reader subsystem were completed. A USB connection between reader and computer allows for the device to be powered, and also for the credit card track information to be conveyed to the user interface.

The requirements for the user interface subsystem were also met. This interface was programmed using Microsoft Visual Studio. It is implemented in Visual C#, using a Windows Form application. All forms within the application are sized to be readable but a user on a standard PC monitor powered by a wall socket. One additional requirement that was added to the user interface was the ability to allow users to create a username and password that would be tied to their account such that they would be able to use the Android Smartphone app. For simplicity, it was decided that the username must be all letters, and the password must be a four digit pin. Thus, the user interface had to be able to verify these two traits. Another additional requirement that was encountered was the ability of the user interface to receive updated drink lists and prices from the bartender. This requirement was successfully completed.

The initial requirements for the bartender interface were completed. Additional requirements were added to this subsystem throughout the design completion process. During this process, it was discovered that the bartender interface must be able to store and extract data from a database. A local database was created using MySQL. At the end of every night, it is possible for the bartender to save the data information to a text file in html format. The bartender can also choose to upload and display data that has been saved from previous nights. Another requirement that was added to the bartender interface was that of the ability to receive drink orders from a Android Smartphone, along with the ability to send a message back to the Android upon receiving an order, and the ability to send a text message to the phone when the drink is ready. This requirement was fulfilled using UDP communication. The bartender interface is able to extract the username and password from its local database. It then must be verified that the username does exist, and that the password is correct for that username. If not, the user must receive a respond from the bartender interface (via text messaging) noting that the either the username did not exist, or that the password was incorrect. If both are correct, then the bartender must issue a UDP response to the Android from which the order was placed telling the user what number order they are. The bartender also needed to have the ability to check to see if a person (when trying to set up a username at an AutoBev kiosk) already had a username. This required conducting a query of the local database. This requirement was completed.

The microcontroller subsystem requirements changed very slightly. Instead of being capable of 5 volt operation, the requirement was changed to having the microcontroller be capable of five volt operation.

The beverage dispensing and sensing subsystem requirements were also met. The unit will only allow for a beverage to be dispensed given that it has been paid for, and that there is a cup present. The amount of volume that has been poured is constantly polled and reported back to the user interface. The power requirement of our system changed to being a regulated power of five volts. This was implemented through the use of a voltage regulator. Thus, the solenoid valve is now controlled with a five volt signal from an output pin on the microcontroller. The cup sensor was also changed from a light sensor to a limit switch. This switch had to be sensitive enough to allow for trigger due to a plastic cup, but strong enough to not break easily due to a careless user.

The PC to microcontroller USB interface subsystem requirements did not change and were successfully implemented.

FUTURE ENHANCEMENTS????

## High Level Description:

The AutoBev System consists of a main user interface, a bartender interface, a beverage dispensing and sensing unit, a localized database, and a Android Smartphone application. The user and bartender interfaces were programmed using Microsoft Visual Studio.

The user interface enables a customer to begin a session by swiping a credit card. After this occurs, the user interface verifies that the track read by the magnetic stripe reader corresponds to a credit card. Once this is done, the user is able to chose if they would like to order a specialty drink (to be made by the bartender), if they would like to pour their own beverage, or if they would like to set up a username and password to be used with a Android Smartphone application.

If the user chooses the option to order a specialty drink, they are presented with a list of drinks and prices from which they may order. The user interface checks for an updated list every time a person begins a session. Once an order has been placed, it is sent to the bartender interface to be added to a drink list queue. The bartender interface sends a message back to the user indicating what drink order number they are, along with how many drink orders are in front of them. The bartender interface appropriately updates the total to be charged to that customer within the local database.

If the user chooses the option to pour their own beverage, they are then presented with the option to select a predetermined size (either twelve, sixteen, or sixty ounces) or to pay per ounce. Updates in price per ounce are checked for every new user. The user must then place their cup against a limit switch and hit a green start button on the user interface to begin pouring. When the user is done pouring, they must stop the flow and proceed to the checkout. At this time, the bartender interface will be informed of the amount of volume poured, and the cost of the purchase. This cost will be added to the user’s total in the local database.

The beverage flow is controlled by a PIC18F4320 microprocessor and a solenoid valve. The microcontroller receives instructions from the user interface via serial communication. The user interface sends a single byte to the microcontroller indicating the volume limit of the beverage to be dispensed, and whether or not the user wants to begin pouring or stop pouring. Once a size has been set and the user indicates that they would like to begin pouring, the microprocessor polls an input to check if for the presence of a cup. Cup presence is determined using a limit switch. The switch is constantly polled while the unit is pouring to ensure that a cup is still present. If a cup is there, the solenoid valve opens and allows for liquid to be dispensed. Simultaneously, the flow sensor begins to deliver pulses to another input on the microcontroller. The pulses are used to keep track of the amount of volume that has been dispensed. Periodically, the microcontroller sends a status byte (indicating if it is still pouring and a cup is still present) and two bytes containing the volume dispensed, back to the user interface. At any time, if a cup is not present, the user is notified and the system stops dispensing. Otherwise, the system stops dispensing when either the volume limit is reached or the user indicates that he or she is finished pouring. At this time, the solenoid valve is closed.

If the user chooses to set up a username and password for a Smartphone, they are presented with a form that allows him or her to enter a username (consisting solely or letters) and a password (a four digit pin). Once these requirements are met, the user can choose to save the username and password. These are sent to the bartender interface which then queries the local database to see if a user already has a username and password. If they do not, the database is updated and the chosen username and password and stored to be associated with the credit card number that was initially read by the magnetic stripe reader.

After proceeding through one of these options, the user may chose to order again, or to checkout of the AutoBev system. If they want to order again, their total will be continuously updated in the database. If they chose to check out, the system will wait for a new user to swipe a credit card to begin a new session.

The AutoBev system has the capability to receive drink orders from a Android Smartphone application. As was described above, a user may set up a username and password at the main AutoBev kiosk. These are associated with a credit card track number and a nightly total. A customer must enter their username and password into the phone application and then pressed a button indicating the specialty drink they desire to order. Before the order is sent to the bartender, it is verified that the customer is willing to pay the cost of the drink. Once this is done, the bartender interface verifies the existence of the username, and that the user has entered the correct password. If the username or password is incorrect, the phone will be sent a text message indicating a failure. Otherwise, the customer is sent a message with his or her drink order number, along with how many orders are in front of them. When the customer’s drink is finished and cleared from the bartenders drink queue, the phone is sent a text message indicating that the order has been completed.

## Expectations:

The design met the expectations of Team AutoBev. The user interface provides customers with an easy-to-use form that enables automated ordering of specialty drinks, as well as the capability to pour your own beverage. The volume that has been poured is kept track of using pulses from the flow sensor. The accuracy of this measurement is as was expected. It was known that 5600 pulses corresponded to a single liter of volume being dispensed. The number of pulses detected is reported to the user interface where pulses are converted to volume. Based on multiple trials, it has been seen that this method is extremely accurate. Despite the fact that there is the possibility of a pulse being missed, the accuracy is still very high due to the low amount of volume associated with each pulse.

The navigation through the user interface is simple and secure. It provides the user with up-to-date prices for specialty drinks, as well as for the self-dispensing beverage. The bartender is able to view a drink queue indicating the drinks to be made. The bartender interface allows the bartender to clear drinks that have been made, as well as the ability to save and load database information from a specified date. The bartender is also able to process a drink request from a Android Smartphone. In both of these cases, the user is informed of what order number they are as well as how many orders are in front of them. The Android application exceeded the expectations of Team AutoBev. This part of the AutoBev system has proven to be very accurate. The appropriate drink order is always displayed on the bartender interface and the cost of that drink is accurately added to the user’s nightly total.

The only expectation that was not met was the expectation that the user would only need to have access to the mouse to use the user interface. Because the Android application was implemented, it was necessary to change the system to allow for the user to also have access to a keyboard (on which they can enter a username and password).

# Detailed Project Description:

## System Theory of operation:

A user is able to interact with the AutoBev system via a kiosk located within the establishment or through the use of a Android Smartphone. When a user orders a specialty drink, the order is converted to bytes and sent as a datagram packet to the bartender. In order to send the order, the IP address of the bartender must be known by the customer. The bartender (or server) accepts input from any IP address. The server sends messages back to the user interface or Android by also using UDP communication with datagram packets. The bartender can choose to save or load the database information from a given night in a text file. The text file is written using html format. The system is displayed on two PC monitors that are powered through a standard wall outlet.

A user is able to control the dispensing of a beverage through the user interface. A single byte indicating desired size and action (start or stop pouring) by the user is sent from the user interface to the microcontroller. A Universal Asynchronous Receiver/ Transmitter (UART) is used to send the byte. The RS-232 protocol is used. The byte is processed by microcontroller code which is implemented in C. The microcontroller then sets control signals to either open or close the solenoid valve. It also polls an input port to determine if a cup is present under the dispenser. When liquid begins to flow, the flow sensor begins to pulse and these pulses are processed as interrupts by the microcontroller. Each time an interrupt occurs, the total pulses are updated. A status byte, along with two bytes holding the number of pulses detected are periodically sent via the UART to the user interface so that the user can see how much volume he or she has dispensed.

Figure 1 below shows a block diagram of the entire AutoBev system.

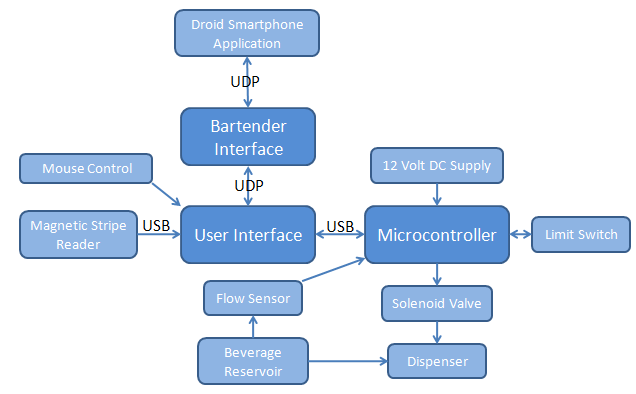


Figure 1 System Block Diagram

### Power Supply:

A twelve-volt DC power brick power the board. It will be used to provide a five volt regulated source to the solenoid valve.

### Microcontroller:

The microcontroller will receive data from the user interface via serial communication through the UART. It will report its status and the volume of liquid that has been dispensed back to the user interface. The microcontroller will control the solenoid valve to either allow or prohibit beverage dispensing. Additionally, the microcontroller will receive input from a limit switch detecting the presence of a cup.

### Beverage Dispensing:

While the user is pouring, the beverage will flow through a flow sensor that delivers pulses to the microcontroller. One liter of volume corresponds to 5600 pulses. The beverage dispensing unit will be controlled with a solenoid valve which is turned on and off by the microcontroller.

### User Interface:

The user interface will take input from a mouse, a keyboard, and a magnetic stripe reader. The mouse and keyboard will be used to allow the customer to set up a username and password, and to allow the customer to navigate through the interface. The magnetic stripe reader will be used to obtain credit card information from the customer. The user interface will direct input to both the microcontroller and the bartender interface. It will be displayed on a standard PC monitor.

### Bartender Interface:

The bartender interface will take input from the Android Smartphone app and the user interface. It will receive data via a UDP connection. It will respond to the user indicating their drink number, and the number of drinks in front of them. It will be displayed on a standard PC monitor

### Interfaces:

User/Android to Bartender: UDP connection using sockets, send orders in bytes that can be decoded into strings

User to Microcontroller: asynchronous communication using Universal Asynchronous Receiver/ Transmitter (UART), following RS-232 protocol

Power to Main Board: Wires

-Schematic (hardware) or flow chart (software) for subsystem

-function of subsystem

-interfaces to other subsystems

-how subsystem was tested

-why engineering decision were made (choice of processor, kind of RF interface, choice of programming language)

-schematics and software need to be described in addition to schematic and code

-hardware: describe how circuit works (how amplifier circuit and filters are designed/ operate)

-software: describe overall flow of the code (interrupt driven, state diagram)

-if there are communications protocols involved in subsystem, description and state transition diagrams should be included

-subsystem testing, how this subsystem was tested to ensure functionality

## Detailed operation of Subsystems

### Detailed Operation of Card Scanner

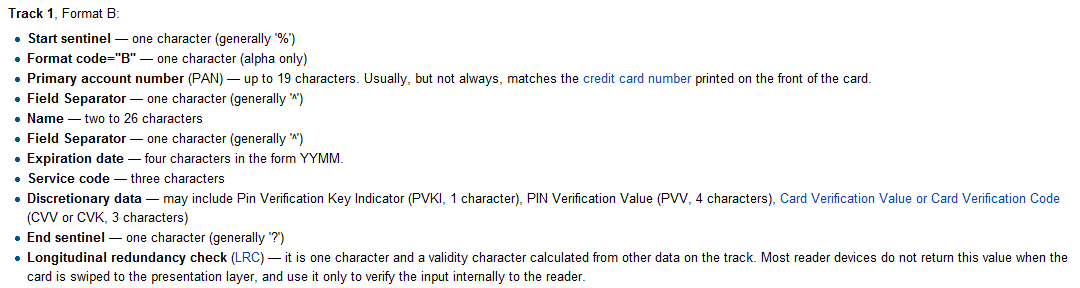
-how subsystem was tested

-subsystem testing, how this subsystem was tested to ensure functionality

Liz – are we deleting the below paragraph?

