**Home Gizmos Inc.**

*School of Engineering Science, Simon Fraser University*

*8888 University Drive, Burnaby, BC, V5A 1S6*

***home-gizmos@sfu.ca***

October 31, 2002

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6

**Re: ENSC 340 Project Design Specifications**

Dear Dr. Rawicz:

Attached is the Home Gizmos group's *Design Specification for a Home Inventory System*, which outlines the basic design for our project for ENSC 340. We are in the process of designing and implementing a unit that will scan in common Universal Price Codes (UPCs), in order to keep an easy and accurate inventory of items by connecting with a computer. Also, we are developing easy-to-use software that will will keep track of the fridge's inventory.

The purpose of this design specification is to detail the design of our SmartFridge system. This document will outline the pertinent information that is required for the successful completion of our project by the end of the year.

Home Gizmos consists of five fifth-year-engineering students, whose imagination and technical skills will be of great value to our project: The SmartFridge™. Michael Nelson, Alexander Dunfield, Cameron Kenny, Colin Knight, and Shaun Jackman make up the core of Home Gizmos. If you have any questions or concerns about our design specification, please do not hesitate to contact me by phone at 604-929-6611, or email our group at home-gizmos@sfu.ca.

Sincerely,

M. Nelson

Michael Nelson
President and CEO
Home Gizmos

# Design Specifications for the SmartFridge Home Inventory System

*Revision 1*

*October 31, 2002*

**Project Team**

Alex Dunfield

Mike Nelson

Cam Kenny

Shaun Jackman

Colin Knight

**Contact**

Alex Dunfield

alexd@sfu.ca

home-gizmos@sfu.ca

## Executive Summary

People everywhere endure the chore of shopping. The SmartFridge is aimed at simplifying the shopping process by allowing the user to effortlessly create shopping lists that they can take to the store or use to order groceries online. The SmartFridge will even track the date that items with a limited shelf life were added to the inventory, and make recipe suggestions based on the current contents of the refrigerator.

The SmartFridge consists of hardware, software, and firmware components. The hardware of the SmartFridge is made up of:

- a scanning wand that will read in the barcode and convert it into an electrical signal
- a microprocessor to
  - interpret this signal, and convert it into a binary number
  - store the binary number for later use
  - control all of the peripheral functions of the reader
- a RS-232 adapter to put the signal onto the serial port

The software will be designed to be easy to use, and will have the following capabilities:

- match the received barcode to a product
- keep track of the current inventory
- print a shopping list
- order needed groceries online, with input from the user
- making recipe suggestions based on the current inventory

The firmware will control the microprocessor in interpreting the signal, adjusting for system noise, and keeping track of peripheral functions, such as monitoring the battery, and keeping track of remaining memory capacity.

A working prototype of the SmartFridge will be completed in December 2002.

## Table of Content

## List of Figures

# List of Tables

## Glossary and Acronyms

**AppWizard:** A Wizard included in Visual C++ that allows an application framework to be generated according to information the programmer provides about the nature of the application.

**ClassWizard:** A Wizard included in Visual C++ that generates much of the low-level code necessary to respond to user events such as button clicks.

**Dialog:** A Windows term used to refer to a window that displays or collects information from the user. Dialogs typically incorporate buttons, text entry boxes, and lists, and contain a dialog frame with minimize, maximize, and close buttons.

**Home Inventory:** The actual products present in the user's refrigerator, pantry, etc.

**Hotkey:** A sequence of keys that are used to activate a menu item, for example Ctrl + V is used for paste.

**Inventory Management Browser (IMB):** A structure that allows the user to browse categories of food as well as see the products in each category. In addition, the quantity of each product and the date of the oldest instance of that product will be displayed in the IMB.

**Modal:** A Dialog box that blocks all other windows from the same application from accepting input from the user.

**Modeless:** A dialog box that allows the user to interact with other windows in the same application while the current window is still open.

**Product Library:** The list of all of the different possible products that could be added to the user's Virtual Inventory. Ideally, the Product Library would contain all possible products. However, for the initial revision, we will be working with a product library that has 20-50 products in it.

**Product Library Browser:** A structure that allows the user to browse the categories of food as well as see the available products in each category.

**Resource:** A file that stores data defined at compilation time such as images and screen layouts.

**Scanne**r: A synonym for "Scanning Wand". Refer to the Scanner Wand entry.

**Scan (Scanned) out:** The process of using the scanning wand to scan the bar code of an old product that is ready to be discarded. When the Scan Workspace in the SmartFridge Software is entered and the Download Scanning Wand button is clicked, the codes of the products that were scanned out are transferred to the SmartFridge software and the quantity of the appropriate products is decreased.

**Scan (Scanned) in:** The process of using the scanning wand to scan the bar code of a new product that has just arrived in the Home Inventory. When the Scan Workspace in the SmartFridge Software is entered and the Download Scanning Wand button is clicked, the codes of the products that were scanned in are transferred to the SmartFridge software and the quantity of the products is increased appropriately.

**Scanned Items List:** The list in the Scan workspace that shows the user what items were scanned in and scanned out recently and allows the user to adjust the quantity.  Once the Save and Return button is pressed, the Scanned Items List is cleared and the items are added to or subtracted from the Virtual Inventory.

**Scanning Wand:** The portable device that is used to scan in and scan out items by being passed across the bar code. The scanning wand is connected to a personal computer through a serial cable, and the product codes that were scanned can be stored into the virtual inventory.

**Shopping List:** The region in the Shop window containing the list of items to shop for.

**SmartFridge:** A bar code scanner based home inventory system.

**User:** Any person who is using the SmartFridge scanning wand or graphical user interface.

**Virtual Inventory:** The inventory of the kitchen according to the SmartFridge software.

**Wand:** see Scanning Wand.

**Workspace:** The screen in the user interface that allows the user to apply a specific function. For example: the Shop workspace.

## Acronyms

| | |
|---|---|
| **EAN:** | European Article Numbering System |
| **DDL:** | Data Definition Language |
| **DML:** | Data Manipulation Language |
| **GUI:** | Graphical User Interface |
| **IMB:** | Inventory Management Browser |
| **PC:** | Personal Computer |
| **PLB:** | Product Library Browser |
| **UPC:** | Universal Price Code |
| **UART:** | Universal Asynchronous Receiver-Transmitter |

# 1 Introduction

The SmartFridge is a system that will keep track of the contents the user's home inventory by reading in the barcodes of groceries, and storing them on a handheld unit. The user will upload these codes to their PC via the serial port. On the PC, the user can view the contents of their fridge, print out a shopping list, order groceries online, or get recipe suggestions based on the available ingredients.

## *1.1 Scope*

This document outlines the design specifications of the SmartFridge. The SmartFridge design team will use this document as a framework on which a working prototype be built. Though comprehensive, this specification will not completely outline the final design of the project. The exact design will be determined as the project progresses, although all of the major functional blocks have been planned out.

## *1.2 Intended Audience*

This document is primarily created for the SmartFridge design team to provide direction in the design of the SmartFridge and ensure that there are not oversights in the design. The Home Gizmos management team will use this document as a measuring tool to evaluate the SmartFridge as it develops to ensure that the project lives up to its expectations.

# 2 System Overview

In order to build the Smart Fridge, a number of different sections were identified. These sections include Scanning Wand, Level Converting, Microprocessor, Serial Line Driver and Personal Computer as shown below in Figure 2.