*The card scanner subsystem is used to extract credit card information from a user before they start the ordering process. At the beginning of a session, a customer must first swipe their credit card. The number on the credit card is sent over the serial connection to the user interface where it is verified to be a valid credit card number. In the AutoBev system, the first track of the card is read and stored as a means to identify the customer. For the whole night, every time a customer orders a drink, the total associated with the track read by the MSR is updated.*

The card scanner subsystem consists of a magnetic stripe reader (MSR). The MSR is located alongside the customer interface so that the customer can scan his or her debit or credit card. The scanner is a “plug and play” device that simply connects to a USB port of any standard PC and emulates keyboard inputs without the need for additional software. The card scanner was programmed to output track one of the credit card according to the ISO/ IEC 7813 standard. This standard defines properties for the cards used in financial transactions such as debit or credit cards. There are two and sometimes three tracks on the cards that conform to this standard. The format of track one is shown below:



http://en.wikipedia.org/wiki/Magnetic\_stripe\_card

Figure 2. Track 1 Format B

After the user swipes his or card, the information contained in track one is sent over a serial connection to the user interface. Here, the track number is checked against the standard seen above in Figure 2 to ensure that it is a valid credit card number. If it is determined to be a valid number, an account is created for that track number. All future purchases associated with that track will be added to a total kept in a local database on the bartender interface.

The MSR chosen for this project was Unitech’s MS240 Magnetic Stripe Reader. This product meets all stated requirements for reading credit cards and is relatively inexpensive. The MS240 interfaces with both a Mac and a PC and is able to emulate a keyboard. An additional benefit of the MS240 is that it is programmable. By using the ‘Reader Configuration Manager,’ it is possible to download settings to the MS240. These include, but are not limited to, turning a beeper on or off when a card is read, specifying which tracks to read, deciding on how information is displayed on the PC end, and choosing the type of ‘end’ character. The end character is what is sent to the PC to indicate that the reading has finished. In our case, the end character was set to be the enter key.

In order to test the functionality of this subsystem, the MSR was plugged into a serial port on a standard PC while a text file was opened. It was verified, using ten different credit cards, that the MSR was in fact reading track one of the credit cards. The MSR was then connected to the PC while the user interface was running. It was verified that when the reader read a valid credit card, the user interface proceeded to the second form in the drinking ordering process. This indicated that the MSR had read a track and then sent the appropriate end character, the enter key.

### Detailed operation Android application

-Schematic (hardware) or flow chart (software) for subsystem

**-function of subsystem**

The Android application allows the user to interact with the AutoBev system with his or her smartphone. The app has all the functionality of the AutoBev system contained at the kiosk except for the ability of the customer to pour his or her own beer.

Before using the AutoBev Android application, the user must sign in to an AutoBev kiosk at the bar. At the kiosk they will click on Connect to Phone after swiping his or her credit card. The user will then be prompted to enter a username and password which, along with the credit card track, will be sent to the bartender and saved to the database.

When the user opens the app he or she is presented with a welcome screen that contains two textboxes and six buttons shown in the figure below. The user enters his or her username and password into the respective textboxes and then clicks a button corresponding to the drink they wish to order. A message is sent over the wireless network using to the bartender’s PC containing the username, password, phone number, IP address of the phone and the desired drink of the user.

The bartender’s PC will verify that the username provided exists in the current database and that the password matches the username. If either of those conditions occur, the bartender will immediately send a message back to the phone telling the user to try again.

If the username and password are correct, the order is stored in the bartender’s queue. A message is then sent to the app telling the user what drink number is his or her and the drink currently being served.

When the drink is filled by the bartender (i.e. the bartender clears it from his or her queue) the bartender’s PC will send a text message and/or email to the Android user alerting him or her that the drink is ready.

**-interfaces to other subsystems**

The Android application will only interface with the bartender PC. The interface will involve a two way communication using the User Datagram Protocol (UDP) to send messages over the wireless network. Messages will also sent to the Android app via Short Message Service (SMS) text messages.

The Android application will have a separate receiving thread that is constantly looking for packets from the bartender. When the bartender sends a message to the application, the thread checks what the message’s purpose is and acts accordingly.

When a user sends a drink order to the bartender, the first character of the message tells the bartender’s program that the message is from the Android and the bartender’s PC then extracts the information from the message to store a drink.

The SMS messaging works using the Simple Mail Transfer protocol (SMTP). The C# code uses an email account setup using Gmail. The code needs to know the phone number of the app, the service of the phone (i.e. Verizon, Sprint) and the email address and password of the sending account.

**-how subsystem was tested**

**-why engineering decision were made (choice of processor, kind of RF interface, choice of programming language)**

**-schematics and software need to be described in addition to schematic and code**

**-hardware: describe how circuit works (how amplifier circuit and filters are designed/ operate)**

**-software: describe overall flow of the code (interrupt driven, state diagram)**

**-if there are communications protocols involved in subsystem, description and state transition diagrams should be included**

**-subsystem testing, how this subsystem was tested to ensure functionality**

### Detailed operation of User Interface

Figure 3 below displays a flow chart representing the paths that the user may take through the interface.

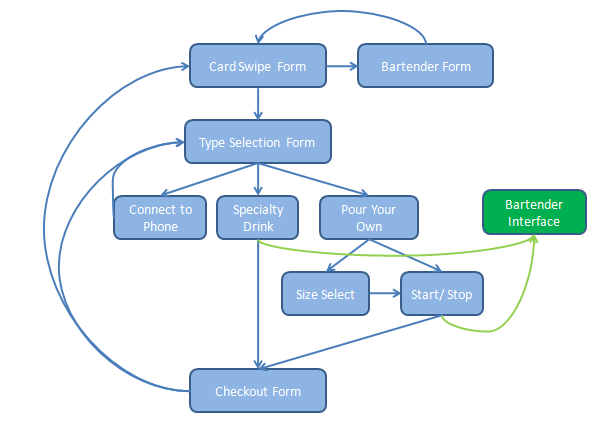


Figure 3. User Interface Flow Chart

-function of subsystem

The user interface allows the user to interact with the AutoBev system. It is displayed on a standard PC monitor. The user interacts with the interface using a keyboard and a mouse. The interface consists of nine different forms.

Software Flow:

Welcome Form:

The first form with which the user is presented is the ‘Welcome Form.’ This form instructs the user to swipe their credit card. Track one of the credit card is read using the MSR and is input into a hidden textbox located on the form. When the specified ‘end character’ is detected as an input into the text box, the form checks the contents of the text box. Two types of input cause the form to respond. The first is if the bartender swipes a special access card. The code of this card is stored within the user interface. If this code is detected, then the user interface brings up a form that only the bartender can access which allows the bartender to dispense a beverage from the AutoBev system without being charged. This functionality is useful for when the reservoir needs to be changed. The other type of input that causes the form to respond is if a valid credit card has been swiped. When the end character is detected, the contents of the text box are also compared against the standards for a valid credit card. Upon validation, the user interface proceeds to the ‘Type Selection Form.’ The track that is read is stored as the ‘cardNumber’ property in a new instance of the class ‘Person.’

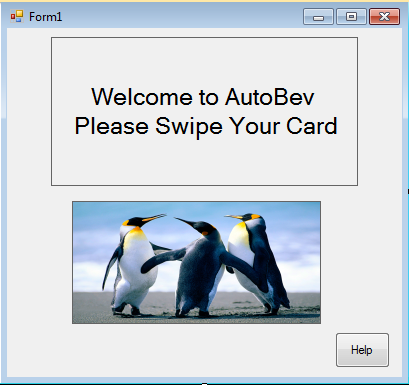


Figure 4. Welcome Form

Bartender Form:

When the Welcome form identifies that a bartender has swiped a special access card, the Bartender Form is opened. The bartender may then choose to pour a beverage. When this option is selected, the bartender is taken to the Start and Stop pouring form. However, the cost associated per ounce with this transaction is set to zero. Additionally, the limit that the bartender can pour is set to the maximum amount. Through the use of this form, it is possible for the bartender to refill the beverage reservoir with ease.

ADD PICTURE

Type Selection Form:

As soon as the type selection form is loaded, a request is sent to the bartender interface for an updated list of drinks and prices. This list is then stored in a byte array which is part of the Person class. The user is presented with three options on the Type Selection Form. The user can choose to set up a username and password to be used with the Android application, to order a specialty drink, or to pour their own beverage. By clicking the button corresponding to any of the aforementioned options will bring the user to a new form.

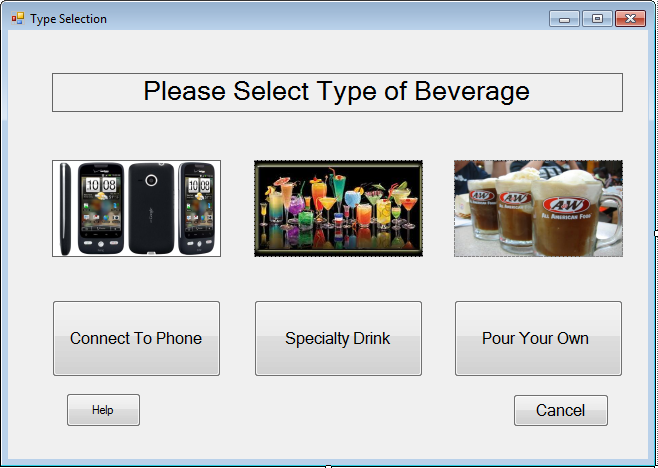


Figure 5. Type Selection Form

Username Form:

The Username form allows the customer to input a username and a password into a text box. When the user clicks the save button, the form verifies that the username consists of all letters and that the password is a four digit pin. The decision to create these constraints on the username and password came out of a desire to simplify the design. If the constraints are met, the track (stored in the current instance of the Person class), the username, and the password are sent to the bartender interface. Here, they are stored to be used as a means of identifying the user when they place an order from their Android. The user is then presented once again with the Type Selection form.

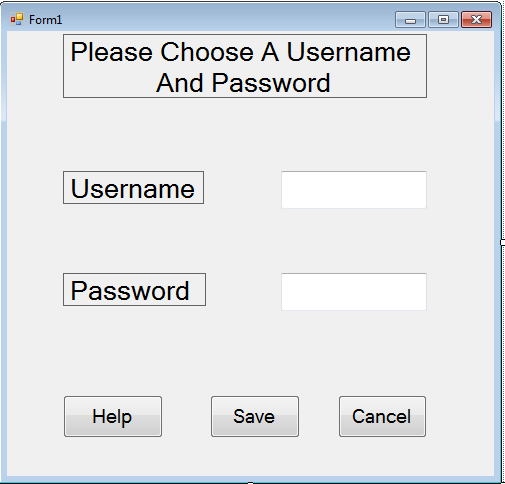


Figure 5. Username Form

Specialty Drink Form:

The Specialty Drink form allows the user to select a drink from a list of six. The price corresponding to the drink is displayed below the corresponding drink button. When a button is pressed, the drink order, along with the track is sent to the bartender interface. Here, the users total is updated. The bartender sends back the order number, as well as how many orders are in front of the current order. This information is displayed to the user.

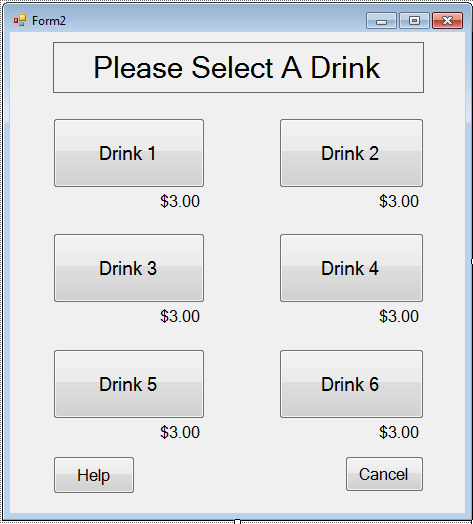


Figure 6. Mixed Drink Form

Beverage Dispensing Type Form:

The Beverage Dispensing Type form presents the user with the option of selecting a size of beverage to be poured, or of paying per ounce. When a user decides to pay by the ounce, the limit is set to three pitchers (or 180 ounces). Additionally, on this form, bit three of the PC to microcontroller status byte is set to indicate the size of the order.