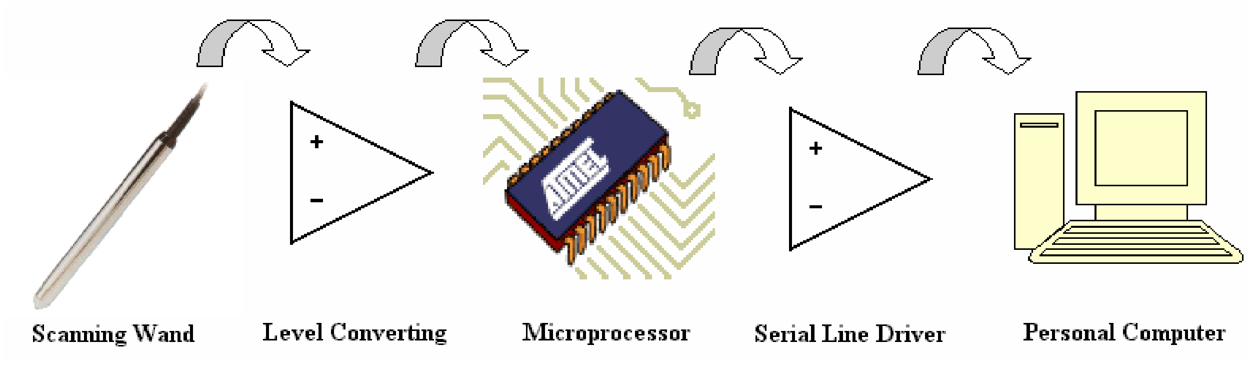


**Figure 1: SmartFridge Sections**

## 2.1 Scanning Wand

The scanning wand serves one key purpose, to change the printed barcode into an electrical signal. To achieve this function, we searched for a solution that uses and LED light source and an optical detector that would meet our functional requirements. We have identified a few options for implementing the scanning wand.

To meet our initial prototyping needs, we decided to use the Tysso PEN40-R scanning wand that is designed for barcode scanning. This device consists of an LED light source, a photo detector and a lens to achieve the required resolution for barcode reading. For operation the wand has three terminals: 5V, Signal and ground. With our device, we only need to connect the power source to the wand and then we have the barcodes electrical signal at the wand's output.

We considered trying to source the components for the scanning wand separately but we came to the conclusion that the Tysso wand should help minimize the time needed to sort out the optics so that we can focus on other areas of the SmartFridge. Later in the project cycle, this wand can be replaced by a more custom optical setup. This setup would include barcode sensors and tips produced by Agilent for use in barcode readers.

## 2.2 Level Converting

The output of the signal from the scanning device cannot be directly inputted to our microcontroller. The reason for this is an optical detector acts more like a current source than a voltage source. Because of this behavior of optical detectors, the voltage measured at the output is minimal, on the order of 40mV.

In order to input a dependable signal into our microcontroller, we decided to use an op amp to change the signal from the wand into a near 5V swinging voltage signal to be fed into the microcontroller's analog comparator. This section also included a simple voltage divider to create the second signal for the analog comparator.

## 2.3 Microcontroller

In order to meet our processing needs for the SmartFridge, we decided on the Atmel AT90S8515. This device is responsible for decoding barcode, storing barcodes in memory, communicating to the host computer and any other features needed for the SmartFridge device. Much of the function of the microcontroller is described in the next section. We decided to use the AT90S8515 because:

- the device has an analog comparator, ideal for our needs as the different colored bars of a barcode will produce different voltage levels
- the device's analog comparator can be used with function input capture allowing us to time between events (key for decoding barcodes as the barcode might not always be swept at the same speed)
- the device has ample input/output capabilities to meet our needs
- the device has a UART for serial communication with a PC
- the device has a large amount of flash (8k) onboard
- the device is relatively inexpensive for the power the chip has
- AVR devices can be programmed using C, saving the hassle of learning to use a proprietary assembly language
- Atmel has an array of other devices including bigger devices, smaller devices and USB devices allowing easy migration to other devices later in project cycle or in future products

## 2.4 Serial Line Driver

The Tysso wand operates at 5V and the Atmel device can operate over a voltage range from 2.7V to 5V however a serial RS-232 interface requires both positive and negative voltages sources that we do not have readily available with our existing components. There are many different devices, including the Maxim 202E, that are available to create the required voltages for a serial RS-232 interface from a single positive source. The Maxim 202E is designed specifically to give the appropriate voltages for RS-232, which could not be obtained from the microprocessor.

## 2.5 Personal Computer

The Personal Computer is a key part in the SmartFridge system. The computer is required to run the application software, written in Visual C++, and the system's Access database. These components are used to implement many of the SmartFridge's unique features including:

- matching the barcode values to the actual product
- tracking the current inventory of the kitchen
- printing a shopping list
- ordering needed groceries online
- making recipe suggestions based on the current inventory
- managing the inventory to account for mistakes and miss-scans

# 3 Hardware Design

The hardware consists of the scanning wand, a level converter, a microprocessor, and a serial line driver. The primary function is to scan and store barcodes digitally, for use by the inventory on the user's PC. A Hardware schematic can be found in Appendix B.

## 3.1 Scanning Barcodes

The American barcode standard, UPC, is a subset of the international barcode tandard EAN. For the prototype, the SmartFridge will be able to scan UPC barcodes. The firmware can then be extended to handle EAN barcodes. We chose to use the PEN-40R scanning wand to do the scanning, as it is built for commercial applications, and has the required resolution of 5 mils.

One of the most important parameters of our design is the first-scan success rate, which depends on the decoding algorithm's resistance to sensor noise due to bar-code distortion, or system noise. Our algorithm has been designed to mitigate both of these effects.

When barcodes are encoded, they include a checksum in the twelfth digit for error detection. Any single digit substitution error will be caught by the checksum. The probability of a double-digit substitution error that happens to produce a valid checksum is less than 0.1%.

The algorithm will attempt to process the recorded data in the forward direction. If that fails, the algorithm will then attempt to process the recorded data in the reverse direction. Only one direction can succeed and that result will generate the scanned barcode. The decoding algorithm has also been designed to be resistant to bleeding black ink and system noise. For more information on the algorithm, see the mathematical analysis of the algorithm in the next section.

When barcodes are encoded, there is a clock encoded in the signal. The decoding algorithm uses no fixed unit of time or distance, as all measurements are with respect to the encoded clock. As such, decoding can cope with barcodes of any reasonable width, and likewise any speed of reading, provided that it is relatively constant.

For the purposes of digitally encoding barcodes, it should be noted that a barcode is 12 digits long.

$$n = \left\lceil \frac{\ln\left(10^{12}\right)}{8\ln(2)} \right\rceil = \lceil 4.98 \rceil = 5$$

**Equation 1: Bytes to Store a 12 Digit Barcode**

The selected micro-controller, the AT90S8515, has 512 bytes of EEPROM. Thus, 100 barcodes can be stored before uploading to the PC.

## 3.2 Hardware User Interface

During the process of scanning, the user will need to have some control over the wand. This control is necessary so that the user is capable to control what's happening, and gets feedback about how the scans are going. The user interface for the scanning wand will include:

- a small piezo speaker to produce an audible beep
- a small push-button for user input
- a LED that will blink when the battery is low, and to stay on steadily to indicate the memory is at capacity
- a small toggle switch to select whether the wand is scanning items in or out

## 3.3 PC Interface

Our microprocessor, the AT90S8515 has one UART for communication with the PC. A MAX202 line driver will be used to convert logic level to RS232 signaling level voltage. Atmel does have versions of the AVR with on-board USB, so a production version of the wand can be migrated to this AVR with minimal engineering changes. Our selected microprocessor's UART, when using a 4 MHz crystal, is capable of transferring data at a rate of 19.2 kbps.

## 3.4 Battery Power

The low-voltage (2.7V) AT90S8515 was selected because of its minimal power drain. Since scanned barcode data is stored in non-volatile EEPROM, the micro-controller can power itself down when not in use. This allows for almost zero idle power consumption. All of these features result in minimal power consumption, and thus prolong battery life.

## 3.5 Environmental Requirements

All of the parts have been selected to operate in commercial temperature ranges (0°C to 70°C) and in all non-condensing humidity conditions.

## 3.6 Enclosure Requirements

The enclosure has yet to be designed, but it should be ergonomic to use, and appealing to the eye. It should also be as small as the internal components will allow.

# 4 Firmware Design

The bulk of the firmware is geared towards interpreting the barcode and turning it into a digitally sored value. All of the functionality is relatively simple, and will be outlined after the decoding algorithm is functioning properly.

## *4.1 UPC Standards*

UPC barcodes consist of 12 digits, with the last digit being a checksum. Each digit is encoded by a sequence of four bars. Each bar width is an integer multiple of the unit width. The sum of these widths is always seven units, as shown in Table 1: Encoding of Digits.

**Table 1: Encoding of Digits**

| Digit | Encoding |
|---|---|
| 0 | 3 2 1 1 |
| 1 | 2 2 2 1 |
| 2 | 2 1 2 2 |
| 3 | 1 4 1 1 |
| 4 | 1 1 3 2 |
| 5 | 1 2 3 1 |
| 6 | 1 1 1 4 |
| 7 | 1 3 1 2 |
| 8 | 1 2 1 3 |
| 9 | 3 1 1 2 |

When the scanning wand is passed over the barcode, it produces alternating high and low voltages as it passes over the black and white bars. The barcode is decoded by measuring the widths of these pulses. For each digit we obtain a vector of four widths, [*a b c d*]. The unit width of the barcode can be obtained by Equation 2: Unit width of barcodes.

$$u = \frac{a+b+c+d}{7}$$

**Equation 2: Unit width of barcodes**

To decode the barcode, we normalize the measured widths by looking at [*a b c d*] / *u*. This would result in a vector that can be compared to the encoding scheme in Table 1. For undistorted measurements, this would produce perfect decoding. However, barcodes measurements are subject to two typical types of error, and this will be outlined in the next section.