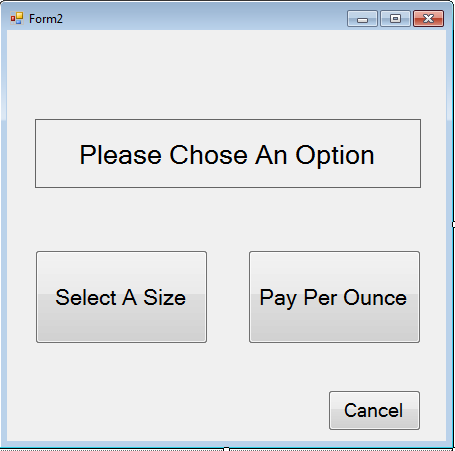


Figure 7. Beverage Pour Type Form

Beverage Size Select Form:

The Beverage Size Select form presents the user with three size options. A user may choose to dispense twelve, sixteen, or sixty ounces. The prices of the three sizes are displayed below their respective buttons. The fourth, fifth, or sixth bit of the PC to microcontroller status byte is set when the user selects to pour twelve, sixteen, or sixty ounces, respectively. After a size is selected, the status byte is is send over the UART to the microcontroller. The user interface waits to bring up the Start and Stop Pouring form until a status byte indicating that the volume limit has been successfully set is sent from the microcontroller to the PC. If an error status byte is sent back, the user is asked to select the size again.



Figure 7. Beverage Size Select Form

Start and Stop Pour Form:

The user is presented with three options on this form. The first option is to start pouring. When selected, the first and second bits of the PC to microcontroller status byte are set and the byte is sent to the microcontroller, which will open the solenoid valve and allow a beverage to be poured. The second button on the form is the stop pouring button. When selected, the second bit of the PC to microcontroller status byte is cleared and the byte is sent to the microcontroller, which will then close the solenoid valve and stop the flow of beverage. This form has a text label which is constantly updated to display to the user how much liquid has been poured. This form is in constant communication with the microcontroller which periodically updates the user interface with its status and the volume dispensed. Each time data is received from the microcontroller, the form checks the status byte. This byte indicates if a cup is still detected by the microcontroller and if the volume limit has been reached. If it has been determined that a cup is not present, the form instructs the user to place a cup under the dispenser and begin pouring again. In a similar manner, if the volume limit has been reached, the customer is notified and instructed to proceed to checkout. When the user finishes pouring, a status byte is sent from the PC to the microcontroller indicating that the current session is over. This reinitializes total volume poured to zero ounces and the microcontroller then beings waiting for further instruction from the user interface.

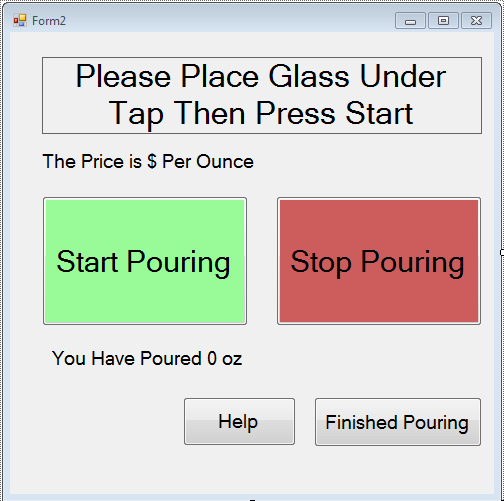


Figure 8. Start and Stop Pouring Form

Checkout Form:

The Checkout form presents the user with two options. The first is to checkout. If the user chooses to checkout, the user interface proceeds to the Welcome Form and waits for a new user. The ‘cardNumber’ property of the Person class that was created when the person first scanned into the system is cleared in order to prepare for a new customer. If the user chooses to order again, the user is taken to the Type Selection form. The track that was read when the person first scanned into the system is kept as the ‘cardNumber’ property for the current instance of the Person class that had previously been created. This is again used as a means to identify the customer and update the associated nightly total in the local database.

ADD IMAGE

-interfaces to other subsystems

-if there are communications protocols involved in subsystem, description and state transition diagrams should be included

The user interface communicates with both the microcontroller and the bartender interface.

A Universal Asynchronous Receiver/ Transmitter (UART) is used to communicate between the user interface and the microcontroller using the RS-232 protocol. A single status byte is sent from the PC to the microcontroller where the individual bits are checked to determined the action desired by the customer. The microcontroller responds with a single status byte and two bytes that are combined by the user interface to represent a 16 bit volume. The bits of the status byte sent from the microcontroller to the PC are individually checked to determine the state of operation of the microcontroller. The meaning of each bit in these bytes is further described in section X.X.

-subsystem testing, how this subsystem was tested to ensure functionality

The user interface was tested by stepping through each path that a customer may take while placing an order. Through this process, it was possible to verify the functionality of each command. Additionally, message boxes (which act like a pop-up message in a web browser) were placed in strategic locations to verify that information was being transmitted and stored properly. Furthermore, verification statements were output to a text file at strategic points throughout the interface. This text file was then viewed and essentially displayed the progression of the customer through the interface.

It was essential to use the debugger that is available in Visual Studio. During the testing of each separate form, break points were inserted before every function (or event callback which was triggered by any action performed on the form). The debugger was then used to ‘step through’ the function. During this process the value of each variable was verified to be correct at every point within the code.

In order to test the integration of the user interface with the microcontroller/ flow sensing hardware, it was verified that the volume specified by the user on the Beverage Size Selection form was that which was poured by the beverage dispensing system. For example, on the user interface, a twelve ounce beverage was selected. Twelve ounces was then measured manually and then allowed to flow through the beverage dispensing system. As this occurred, it was verified that the amount poured was updated on the Start and Stop Pouring form, and that the solenoid valve closed and prohibited further dispensing after twelve ounces had been poured. This was repeated for the other two sizes.

In an effort to make the C# code as error-proof s possible, try-catch statements were used extensively. They were especially important to use on forms that interface with other subsystems. For example, every time the bartender is contacted, a UDP connection must be established. However, if a bartender is not available, it will not be possible to establish a connection. Therefore, a try-catch statement was used to test for a possible connection. The same logic was used when deciding to use a try-catch statement when attempting to interface with the microcontroller through a serial port.

-why engineering decision were made (choice of processor, kind of RF interface, choice of programming language)

The user interface was implemented as a Windows Form Application using the Microsoft Visual Studio 2010 IDE. After investigating and comparing several languages supported by this environment, it was decided that the best way to achieve a professional-looking and easy-to-use interface was to program using Visual C#. The combination of Microsoft Visual Studio and Visual C# enabled access to many useful predefined libraries. These existing libraries made it relatively simple to communicate with the microcontroller, the bartender interface, and the magnetic stripe reader.

The decision to stylize the user interface as Windows Forms was based on the fact that forms are easy for users to navigate. During the drink ordering process, every time the user needs to make a selection, he or she is presented with a self-explanatory form. The interface then proceeds to the next form. The ability to separate each step of the drink ordering process with the creating of multiple forms greatly simplified the development of the user interface.

### Detailed operation Bartender Interface

The fundamental purpose of the bartender interface is to serve as a database of recently ordered specialty drinks. The software allows the bartender to read drink entries made by the customer at the user hub, and then delete the orders after they have been prepared. The display for the bartender is a drink list that is updated with additions from the User hub. It provides options for the bartender to scroll through the orders and delete selected drinks. The form for the bartender interface can be seen in figure X below and is called Form1 in the bartender program.

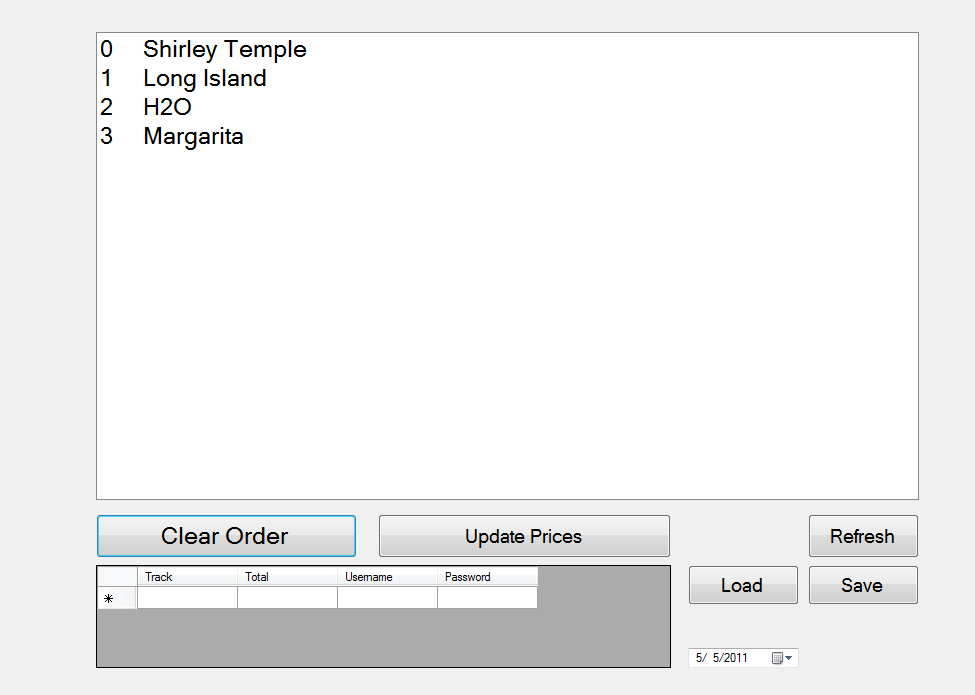


Figure X. Bartender Interface Drink List Form

The bartender needs to be able to interface with the customer. This takes place over two separate computers. A UDP Server/Client protocol was used for communication between the two computers, the User and Bartender hub, over a local area connection. A UDP protocol was chosen because two of the team members had experience with computer networks and because the amount of information being communicated is extremely small and only needs to be sent occasionally in packets, which is ideal for UDP. The protocol was written in C# so that it can seamlessly interface with the customer interface. For a more detailed description of the client/server interface refer to section X.X.

The bartender interface needed to able to store the drink orders and order numbers to different arrays. Two variables, *counter* and *i* were initialized to 0 at the beginning of the program and were incremented by 1 each time a new order was processed. The integer value of the variable *counter* was placed in a *count* array and was the number sent to the user as their order number after an order was placed. The variable *i* reflects the number of the last index.The purpose of storing the variable in a *count* array was to make it so that when a drink order was cleared the *count* array would shift all values in the *count* array after the *count* index that value is stored in up into the previous index. Once this was done, the variable *i* was decremented by one to keep track of how many values were actually being stored in the *count* array. This function occurrs any time the Clear Order button is pressed on the bartender interface. The purpose of this was so that the order numbers that were already served were no longer being stored and also for the correct order number to be written on the list next to the appropriate drink order. The value that is stored in *count[0]* is always the order number of the first drink on the list and so on. The clear button called on the function *DrinkQueue.Items.RemoveAt* with the input *DrinkQueue.SelectedIndex* when clicked. This function removes information at the selected index from the list. To allow for the user to know how many orders are ahead of them in the list, whenever their order is processed the program counts all items on the list using the function *DrinkQueue.Items.Count,* stores this value in a variable *numItemsInList,* and sends the value in this variable to the user as a string. The complete message sent to the user is in the form of the following string:

"Your Number is " *count[i]* ".

There are " *numItemsInList* " Orders in Front of You"

The bartender interface also allows for the bartender to update the drink choices and prices for the night. This is done by selecting the Update Prices button on the bartender interface. Liz detail this function here The form from which the bartender can update the drink choices and prices is called DrinkPrices in our program and can be seen in figure x below.