## *4.2 Error Compensation*

When the barcode is printed, the black ink tends to bleed in a deterministic way that causes the black bars to be uniformly wider, and white bars to be uniformly thinner. Also, each measurement will be subject to some random error caused by system noise and by changes in the speed the user moves the scanner across the barcode. A model for the actual measurements, outlined in Equation 3.

$$\begin{aligned}
\delta &= bleed \\
\varepsilon &= noise \\
a &= a' + \delta + \varepsilon_a \\
b &= a' - \delta + \varepsilon_b \\
c &= a' + \delta + \varepsilon_c \\
d &= a' - \delta + \varepsilon_d
\end{aligned}$$

**Equation 3: Parameters of barcode noise**

We can eliminate the effect of bleed by looking at linear combinations of [*a b c d*] instead of the direct measurements. For example, *a*+*b* corresponds to the time from the first rising edge to the second rising edge, and *b*+*c* corresponds to the time from the first falling edge to the second falling edge. These two derivative values have the benefit that they are not affected by bleed.

$$\begin{aligned}
r &= a + b = a' + \delta + \varepsilon_a + b' - \delta + \varepsilon_b = a' + b' + \varepsilon_{ab} \\
f &= b + c = b' + c' + \varepsilon_{bc}
\end{aligned}$$

**Equation 4: Definitions of rising and falling edge barcode parameters**

**Table 2: Rising and falling edge parameters**

| Digit | Rising edge (*r*) | Falling edge (*f*) |
|---|---|---|
| 0 | 5 | 3 |
| 1 | 4 | 4 |
| 2 | 3 | 3 |
| 3 | 5 | 5 |
| 4 | 2 | 4 |
| 5 | 3 | 5 |
| 6 | 2 | 2 |
| 7 | 4 | 4 |
| 8 | 3 | 3 |

| | | |
|---|---|---|
| 9 | 4 | 2 |

Unfortunately, this reduced set of characteristics cannot be used to fully describe all possible digits. We can see that 2 & 8 share the same widths, as well as 1 & 7. We must derive one further characteristic from our raw measurements to distinguish between these last two cases. This further characteristic is the duty cycle that is shown in Equation 5. The duty cycle is the relative amount of the character that is black.

$$p = a - b + c - d = a' + b' + c' + d' + 4\delta + \varepsilon_{abcd}$$

**Equation 5: Definition of duty cycle parameter**

Though at first it looks like this characteristic has far too much noise inherent in it, we can see in practice it only takes on two extreme values that can be easily distinguished.

**Table 3: Duty cycle parameters by digit**

| Digit | Duty cycle (p) |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 2 | 1 |
| 3 | -3 |
| 4 | 1 |
| 5 | 1 |
| 6 | -3 |
| 7 | -3 |
| 8 | -3 |
| 9 | 1 |

Looking at duty cycle as a last comparison allows us to distinguish 2 from 8, and 1 from 7.

Using these three parameters, [$r\,f\,p$], derived from the original [a b c d], we can distinguish all possible encodings. We must now derive a decision rule set based on these parameters.

We can see the rising edge takes on one of four values, [2 3 4 5]. We obtain our first soft bit of information by testing if $r$ exceeds the mean of these values:

$$x_1 : r > 3.5u$$
$$x_1 = r - 3.5u = a + b - 3.5u = a + b - 3.5(a + b + c + d)/7$$
$$x_1 \propto 2a + 2b - a - b - c - d = a + b - c - d$$

**Equation 6: First soft bit calculations**

We can now combine these variables to generate new lookup table, shown in Table 4.

**Table 4: Decision lookup table**

| Digit | $x_1$ | Decision |
|---|---|---|
| 0 | 3 | True |
| 1 | 1 | True |
| 2 | -1 | False |
| 3 | 3 | True |
| 4 | -3 | False |
| 5 | -1 | False |
| 6 | -3 | False |
| 7 | 1 | True |
| 8 | -1 | False |
| 9 | 1 | True |

We then check if $r$ is between 2.5 and 4.5 times the unit width $u$, and generate the second test bit, $x_2$.

$$x_2 : r < 2.5u \cup r > 4.5u$$
$$x_{2a} = r - 4.5u = a + b - 4.5(a+b+c+d)/7 \propto 14a + 14b - 9a - 9b - 9c - 9d = 5a + 5b - 9c - 9d$$
$$x_{2b} = 2.5u - r = 2.5(a+b+c+d)/7 - (a+b) \propto 5a + 5b + 5c + 5d - 14a - 14b = -9a - 9b + 5c + 5d$$
$$x_2 = x_{2a} > 0 \cup x_{2b} > 0$$

**Equation 7: Second bit calculations**

Note that $x_{2a}$ is only true if $x_1$ is true, and $x_{2b}$ is only true if $x_1$ is false, so we need only consider the particular case that applies. This expands our decision table:

**Table 5: Decoding Lookup Table**

| Digit | $x_1$ | Decision | $x_{2a}$ | $x_{2b}$ | Decision |
|---|---|---|---|---|---|
| 0 | 3 | True | 7 |  | True |
| 1 | 1 | True | -7 |  | False |
| 2 | -1 | False |  | -7 | False |
| 3 | 3 | True | 7 |  | True |
| 4 | -3 | False |  | 7 | True |
| 5 | -1 | False |  | -7 | False |
| 6 | -3 | False |  | 7 | True |

| | | | | | |
|---|---|---|---|---|---|
| 7 | 1 | True | -7 | | False |
| 8 | -1 | False | | -7 | False |
| 9 | 1 | True | -7 | | False |

We now look at the falling edge parameter. If we look at **Error! Reference source not found.**, we notice that when $r$ is one of {3,5}, $f$ is one of {3,5}. Likewise, when $r$ is one of {2,4}, f is one of {2,4}. We can use this to our advantage by choosing our cutoff based on $x_1$ and $x_2$.

$$x_{3a} : f > 4$$
$$x_{3a} = f - 4u = b + c - 4(a+b+c+d)/7 \propto 7b - 7c - 4a - 4b - 4c - 4d = -4a + 3b + 3c - 4d$$
$$x_{3b} : f < 3$$
$$x_{3b} = 3u - f = 3(a+b+c+d)/7 - (b+c) \propto 3a + 3b + 3c + 3d - 7b - 7c = 3a - 4b - 4c + 3d$$

**Equation 8: Third bit calculations**

We now look at our updated decision table:

**Table 6: Updated Decoding Lookup Table**

| Digit | $x_1$ | Decision | $x_{2a}$ | $x_{2b}$ | Decision | $x_{3a}$ | $x_{3b}$ | Decision |
|---|---|---|---|---|---|---|---|---|
| 0 | 3 | True | 7 | | True | -7 | | False |
| 1 | 1 | True | -7 | | False | | -7 | False |
| 2 | -1 | False | | -7 | False | -7 | | False |
| 3 | 3 | True | 7 | | True | +7 | | True |
| 4 | -3 | False | | 7 | True | | -7 | False |
| 5 | -1 | False | | -7 | False | +7 | | True |
| 6 | -3 | False | | 7 | True | | +7 | True |
| 7 | 1 | True | -7 | | False | | -7 | False |
| 8 | -1 | False | | -7 | False | -7 | | False |
| 9 | 1 | True | -7 | | False | | +7 | True |

We can see 1 & 7, and 2 & 8 share the same decision vector. To distinguish we now look at the duty cycle parameter, p which takes on only two values: [-3 1]. We set our decision threshold at their mean, -1.

$$x_4 : p > -1u$$
$$x_4 = p + u = (a-b+c-d)+(a+b+c+d)/7 \propto 7a-7b+7c-7d+a+b+c+d = 8a-6b+8c-6d$$

**Equation 9: Fourth bit calculation**

We can now find our entire decision table by looking at Table 7.