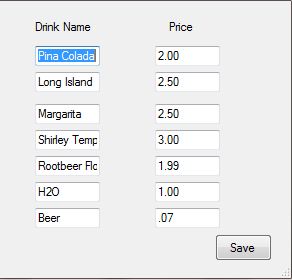


Figure X. Bartender Interface DrinkPrices Form

Liz add here stuff about storing and loading…etc…

Need Flow Chart of Interface

The bartender interface was tested by verifying that the drink selected on the user interface was properly displayed on the bartender interface along with its assigned drink order number. This process was repeated for each possible drink selection and for multiple orders to ensure that the server was updating the list correctly. We also needed to test that the clear button worked correctly and did this by selecting orders from various places in the list and making sure that it updated correctly. Most of this testing was done by simply checking that the program was running correctly. In order to debug the program when things were not appearing as expected on the list we had message boxes pop up with the current values of the different variables and arrays to make sure they were what we expected them to be. If they were not as expected we could have a better idea of what the program was doing based on the values given and reviewing what areas of the code would have caused them to have that value.

### Detailed operation of Microcontroller

The microcontroller functions to control the state of a solenoid valve, to detect when a cup is present at a limit switch, and to keep track of the number of pulses put out by a flow sensor and communicate a status dependent on these factors to the computer. Therefore, the AutoBev kiosk requires a microcontroller capable of serial communication with a computer, at least 4 I/O pins, with at least 2 interrupts (16 and 8 for expandability), capability of handling C programmed routine of about 2KB, and an adjustable baud rate of 9600, and ability to hand solder. For these reasons, the PIC18F4320 by Microchip was chosen.

The PIC18F4320 has an internal flash memory of 8KB, serial transmit and receive pins, and 34 I/O pins. The PIC18F4320 is programmed in simple C and is compatible with Sourceboost compiler and the MeLabs programmer which was available on the learning center computers. The group also has some past experience using a similar microcontroller which made it a good choice. The last requirement that the chip be hand soldered was fulfilled by the .8 pitch spacing between pins in the TQFP- 44 pin layout.

The PIC18F4320 can be powered using a 3.3 or 5 V supply, the AutoBev controller is powered by a regulated voltage of 5 V which was stepped down from 12-volt supply that also powers the solenoid coil driver circuit.

A schematic of the microcontroller can be seen below in Figure X.X.

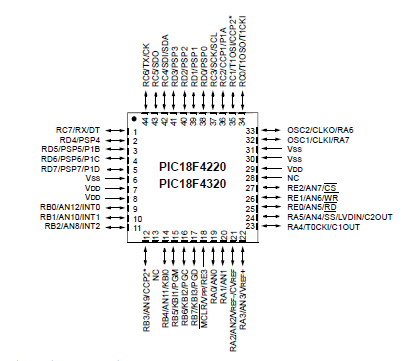


Figure - Pin Layout of the 44-Pin PIC18F4320

The microcontroller communicates with the user interface through UART serial communication. The program on the microcontroller is designed to wait until a status byte is sent from the user interface to begin a new drink order session. This status byte designates instructions which include the volume to be poured, and additionally if the user is requesting to start or stop pouring. The microcontroller responds to the user interface with a status byte and two bytes which together form a six bit number holding the volume that has been poured. The individual bit meanings can be seen in Tables X.X through X.X and are described in detail in section X.X.

-how subsystem was tested

-schematics and software need to be described in addition to schematic and code

-hardware: describe how circuit works (how amplifier circuit and filters are designed/ operate)

-

-software: describe overall flow of the code (interrupt driven, state diagram)

-if there are communications protocols involved in subsystem, description and state transition diagrams should be included

-subsystem testing, how this subsystem was tested to ensure functionality

### Detailed operation of Android application

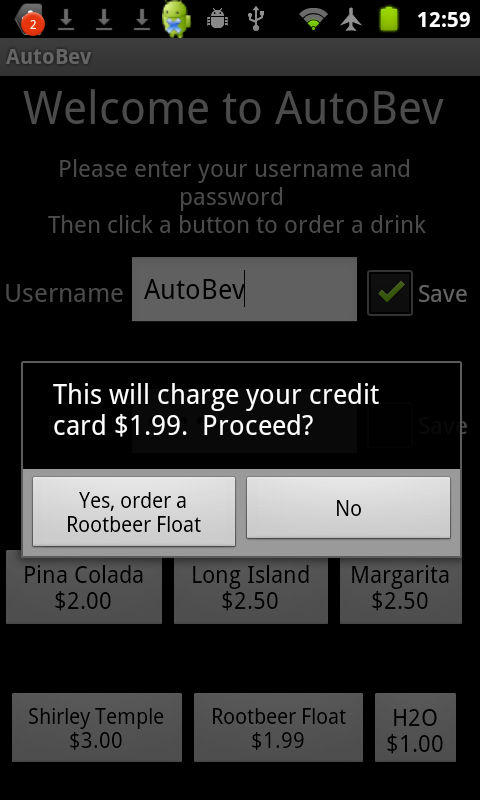
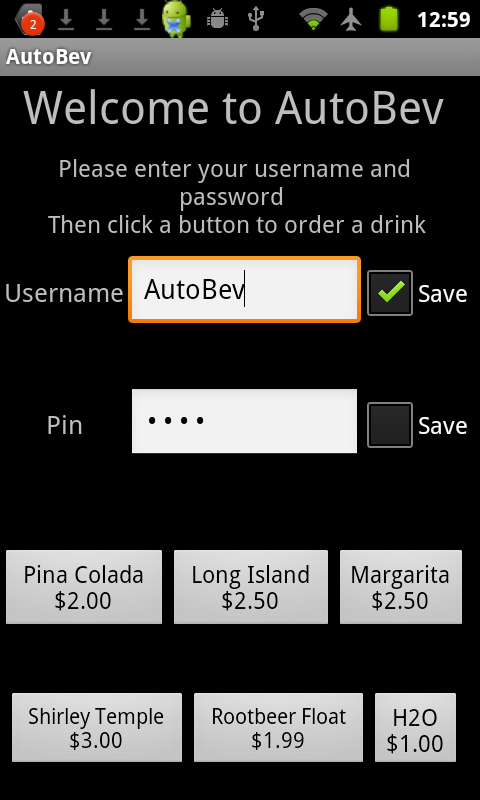
**-Schematic (hardware) or flow chart (software) for subsystem**

**-function of subsystem**

The Android application allows the user to interact with the AutoBev system with his or her smartphone. The app has all the functionality of the AutoBev system contained at the kiosk except for the ability of the customer to pour his or her own beer.

Before using the AutoBev Android application, the user must sign in to an AutoBev kiosk at the bar. At the kiosk they will click on Connect to Phone after swiping his or her credit card. The user will then be prompted to enter a username and password which, along with the credit card track, will be sent to the bartender and saved to the database.

When the user opens the app he or she is presented with a welcome screen that contains two textboxes and six buttons shown in the figure below. The user enters his or her username and password into the respective textboxes and then clicks a button corresponding to the drink they wish to order. A message is sent over the wireless network using to the bartender’s PC containing the username, password, phone number, IP address of the phone and the desired drink of the user.



The bartender’s PC will verify that the username provided exists in the current database and that the password matches the username. If either of those conditions occur, the bartender will immediately send a message back to the phone telling the user to try again.

If the username and password are correct, the order is stored in the bartender’s queue. A message is then sent to the app telling the user what drink number is his or her and the drink currently being served.

When the drink is filled by the bartender (i.e. the bartender clears it from his or her queue) the bartender’s PC will send a text message and/or email to the Android user alerting him or her that the drink is ready.

**-interfaces to other subsystems**

The Android application will only interface with the bartender PC. The interface will involve a two way communication using the User Datagram Protocol (UDP) to send messages over the wireless network. Messages will also sent to the Android app via Short Message Service (SMS) text messages.

The Android application will have a separate receiving thread that is constantly looking for packets from the bartender. When the bartender sends a message to the application, the thread checks what the message’s purpose is and acts accordingly.

When a user sends a drink order to the bartender, the first character of the message tells the bartender’s program that the message is from the Android and the bartender’s PC then extracts the information from the message to store a drink.

The SMS messaging works using the Simple Mail Transfer protocol (SMTP). The C# code uses an email account setup using Gmail. The code needs to know the phone number of the app, the service of the phone (i.e. Verizon, Sprint) and the email address and password of the sending account.

**-how subsystem was tested**

The Android application was tested by stepping through the program so that every possible path is covered to ensure there are no errors. Debugging was achieved by printing to a terminal to know where code was hanging and to print variables.

The Android application was tested and performed flawlessly on both the Android 2.2 and Android 2.3 platforms. The different platform simply required choosing a different target in the development software Eclipse.

**-why engineering decision were made (choice of processor, kind of RF interface, choice of programming language)**

The decision was made to develop an application on an Android smartphone as opposed to a different operating system such as an iPhone. This choice was made for a variety of reasons. A member of the AutoBev team had experience with Java, the online support for Android development, the open source “Linux-like” community and finally the Notre Dame Computer Science Department develops on Androids and was willing to lend a phone to the group for development purposes.

The choice to use the User Datagram Protocol (UDP) for the interface between the bartender PC and the Android phone because that type of protocol was used for the bartender and user interface already and was easily adapted for the phone – PC communication.

**-schematics and software need to be described in addition to schematic and code**

**-software: describe overall flow of the code (interrupt driven, state diagram)**

The Android application has two threads constantly running: the main UI thread and the UDP server thread.

The main thread takes input from the user via the phone’s touch screen. The thread is constantly waiting for user action and does not act until the user has clicked one of the six drink buttons. Once one of the drink buttons is clicked the code checks whether the username and password are of the right form. The username must contain only letters and the password must be four numbers. If the username and password are not correct then the program asks the user to try again.

If the username and password are of correct form, a popup form asks the user to verify that his or her credit card will be charged the cost of the drink. If he or she clicks yes then a message will be sent to the bartender containing the username, password, phone number, IP address of the phone and the desired drink of the user.

The bartender checks to see if the username and password are correct and then sends a message back to the phone stating that the drink order has been placed and gives the drink serving number the user is, along with the current drink being served.

The second thread is the UDP server thread that is constantly waiting for a message from the bartender. There are three types of messages which are distinguishable by the first character of the received message corresponding to: wrong password, username does not exist, and order placed successfully.

A while loop runs in the main thread until a message is a received which sets a Boolean flag to true. If a message is not received in a limited time (approximately five seconds) a popup box says that the bartender did not respond and to try again. This is to prevent the code from hanging on the receive if there is a communication breakdown.

**-if there are communications protocols involved in subsystem, description and state transition diagrams should be included**

The communication protocol is described in great detail in the Bartender to User Interface section.

The message sent from the Android phone to the bartender is of the form: "&" + Phone Number + "#" + IP Address + "#" + username + "#" + pin + "#" + drink order + "#".

The message received by the Android phone from the bartender is of the form: "X#" + count[i] + "#" + count[0] + "#", AndroidIP. Where X tells the phone what type of message is received, count[i] is the user’s order number, count[0] is the order currently served and AndroidIP is the phone’s IP Address.

### Detailed operation of Beverage Dispensing and Sensing

After a drink has been ordered, this feature allows for drinks that do not need to be made by a bartender to be instantly available. The computer that serves as the user interface communicates with the microcontroller via a USB interface, which controls all functions associated with the keg. Customers are able to place their cup under a tap and pour their selected beverage once they have completed all necessary steps on the user interface. The mechanical system to control flow is created to connect to a keg of liquid. In order to keep a constant pressure in the keg, a gas tank with a constant pressure valve is used. A spigot, keg adapter, and tubing are also included in the keg setup. The solenoid valve is attached to the part of the tube close to the tap and is actuated by a digital output from the microcontroller. The microcontroller signals the solenoid to open if a customer has placed their order and is pouring their own drink. A flow sensor is also placed in the flow path to send a signal to the microcontroller which specifies how much drink has been poured for pricing purposes and to monitor flow in case the customer opts to have a specific amount of drink poured rather than to pay by ounce. We also have a limit switch by the tap to indicate that a cup has been placed there in the chance that someone is trying to pour a drink without a cup. Additionally, our current system of flow sensor, limit switch, solenoid, and microcontroller board are attached to the output of a Danby 5.8 Cu. Ft. Capacity Keg Cooler (Model # DKC645BLS). The keg cooler has a manual spigot that we are using as an emergency stop that is independent of the software in case anything goes wrong and we want to stop beverage flow. Upon completion of serving a drink, the microcontroller sends information back to the user interface so that the tab can be updated.

Note that our system is designed to be expandable and the microcontroller has circuitry to handle up to 2 each of the solenoid, flow sensor, and limit switch and has available I/O ports for up to a total of 4.. The actual system demonstration only utilized one of the sets.

The SwissFlow SF800 flow sensor was selected as the flow sensor for our system. In reviewing different sensors online, this sensor was used by others in measuring beer poured from a keg and was said to have accurate results . It also is made so that communication with a microcontroller is feasible via a pulsed output. Furthermore, the people of SwissFlow were willing to send the group free samples, which allowed for us to be able to have extra sensors in case anything went wrong during testing. The SwissFlow SF800 flow sensor is supplied by a 12V source and requires a resistive network as seen in figure x below. The output of the sensor is a pulsed output that generates 5600 pulses/liter. The flow sensor allows the microcontroller to know how much liquid has been poured. By counting the pulses our program is able to calculate the volume of the liquid that is passing through the tube.

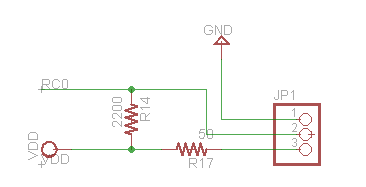
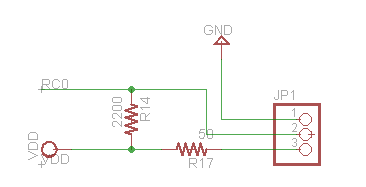
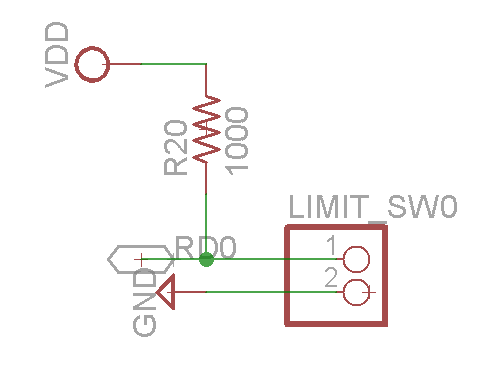


Figure x. Flow Sensor Circuit



A limit switch acts as a safety for the system such that a beverage will not be dispensed unless there is a cup present beneath the spigot. A normally closed switch is connected to input pin RD , and holds the pin at VDD until the switch is pushed where it sets the pin low to GND. The circuit can be seen in Figure.x below.



Our solenoid of choice was the ECT 12 V solenoid (US5311162). The solenoid takes in digital input, which is necessary for communication with the microcontroller. Furthermore, the solenoid was provided to the group for free by Professor Mike Schafer of the University of Notre Dame. The solenoid serves as the main action unit as it will open or close to allow or stop the beverage from flowing. The ECT 12 V solenoid (US5311162) is a normally closed valve that will open when 12 V is applied to the coil. The internal resistance of the solenoid coil is about 23 ohms, therefore to switch open the solenoid requires about .53 amps. A power MOSFET, NDS355, was chosen appropriately for the coil driving circuit, because it can handle up to 1.7 amps and up to 30 V, and can be actuated by a 5V output pin connected to the ground through a 500 ohm resistor. This digital output has a corresponding indicator LED for troubleshooting so that the owner can check the status of the solenoid. The driving circuit can be seen in figure x below.

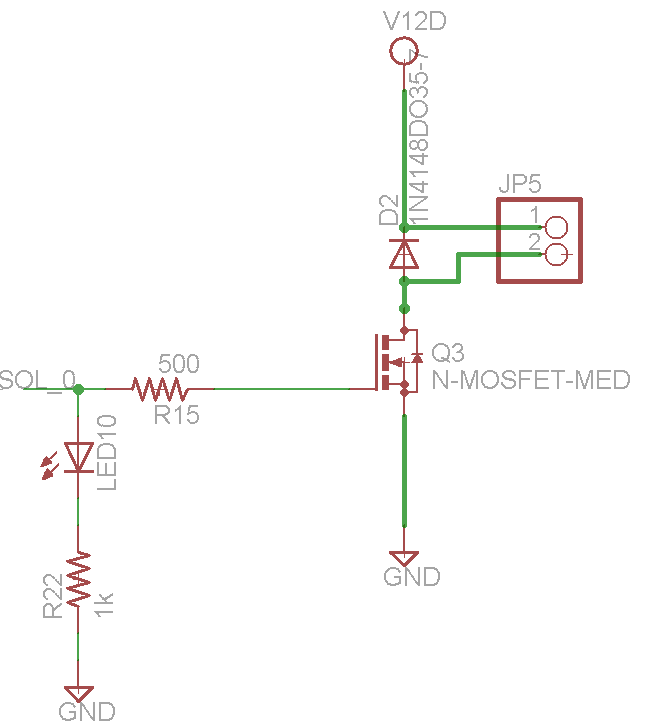


Figure 9. Solenoid Circuit UPDATE

The schematic of the overall board design for the beverage dispensing and sensing system can be seen below.

Figure x.

-MARK hardware testing, how the hardware was tested to ensure functionality

LIZ describe the Microcontrolller software here

-software testing, how the program was tested to ensure functionality

## Interfaces and Sensors:

### PC To Microcontroller Interface:

The PC to Microcontroller interface functions to connect the user interface to the microcontroller. A Universal Asynchronous Receiver/ Transmitter (UART) is used to transfer bytes between the two subsystems in accordance with the RS-232 standard. In order to convert the asynchronous serial messages sent by the microcontroller into standardized USB signals (recognized at the COM port of the PC), an intermediary is needed. The device used to do this is the FT-232-RI from FTDI. Professor Mike Schafer of the University of Notre Dame designed the circuit governing the operation of this device. The circuit was used with his permission and can be seen in the following figure.

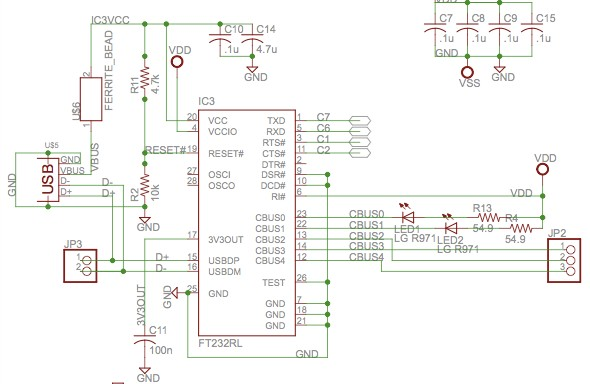


Figure X.X FT-232-RL Schematic

The FT232 is powered on pin 20 VCC by the USB connection, and the I/O pins are set to the voltage VDD of the voltage regulator output and connected to pin 4. Both of these values are 5V. The USB signal input/output is on pins 15 and 16 and the converted transmit and receive signals are on pins 1 and 5 respectively and go to the microcontroller pins RC7 and RC6. The state of these pins can be seen on CBUS0 and 1 and indicated on the board by a green (transmit) and yellow (receive) LED. Pins 3 and 11 are connected to the microcontroller pins RC1 and RC2 and held low to enable proper asynchronous communication.

The user interface communicates with the microcontroller using a single byte. This seven bits of this status byte indicate the status of the PC, the desired beverage size, if the user wants to start or stop pouring, and if the user has finished pouring.

The organization of the PC to microcontroller status byte can be seen in table X below.

**PC**  **Microcontroller**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit 0** | **Bit 1** | **Bit 2** | **Bit 3** | **Bit 4** | **Bit 5** | **Bit 6** | **Bit 7** |
| Success/Error | Message  Type | Start or  Stop | Order  Type | Size 1 | Size 2 | Size 3 | Filler |
| 1 = success  0 = error  (resend) | 1 =  start/stop  0 = new order | 1 = start  0 = stop | 1 = preset size  0 = pay per oz. | 1 = 12 ounce  0 = no order | 1 = 16 ounce  0 = no order | 1 = 60 ounce  0 = no order | Reset Order |

Table X. PC to Microcontroller Status Byte

As can be seen in table X above, the first bit of the status byte allows for indication of an error state by the PC. When this bit is zero, the microcontroller takes no action in response to the byte. It is assumed that an incorrect byte has been sent. Bit one of the status byte indicates the type of message that is being sent. If bit one is cleared, it is indicative that a new order is being placed. When this occurs, the microcontroller must check bits 3 through 6 to see to what the volume limit must be set. Only one of these bits is set in each status byte. If none of bits three through six are set when bit one is cleared, the microcontroller issues an error message to the PC and the user is asked to choose a size again. After the size has been set, the user interface sets bit one to indicate that the instruction is either to start or stop pouring a beverage. The microcontroller continues to listen to PC status bytes to detect when the user wants to stop pouring (bit 2 cleared). Bit seven is used to indicate when a user has finished pouring. When this bit is set, the variables in the microcontroller are reinitialized to their starting value and the microcontroller begins waiting for a new order from the PC.

The Microcontroller sends three bytes back to the user interface. The first byte contains the status of the beverage dispensing system, and the second two bytes hold a sixteen bit number that corresponds to the number of pulses that have been detected. The organization of these three bits can be seen in table X.X through table X.X below.

**Microcontroller**  **PC**

**Byte 1 (Status):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit 0** | **Bit 1** | **Bit 2** | **Bit 3-7** |
| Success/Error | Status | Cup Sensor | Filler |
| 1 = success  0 = error (resend) | 1 = still pouring  0 = done pouring | 1 = cup present  0 = no cup | DC |

Table X.X Status Byte From Microcontroller to PC

**Byte 2 (Last 8 bits of 16-bit value of volume):**

|  |
| --- |
| **Bit 8-15** |
| Bits 8-15 of Volume Poured  Value (pulses) |

Table X.X Lower Eight Bits of Volume (in terms of pulses)

**Byte 3 (First 8 bits of 16-bit value of volume):**

|  |
| --- |
| **Bit 0-7** |
| Bits 0-7 of Volume Poured  Value (pulses) |

Table X.X Upper Eight Bits of Volume (in terms of pulses)

Table X.X through X.X show the organization of the bits in the three bytes that are sent from the microcontroller to the PC. The first byte represents the status of the microcontroller. When bit zero is cleared, this indicates that the microcontroller has encountered an error. When bit zero is set, the microcontroller is successfully functioning. Bit one is set when the microcontroller is in a state of pouring. This occurs when a ‘start pouring’ instruction has been received from the user and a cup has been detected by the limit switch. Only under these two conditions will bit one be set. Bit two is used to indicate when a cup is present. If the limit switch detects that there is no cup, both bit zero and bit two are cleared and the status is sent to the user interface which instructs the user to place their cup and press start again.

The second byte contains the upper eight bits of the amount of volume that has been dispensed. The third byte contains the lower eight bits of the amount of volume that has been dispensed. The volume is in units of flow sensor pulses. Every fifteen pulses (0.091 ounces) the microcontroller reports its status and the amount of volume dispensed back to the user interface. The microcontroller also sends these three bytes of data any time a missing cup is detected or when the volume limit has been reached.

-how was the interface tested?

The interface was tested by once again using the debugger in Visual Studio. A breakpoint was placed on the function that executes when asynchronous data is received through the serial port. When the microcontroller sends the three status bytes, this function is called. It is then possible to open the buffer that holds the received data and check the individual bits of the status. This was done multiple times to ensure functionality. Additionally, pop-up message boxes were inserted into the user code that displayed the Pc to microcontroller status byte. These bytes were verified for every possible message that could be sent in either direction.

-design decisions

The number of status bytes used to send information from the user interface to microcontroller and from the microcontroller to the user interface was based on the desire to keep the protocol as simple as possible. In both cases, the minimal amount of bytes that could be used to communicate the necessary data was used.

### Bartender To User Interface:

The user to bartender interface functions to convey drink orders, ‘pour your own’ charges, and new usernames and passwords from the user interface to the bartender interface.

As detailed in section X.X a UDP Server/Client protocol was chosen for communication between the two computers, the User and Bartender hub, over a local area connection. This enables two way communication between the bartender and user. This could also be done using a TCP protocol, the amount of information being communicated is extremely small and will only need to be sent occasionally in packets, which is ideal for UDP. UDP applications use [datagram sockets](http://en.wikipedia.org/wiki/Datagram_socket) to establish host-to-host communications. An application binds a socket to its endpoint of data transmission, which is a combination of an [IP address](http://en.wikipedia.org/wiki/IP_address) and a service port.The basic flow behind the UDP Server/Client can be seen in figure XX below. The protocol was written in C# so that it could easily be integrated into the user and bartender programs.

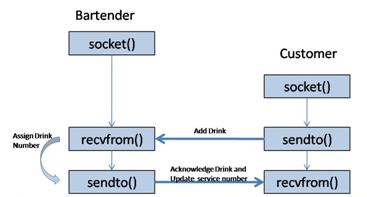


Figure XX. Flow of Client/Server

The client UDP protocol program is embedded within the UI program as a separate function on the MixedDrinksForm. Upon selecting a mixed drink order on the form, the client function is called with the selected drink as an input. The function sends a string message to the server that included the selected drink. The client UDP protocol program that was embedded into the Android application is similar and was discussed in section X.X.

The server UDP protocol program is embedded within the bartender program. It has the capability to open multiple communication threads in order to allow the owner to add multiple machines to the system. This was made possible by the use of multithreading in the program. Multithreading also made it feasible for the application to remain responsive to input, while still being able to add and clear orders from the drink list. In a single-threaded program, if the main executions thread blocks on a long running task, the entire application can appear to freeze. The initial program was experiencing this problem and, as a result, was not printing orders beyond the first one placed to the drink list. It was still processing the orders, but was not able to add them to the list and make them visual to the bartender. By moving such a long running task as the ability to continuously take in drink orders to a worker thread that runs concurrently with the main execution thread, it is possible for the application to remain responsive to user input while executing tasks in the background. Once multithreading was integrated into the server program the drink list which was called *DrinkQueue* in the program was able to add each new drink order processed to the list.

There are four possible types of messages that the user may send to the bartender. The first type of message is a specialty drink order. The second type of message is a ‘pour your own beverage’ order. The third type of message is the setting up of a username and password by a customer to be later used with the Android Smartphone application. The last type of message is a request by the user for an updated list of drinks and prices. The bartender can also receive a string message from an Android Smartphone through the UDP connection. In all five cases, the message that is sent from the user to the bartender is a string. This ASCII code for the string is encoded in bytes and sent via the UDP connection to the bartender where it is decoded. The bartender responds to the message based on the first character that is sent. Each order string is further subdivided into several other strings separated by a pound sign. Based on the first character in the string, the bartender is able to decode and use the information in the substrings. Table X.X below shows the organization of these substrings.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Start Character | String 1 | String 2 | String 3 | String 4 | String 5 | Order Type |
| # | Drink Name | Track Number | Cost | Null | Null | Kiosk Order |
| @ | Track Number | Pulse Count | Null | Null | Null | Pour Your Own |
| $ | Track Number | Username | Password | Null | Null | Android App Initialization |
| & | Phone Number | IP Address | Username | Pin | Drink | Phone Order |
| ! | Null | Null | Null | Null | Null | Price Request |

Table X.X String Format for Orders Between Bartender and User

Table X.X above shows the format used for the strings sent from the user to the bartender. Each string is separated by a pound sign. Using the pound sign as a delimiter allows the bartender to extract the necessary information from each user request. Depending on the type of request, the bartender responds with a message in string format. It also updates the local database as needed.

The above protocol was decided upon to allow for multiple types of orders to be sent to, and processed by, the bartender interface. Without the use of a start character and a delimiter character, it would not have been possible to send five different types of messages to the bartender. In the above table, a null string simply means that no string is sent. Thus, for a Price Request order type, a single character is sent from the user interface to the bartender interface. This was meant to keep the data transmitted to a minimum.

The system was tested by sending each type of order from the user interface. The entirety of the message was first displayed to make sure that the whole string was sent successfully. Next, the individual substrings were displayed on a list in the bartender interface. Finally, pop-up message boxes were added to the user interface to show the string message that the bartender sends back in response to an order. These messages were unique to the type of order request. Thus, it was possible to detect if two way communication was successful.

--flow sensor, limit switch, emergency stop

To the extent that the interfaces between subsystems or sensors in the system need further explanation, do so. (Parts of this may be covered in the subsystem descriptions.)

# System Integration Testing

## Describe how the integrated set of subsystems was tested.

## Show how the testing demonstrates that the overall system meets the design requirements

# Users Manual/Installation manual

Thank you for purchasing the AutoBev System. We promise that this system will dramatically improve service at your establishment. If you recently purchased an AutoBev System you have everything you need to install and integrate one AutoBev kiosk into your current bar operations. If business is booming and you need to expand, please order additional kiosks at <http://seniordesign.ee.nd.edu/2011/Design%20Teams/AutoBev/AutoBev_Base.html>.

The original AutoBev System includes:

* One Bartender system including:
  + Computer and monitor (with all necessary connectors)
  + Mouse and keyboard
* One AutoBev Kiosk including:
  + Computer and monitor (with all necessary connectors)
  + Mouse (if monitor is not a touch screen)
  + Credit Card Reader
  + Bartender Access Card
  + Refrigeration Unit
  + Tubing and Keg Connections
  + CO2 tank
  + Control Hardware

With the AutoBev system you will reap many benefits including:

* Shorter and fairer wait times
* Increased efficiency
* Easy expansion
* Increased order accuracy
* Ability to received orders from Smartphones
* Decreased employee workload

## How to install your product

Installing the AutoBev System is as easy as booting up two computers and connecting a keg. Follow these basic steps in order to have your system fully integrated in no time at all.

**Bartender System**

* + - 1. Find a place behind the bar for the Bartender System with access to a wall outlet. Space reserved should be large enough to fit a computer box, a monitor, a keyboard, and a mouse.
      2. Plug in the computer labeled “Bartender” into the wall outlet.
      3. Plug in the keyboard, mouse and monitor into the computer.
      4. Turn on the computer and monitor.
      5. On the desktop double click on the executable called “Bartender Interface”.
      6. The first form encountered is a list of drinks and prices. These may be updated in real time as often at the bartender desires (set up initially here, and then can be changed by clicking the ‘Update Prices’ button on the main bartender form.

**AutoBev Kiosk**

1. Find a place in your bar near a wall outlet from which you would like your customers to order. It is recommended for this location to be in a low-traffic area of the bar that is easily accessible to customers.
2. Plug in the main power into the wall outlet.
3. Plug in the mouse (if not touch screen) and monitor into the computer box.
4. Turn on the computer and monitor.
5. On the desktop double click on the executable called “User Interface”. Customers may then begin a session by swiping their credit card.
6. Connect the CO2 tank and keg according to Dispensing hardware instructions.

**Dispensing hardware (how to replace a keg)**

1. Attach 5/8” hose to pressure regulator and secure with the red pinch clamp
2. Attach C02 tank to the pressure regulator via the hex nut connector, make sure the regulator lever is perpendicular to the hose connection.
3. Run hose through the hole on the back of the refrigerator unit, and place CO2 tank in the metal bracket.
4. Attach the 5/8” to the keg connector and clamp to seal.
5. Connect 3/8” hose from the spigot to the Solenoid / Flow Sensor unit with the arrow on the flow sensor pointing in the direction of flow.
6. Connect Solenoid/Flow Sensor to the keg connector and clamp to seal.
7. Push and turn keg connector to the keg output and push down lever to seal.
8. Open CO2 tank, and move regulator lever to the downward position(in line with hose) and adjust regulator to 12psi.

**What you need to Provide:**

1. Local wireless network that both the AutoBev Kiosk and Bartender Interface are connected to.
2. A keg for the kiosk
3. Lots of Thirsty Customers

## How to use your product

Since the AutoBev System comes completely configured and ready to use, if you have followed the above installation steps you are ready to go!

Using the AutoBev system is very intuitive. Customers order drinks at the AutoBev Kiosks and with their Smartphones. When orders come in, the Bartender System will display the order in the order in the order in which they were requested. which orders should be made in what order.

As orders are filled, the bartender will see a list of drinks that need to be made. After the bartender makes a drink he or she will click on the respective drink and hit “clear”. The customer will be alerted either via text message or by a screen displaying the order number currently being served.

At the end of the night the bartender will press the “Save” button in order to send the transaction information to the respective financial institutions to charge the credit cards stored in the database. The bartender may also view transaction data from previous dates by clicking “Load” on the Bartender Interface.

The AutoBev System is an isolated system so that it does not interfere with your current data handling and transaction operations. If you desire to have these integrated you must call the AutoBev support team and they will assist you in fully integrating AutoBev with your current system.

## How the user can tell if the product is working

The AutoBev System error checks itself so that when a piece of hardware or software malfunctions, an error popup box will tell the user exactly what went wrong. Many times a hard reset is all that is needed for the system to start working properly.

The following are potential errors and possible solutions: Liz can you do this?”

On the Welcome Form of the User Interface:

When the user interface is opened, the application tries to connect with a server application. If a server is detected, the user is allowed to swipe their card and begin the drink ordering process. However, if a server is not detected, and error message that reads “Server is currently down, please alert a bartender” appears. When a bartender sees this message, the first thing that should be done, is to press the ‘Retry Connection’ button in the lower right hand corner. This button runs the function that searches for an active server. If a server is found, the proper welcome form will be displayed. If not, the problem is most likely that the server application is not running on another computer within the bar. The bartender should then go to the location of the server computer and make sure it is running. It is probable that one of these fixes will solve the problem.

On the Type Selection Form:

When the Type Selection Form is opened, the user is presented with three options. One of these options is to ‘Pour Your Own” Beverage. In order to do so, the user interface must be connected to the device running the beverage dispensing and flow sensing software. Therefore, before the form allowing a user to choose a size of beverage to dispense is allowed to opened, the application checks for a valid connection. When this does not exist, the Type Selection Form changes appearance and the error message “Sorry for the Inconvenience. Beverage Dispensing Unavailable. Please Alert a Bartender” appears. When this happens, the user should immediately request for the help of a bartender. The bartender has a few options. The first is to press the “Retry” button located in the lower left hand corner. This will recheck for a steady connection between the beverage dispensing and flow sensing hardware. If one is found, the user will be allowed to proceed to a form where he or she is allowed to choose the type of dispensing (either pre-selected size, or pay per ounce) for the order. If a connection is not found, the error message will remain on the form. When this happens, the bartender should ensure that the serial port (connecting the dispensing hardware is connected). If it is, the bartender should remove the USB connection from the computer and then plug it back in. If neither of these fixes works, the bartender should check to ensure that the hardware is being provided with power, by making sure the regulator is plugged into the power strip, and to the power port on the board. If there is no power, the device needs to be plugged in. It is more than likely that after these three things are checked, the problem will be solved.

On the Checkout Form:

On this form, the user will receive one of two messages. If the user has ordered a specialty drink, the user is informed of their order number, how many drinks are to be made before their own, and how much they have been charged. When these three things appear on the form, an order was placed successfully. The other type of message occurs when the user has poured their own beverage. In this case the user will be informed of the amount charged to their card for that purchase. When these messages occur, the user can be confident that their order was completed.

On the Android Application:

When a user places an order from a Android Smartphone, he or she is required to provide the username and password that were established at an AutoBev kiosk earlier in the night. When the order is placed, the username and password, along with the drink order, are sent to the bartender interface where it is first verified that the username exists. The password associated with that username is then checked against that which was sent from the phone. When both of these two strings are identical, the drink order is placed. However, if the username is not found within the local database, the message “Username Does Not Exist” is sent to the phone and appears in a pop-up message box on the Android Smartphone. If the password associated with the sent username does not match that which is stored in the database, the message, “Password incorrect for existing username” appears in a pop-up box on the Android Smartphone. If either of these two messages occur, the user should re-enter their information and try ordering again. If this still does not work, the user may swipe the card that was used to set up the username and password at an AutoBev kiosk and proceed to the “Username” form. Here, the user may try to set up a new username and password. When the user clicks the “Save” button, a request to set up a new username and password is sent to the bartender. If it is determined that the user’s card number is already associated with a username, the user is notified of the username and password in the form of a pop-up box that appears on the username form. It is important to note, that each night the user will be required to set up a new username and password at the kiosk. This is done for security reasons. It also makes it possible for the user to associate various night’s transactions with different credit cards.

Overall, it is fairly intuitive to know if the AutoBev System is working correctly. When a drink order is placed from either an AutoBev kiosk or from an Android application, the drink appears in a queue on the bartender interface and a confirmation is send back to the user informing them that a successful order has been placed. Otherwise, an error message is sent to the user. When the user is sent an error message, the type of error that has occurred is sent as the message so the user knows exactly what has gone wrong. The bartender can use the error message to provide the user with assistance.

How the user can troubleshoot the product

The user can troubleshoot the product using error messages that appear when a failure occurs. These error messages are often self explanatory such as “No Server Available,” “Beverage Dispensing Unavailable,” and “Username/Password” incorrect. These messages are associated with specific areas of the application and narrow down the possible sources of error. When an error message associated with a specialty drink request is encountered, the connection between the bartender and user should be checked and then re-established. When an error message associated with a “pour your own” beverage request is encountered, the hardware connections between the user interface and the beverage sensing and dispensing hardware should be checked to ensure that they are stable.

AutoBev technical support is available during normal business hours to field any questions and direct the user through the troubleshooting process.

# To-Market Design Changes

The following list of changes would be made before the AutoBev System was taken to the commercial market. Some features are required before deployment while others are desirable but not necessary.

1. Change: Android application to be able to update the drink list and drink prices.
   1. Feasible: 100% feasible. Some time was spent trying to implement this feature with some success; however, the feature was not completely finished to be shown in the prototype.
   2. Importance: This feature is absolutely necessary for the customer to know which drinks are being offered at which price. It is also necessary for the bartender to know the desired order.
2. Change: Touch screen for AutoBev kiosk.
   1. Feasible: 100% feasible. The software will work with a touch screen because touch screens emulate clicks of a mouse.
   2. Importance: This feature is desired so there is no mouse at the kiosk; however, it is not vital for the overall functionality of the system.
3. Change: Integrate with the current transaction tracking system in the establishment.
   1. Feasible: This is feasible; however, the feature will require technical support.
   2. Importance: This feature is desired so the bar has all the transactions on one system; however, it is not vital for the overall functionality of the system because the AutoBev System may run in complete isolation.
4. Change: Charge credit cards.
   1. Feasible: 100% feasible. Will require technical support to interface the AutoBev System with the financial transactions.
   2. Importance: This feature is absolutely necessary for the system to function as it is intended.
5. Change: Cognitive testing. Must be able to determine if the customer is sober enough to have another drink.
   1. Feasible: Not very feasible. The only way to truly tell if someone has consumed enough alcohol is with human inspection. The AutoBev System must remain in view of the bartender so the bartender can use his or her own discretion.
   2. Importance: This feature would be more of entertainment than a necessary function. Not very important.
6. Change: AutoBev kiosks and Android application to be able to determine IP address of Bartender automatically.
   1. Feasible: Extremely feasible. Several networking schemes exist that can accomplish this.
   2. Importance: This feature is desired to reduce the complexity of installation; however, it is not vital for the overall functionality of the system.

# Conclusions

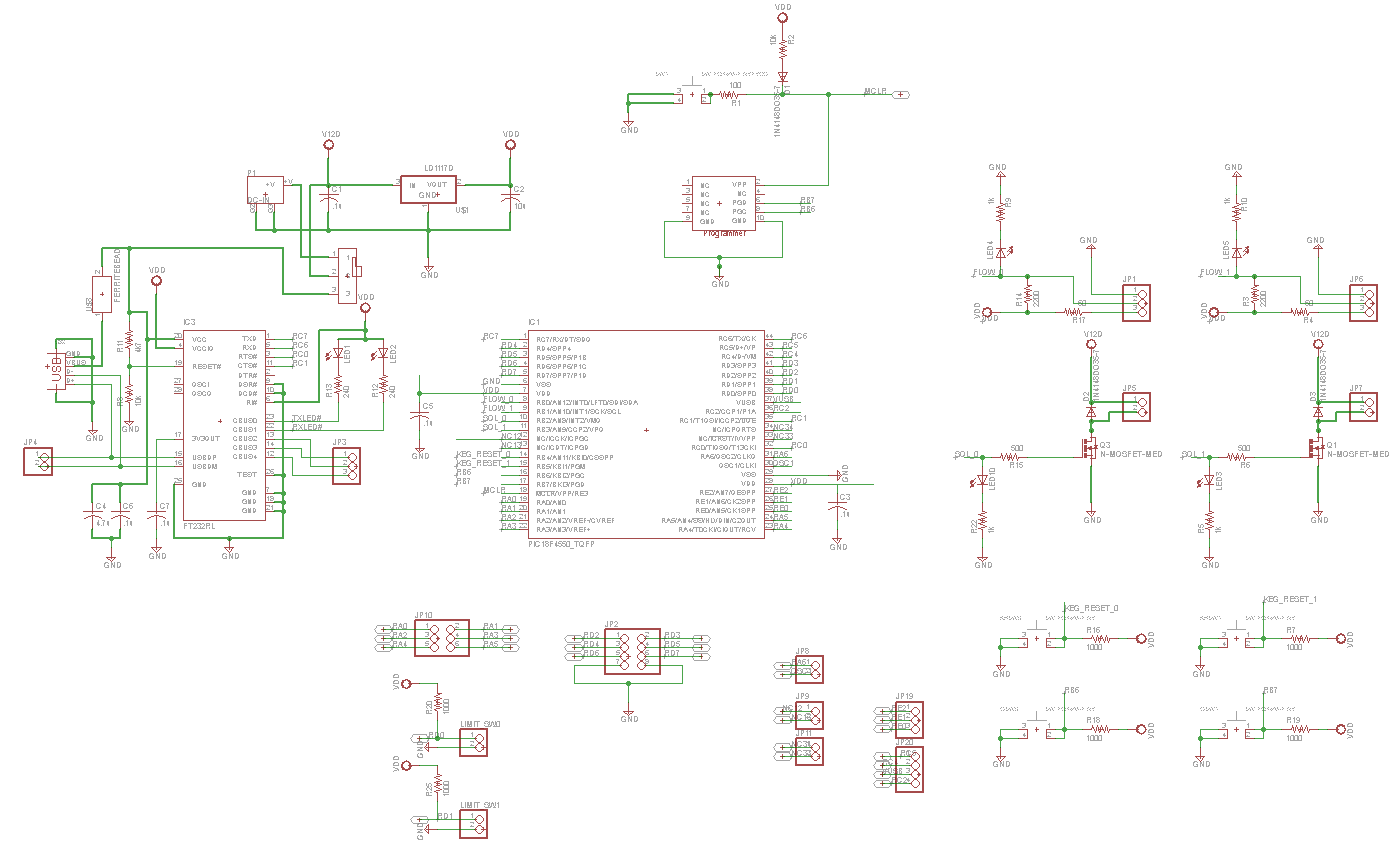
The following is from our LLD – basically we just need to reword the introduction from up top

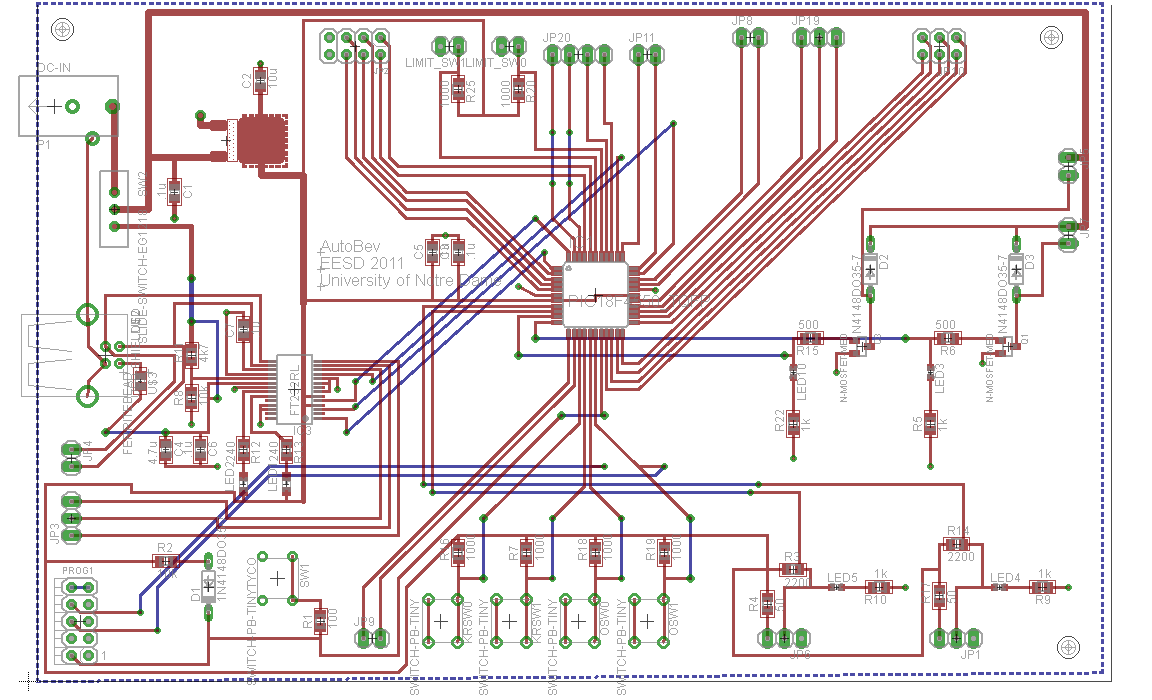
This project addresses a problem that limits the revenue of service establishments such as bars and restaurants. The proposed solution will eliminate the need for a server to deal with the time consuming payment process. We acknowledge that throughout the completion of this project, many obstacles will be encountered. The implementation of software subsystems that must communicate with one another will require extensive research and a good deal of learning. The hardware portion is also an extensive portion of the project that will take time and consideration.

If our group succeeds, we expect that our project will help to increase the efficiency of drinking establishments through the automation of the ordering and payment process. The AutoBev system is a marketable design that can be easily integrated into any service establishment.

# Appendices

Relevant parts or component data sheets (do NOT include the data sheets for the microcontroller or other huge files but give good links to where they may be found.)

9.1.1 Complete hardware schematics  
9.1.2Complete Board Layout



9.2 Complete Software listings

9.2.1 uC

9.2.2 C#

**9.2.3 Android Application Code**

**9.2.3.A AutoBev.java**

// AutoBev.java

// AutoBev kiosk emulated on smartphone

// ND EE Senior Design May 2011

// Alex Macomber, Liz Clark, Lori Garcia, Mark Pomerenke

package autoBev.android;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.text.DecimalFormat;

import java.util.Enumeration;

import java.net.NetworkInterface;

import android.app.Activity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.CheckBox;

import android.widget.EditText;

import android.telephony.TelephonyManager;

import android.app.AlertDialog;

import android.app.AlertDialog.Builder;

import android.content.Context;

import android.content.DialogInterface;

import android.content.SharedPreferences;

public class AutoBev extends Activity {

Thread UDPserverThread ;

// Preference file used for saving variables

public static final String PREFS\_NAME = "MyPrefsFile";

// Define buttons,checkboxes and textboxes so they can be changed later

public Button button2,button3,button4,button01,button02,button03;

CheckBox checkBox1,checkBox2;

private EditText username, pin;

// Client/server vars

DatagramSocket socket;

DatagramPacket packet;

InetAddress address;

// global vars

boolean receive = false;

String[] rcvMessage = {"0"};

// Define prices and drinks

double[] prices = new double[] {2.00,2.50,2.50,3.00,1.99,1.00};

String[] drinks = new String[] {"Pina Colada", "Long Island", "Margarita", "Shirley Temple", "Rootbeer Float", "H2O"};

// Set decimal format so prices show up as $X.XX

DecimalFormat priceFormatter = new DecimalFormat("$#0.00");

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

// Set buttons to show drink name and price

button2 = (Button) this.findViewById(R.id.button2);button3 = (Button) this.findViewById(R.id.button3); button4 = (Button) this.findViewById(R.id.button4);

button01 = (Button) this.findViewById(R.id.Button01); button02 = (Button) this.findViewById(R.id.Button02); button03 = (Button) this.findViewById(R.id.Button03);

button2.setText(drinks[0]+"\n"+priceFormatter.format(prices[0])); button3.setText(drinks[1]+"\n"+priceFormatter.format(prices[1])); button4.setText(drinks[2]+"\n"+priceFormatter.format(prices[2]));

button01.setText(drinks[3]+"\n"+priceFormatter.format(prices[3])); button02.setText(drinks[4]+"\n"+priceFormatter.format(prices[4])); button03.setText(drinks[5]+"\n"+priceFormatter.format(prices[5]));

// Initialize checkboxes

checkBox1 = (CheckBox) findViewById(R.id.checkBox1);

checkBox2 = (CheckBox) findViewById(R.id.checkBox2);

// Initialize textboxes

username = (EditText) findViewById(R.id.editText1);

pin = (EditText) findViewById(R.id.editText2);

UDPserverThread = new Thread(new Runnable() { public void run() { UDPServer(); }});

UDPserverThread.start();

} // End onCreate

/////// Load previously saved username/password

protected void onResume(){

super.onResume();

SharedPreferences settings = getSharedPreferences(PREFS\_NAME, 0);

String savedUserName = settings.getString("username", "");

String savedPin = settings.getString("pin", "");

username.setText(savedUserName);

pin.setText(savedPin);

}

//////// These methods are called on button click ////////

public void drink1\_clicked(View view) { // User clicked Drink 1

sendToServer(drinks[0],prices[0]);

}

public void drink2\_clicked(View view) { // User clicked Drink 2

sendToServer(drinks[1],prices[1]);

}

public void drink3\_clicked(View view) { // User clicked Drink 3

sendToServer(drinks[2],prices[2]);

}

public void drink4\_clicked(View view) { // User clicked Drink 4

sendToServer(drinks[3],prices[3]);

}

public void drink5\_clicked(View view) { // User clicked Drink 5

sendToServer(drinks[4],prices[4]);

}

public void drink6\_clicked(View view) { // User clicked Drink 6

sendToServer(drinks[5],prices[5]);

}

// Function that sends drink to server

public void sendToServer(final String drink, double price){

// Check if username/pin is of correct format

boolean correctInfo = true;

// Check if username is all letters

if ((!isLetters(username.getText().toString()) || username.getText().toString().length() == 0)) {

alertbox("Your username must contain letters only.");

correctInfo = false;

}

// Check if pin is a 4 digit number

if ((correctInfo) && ((!isInteger(pin.getText().toString())) || (numDigits(pin.getText().toString()) != 4))) {

alertbox("Your pin must be 4 numbers.");

correctInfo = false;

}

// If the information is correct open a dialog box and send order to server

if (correctInfo) {

// Open dialog to confirm order

Builder builder = new AlertDialog.Builder(this);

builder.setMessage("This will charge your credit card " + priceFormatter.format(price) + ". Proceed?")

.setCancelable(true)

// if they click yes below code runs

.setPositiveButton("Yes, order a " + drink,

new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog,

int which) {

// Save username and password

saveInformation();

// Send drink to server

UDP\_Send(drink);

// wait til receive from bartender

int n = 0; double y; int limit = 100000000;

if (!receive) {

while (!receive & n < limit) {

n++; y = 3\*3\*2.3+56; // this is used to timeout if no response

} // timeout after certain amount of itme

}

// If no response, start over

if (n >= limit) {

alertbox("There was no response from the bartender. Please try again.");

}

// if there is a response

else {

// If order went through

if (rcvMessage[0].equals("1")) {

alertbox("Thank you for your purchase! You're order #" + rcvMessage[1] +

".\nCurrently serving order #" + rcvMessage[2] +

".\nYou'll receive a text message when your drink is ready.");

}

// If username was wrong

else if(rcvMessage[0].equals("2")) {

alertbox("The username \"" + username.getText().toString() +

"\" does not exist. Please setup an account at an " +

"AutoBev kiosk and try ordering a " + drink + " again. " +

"Your credit card has not been charged.");

}

// If password was wrong

else if(rcvMessage[0].equals("3")) {

alertbox("Your password did not match the saved password for username " +

"\"" + username.getText().toString() + "\". Please try ordering a " +

drink + " again. Your credit card has not been charged.");

}

} // end else

// reset receive flag

receive = false;

}

})

// If they click no, nothing happens

.setNegativeButton("No",

new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog,

int which) {

//////////////////////////////////////

// if they click no this code runs //

//////////////////////////////////////

}

}

);

AlertDialog dialog = builder.create();

dialog.show();

}

} // end sentToServer function

// Thread that is constantly receiving packets

public void UDPServer()

{

while(true) {

final String TAG = "UDP playback";

try {

byte[] buffer = new byte[140];

int port = 9050;

// Create new UDP-Socket and packet

DatagramSocket socket = new DatagramSocket(port);

DatagramPacket packet = new DatagramPacket(buffer, buffer.length );

// Receive the UDP-Packet

socket.receive(packet);

// Set flag to tell main code that message has been received

receive = true;

rcvMessage = new String(buffer).split("#");

}

catch (Exception e) {

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

////////// Function that sends UDP message to C# server /////////////

public void UDP\_Send(String drink){

try {

socket = new DatagramSocket();

address = InetAddress.getByName("10.40.167.6");

String port = "9050";

int pnum = Integer.parseInt(port);

byte[] messageBytes = new byte[0];

// Get phone number of Android

String phoneNumber = "null";

try {

TelephonyManager tm = (TelephonyManager)

getSystemService(Context.TELEPHONY\_SERVICE);

phoneNumber = tm.getLine1Number();

}

catch (Exception e) {

}

// Get IP address of droid

String IP = getIpAddress();

// Create complete message with num,username,pin, and order

String drinkComplete = "&" + phoneNumber + "#" + IP + "#" + username.getText() +

"#" + pin.getText() + "#" + drink + "#";

// Convert message to bytes

messageBytes = drinkComplete.getBytes();

// Create datagram packet

packet = new DatagramPacket(messageBytes, messageBytes.length, address, pnum);

packet.setLength(messageBytes.length);

// send packet to server

socket.send(packet);

// close socket

socket.close();

}

catch (IOException io) {

}

}

////////// Function that returns true if string is made up of all letters //////////

public boolean isLetters(String str)

{

for(int i = 0; i < str.length(); i++)

{

int Ascii = (int)str.charAt(i);

if(Ascii < 65 || Ascii > 122){ // check if it not a letter

return false; }

}

return true; // if all letters

}

////////// Function that returns true if string is made up of all numbers //////////

public boolean isInteger(String i)

{

try

{

Integer.parseInt(i);

return true;

}

catch(NumberFormatException nfe)

{

return false;

}

}

////////// Function that returns number of digits in integer //////////

public int numDigits(String str){

int x = Integer.parseInt(str);

int count = 0;

while(x > 0)

{

x /= 10;

count++;

}

return count;

}

////////// Function that displays alert box //////////////

protected void alertbox(String mymessage)

{

new AlertDialog.Builder(this)

.setMessage(mymessage)

.setCancelable(true)

.setNeutralButton(android.R.string.ok,

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton){}

})

.show();

}

///////// Function that gets Ip address of device

public String getIpAddress() {

try {

for (Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfaces(); en.hasMoreElements();) {

NetworkInterface intf = en.nextElement();

for (Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses(); enumIpAddr.hasMoreElements();) {

InetAddress inetAddress = enumIpAddr.nextElement();

if (!inetAddress.isLoopbackAddress()) {

return inetAddress.getHostAddress().toString();

}

}

}

} catch (Exception E) {

}

return null;

}

///////// Function that saves username and password

public void saveInformation() {

if (checkBox1.isChecked()) {

SharedPreferences settings = getSharedPreferences(PREFS\_NAME, 0);

SharedPreferences.Editor editor = settings.edit();

editor.putString("username",username.getText().toString());

editor.commit();

}

// Save pin if save checkbox is checked

if (checkBox2.isChecked()) {

SharedPreferences settings = getSharedPreferences(PREFS\_NAME, 0);

SharedPreferences.Editor editor = settings.edit();

editor.putString("pin",pin.getText().toString());

editor.commit();

}

}

} //end class

**9.2.3.B – main.xml: layout of app code**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_height=*"fill\_parent"* android:layout\_width=*"wrap\_content"* android:orientation=*"vertical"*>

<TextView android:layout\_width=*"wrap\_content"* android:textSize=*"30sp"* android:text=*"@string/welcome"* android:id=*"@+id/textView3"* android:layout\_height=*"wrap\_content"* android:layout\_gravity=*"center"*></TextView>

<TextView android:layout\_height=*"wrap\_content"* android:textSize=*"16sp"* android:id=*"@+id/welcomeText"* android:layout\_width=*"fill\_parent"* android:layout\_weight=*"1"* android:gravity=*"center\_vertical|center\_horizontal"* android:text=*"@string/enterInfo"*></TextView>

<RelativeLayout android:layout\_height=*"wrap\_content"* android:id=*"@+id/relativeLayout2"* android:layout\_width=*"fill\_parent"* android:layout\_weight=*"1"*>

<TextView android:id=*"@+id/textView1"* android:textSize=*"17sp"* android:layout\_alignParentLeft=*"true"* android:layout\_height=*"50dip"* android:text=*"@string/username"* android:layout\_width=*"85dip"* android:gravity=*"center"*></TextView>

<EditText android:layout\_height=*"50dip"* android:id=*"@+id/editText1"* android:layout\_toRightOf=*"@+id/textView1"* android:layout\_alignTop=*"@+id/textView1"* android:layout\_alignBottom=*"@+id/textView1"* android:layout\_width=*"155dip"*></EditText>

<CheckBox android:layout\_height=*"wrap\_content"* android:text=*"Save"* android:id=*"@+id/checkBox1"* android:layout\_toRightOf=*"@+id/editText1"* android:layout\_alignTop=*"@+id/editText1"* android:layout\_alignBottom=*"@+id/editText1"* android:layout\_gravity=*"left"* android:layout\_width=*"fill\_parent"*></CheckBox>

</RelativeLayout>

<RelativeLayout android:layout\_height=*"wrap\_content"* android:id=*"@+id/relativeLayout3"* android:layout\_weight=*"1"* android:layout\_width=*"wrap\_content"*>

<TextView android:id=*"@+id/textView2"* android:textSize=*"17sp"* android:layout\_alignParentLeft=*"true"* android:layout\_height=*"50dip"* android:text=*"@string/pin"* android:layout\_width=*"85dip"* android:gravity=*"center"*></TextView>

<EditText android:layout\_height=*"50dip"* android:id=*"@+id/editText2"* android:layout\_toRightOf=*"@+id/textView2"* android:layout\_alignTop=*"@+id/textView2"* android:layout\_alignBottom=*"@+id/textView2"* android:inputType=*"textPassword"* android:layout\_width=*"155dip"*></EditText>

<CheckBox android:layout\_height=*"wrap\_content"* android:text=*"Save"* android:id=*"@+id/checkBox2"* android:layout\_toRightOf=*"@+id/editText2"* android:layout\_alignTop=*"@+id/editText2"* android:layout\_alignBottom=*"@+id/editText2"* android:layout\_width=*"wrap\_content"*></CheckBox>

</RelativeLayout>

<RelativeLayout android:id=*"@+id/relativeLayout1"* android:layout\_width=*"fill\_parent"* android:layout\_height=*"wrap\_content"* android:gravity=*"center"* android:layout\_weight=*"1"*>

<Button android:id=*"@+id/button2"* android:textSize=*"15dip"* android:layout\_alignParentTop=*"true"* android:layout\_width=*"wrap\_content"* android:layout\_height=*"wrap\_content"* android:onClick=*"drink1\_clicked"* android:text=*"@string/drink1"*></Button>

<Button android:id=*"@+id/button3"* android:textSize=*"15dip"* android:layout\_width=*"wrap\_content"* android:layout\_toRightOf=*"@+id/button2"* android:layout\_height=*"wrap\_content"* android:layout\_alignTop=*"@+id/button2"* android:layout\_alignBottom=*"@+id/button2"* android:text=*"@string/drink2"* android:onClick=*"drink2\_clicked"*></Button>

<Button android:id=*"@+id/button4"* android:textSize=*"15dip"* android:layout\_width=*"wrap\_content"* android:layout\_toRightOf=*"@+id/button3"* android:layout\_height=*"wrap\_content"* android:layout\_alignTop=*"@+id/button3"* android:layout\_alignBottom=*"@+id/button3"* android:text=*"@string/drink3"* android:onClick=*"drink3\_clicked"*></Button>

</RelativeLayout>

<RelativeLayout android:layout\_height=*"wrap\_content"* android:id=*"@+id/RelativeLayout01"* android:layout\_width=*"fill\_parent"* android:onClick=*"drink4\_clicked"* android:gravity=*"center"* android:layout\_weight=*"1"*>

<Button android:layout\_height=*"wrap\_content"* android:text=*"@string/drink4"* android:onClick=*"drink4\_clicked"* android:layout\_width=*"wrap\_content"* android:layout\_alignParentTop=*"true"* android:id=*"@+id/Button01"*></Button>

<Button android:layout\_height=*"wrap\_content"* android:layout\_toRightOf=*"@+id/Button01"* android:text=*"@string/drink5"* android:onClick=*"drink5\_clicked"* android:layout\_width=*"wrap\_content"* android:id=*"@+id/Button02"* android:layout\_alignTop=*"@+id/Button01"* android:layout\_alignBottom=*"@+id/Button01"*></Button>

<Button android:layout\_height=*"wrap\_content"* android:textSize=*"15dip"* android:layout\_toRightOf=*"@+id/Button02"* android:text=*"@string/drink6"* android:onClick=*"drink6\_clicked"* android:layout\_width=*"wrap\_content"* android:id=*"@+id/Button03"* android:layout\_alignTop=*"@+id/Button02"* android:layout\_alignBottom=*"@+id/Button02"*></Button>

</RelativeLayout>

</LinearLayout>