**Table 7: Final Lookup Table**

| Digit | $x_1$ | Decision | $x_{2a}$ | $x_{2b}$ | Decision | $x_{3a}$ | $x_{3b}$ | Decision | $x_4$ | Decision |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | True | 7 | | True | -7 | | False | 14 | True |
| 1 | 1 | True | -7 | | False | | -7 | False | 14 | True |
| 2 | -1 | False | | -7 | False | -7 | | False | 14 | True |
| 3 | 3 | True | 7 | | True | +7 | | True | -14 | False |
| 4 | -3 | False | | 7 | True | | -7 | False | 14 | True |
| 5 | -1 | False | | -7 | False | +7 | | True | 14 | True |
| 6 | -3 | False | | 7 | True | | +7 | True | -14 | False |
| 7 | 1 | True | -7 | | False | | -7 | False | -14 | False |
| 8 | -1 | False | | -7 | False | -7 | | False | -14 | False |
| 9 | 1 | True | -7 | | False | | +7 | True | 14 | True |

We can see this last parameter distinguishes 1 & 7, and 2 & 8. If we take the outcome of each decision rule as a single bit, we can build a table, as seen in Table 8.

**Table 8: Conversion of barcode digits to binary**

| Digit | $[x_1\ x_2\ x_3\ x_4]$ Binary | $[x_1\ x_2\ x_3\ x_4]$ Decimal |
|---|---|---|
| 0 | 1101 | 13 |
| 1 | 1001 | 9 |
| 2 | 0001 | 1 |
| 3 | 1110 | 14 |
| 4 | 0101 | 5 |
| 5 | 0011 | 3 |
| 6 | 0110 | 6 |
| 7 | 1000 | 8 |

| | | |
|---|---|---|
| 8 | 0000 | 0 |
| 9 | 1011 | 11 |

The above algorithm can be summed up in eighteen lines of assembly language code as shown in listing 4.1.

Listing 4.1.

```
// falling edge
int16_t x1 = +1*a +1*b -1*c -1*d;
int16_t x2 = x1 > 0 ?
         +5*a +5*b -9*c -9*d :
         -9*a -9*b +5*c +5*d;

// rising edge
int16_t x3 = (x1 > 0) == (x2 > 0) ?
         -4*a +3*b +3*c -4*d :
         +3*a -4*b -4*c +3*d;

// duty cycle
int16_t x4 = +4*a -3*b +4*c -3*d;

uint8_t x = (x1>0) << 3 | (x2>0) << 2 | (x3>0) << 1 | (x4>0);
static uint8_t table[] =
    {8, 2, -1, 5, -1, 4, 6, -1, 7, 1, -1, 9, -1, 0, 3, -1};
return table[x];
```

# 5 Software Design

The software is responsible for the bulk of the user interface, and it will be very important in how the user views the system as a whole.

## *5.1 High-Level User Interface Design*

At a high level, the user interface consists of a stylized menu screen, with buttons to link to the scan, inventory, shop, and recipes workspaces.  From these screens, the users can navigate through all of the functionality of our inventory system.

### 5.1.1 Main Menu

The main menu consists of the scan, shop, inventory management, and recipes workspaces as depicted in Figure 2.  The user can switch between workspaces by clicking on the corresponding button or by clicking on the workspace's icon.

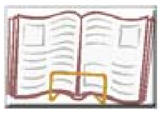
| Workspace | Icon | Functions |
|---|---|---|
| Scan | | Scanning in products from wand, verifying the correctness of scanned items, identifying unknown items |
| Inventory | | Manage accuracy of inventory, add custom codes |
| Shop | | Prepare shopping lists, adding new items, order groceries online |
| Recipes |  | Navigate recipe database, add new recipes |

**Figure 2: Main Menu**

A screen shot illustrating the desired appearance of the layout is shown in Figure 2.  The workspaces, as shown in Figure 3 and Tables Table 9: File Menu to Table 12: Help Menu, appear on top of the graphic of the opened refrigerator.  The purpose of this user interface layout is to organize the GUI according to a typical user's workflow path.  Each workspace corresponds to a standard workflow path: connecting the wand to the computer and uploading the scanned codes (scan), managing inventory and keeping the database synchronized with the actual contents of the user's refrigerator (inventory manager), generating shopping lists and browsing the store catalogue for new items (shop), and searching through a list of recipes for new cooking ideas (recipes). By having the workspaces embedded as pages within the main window, the user is able to seamlessly switch between tasks and the GUI remains uncluttered.

The main menu uses a drop-down menu to allow the user to access all the software functionality using only the keyboard. Hotkeys are provided for the edit functions to provide consistency with other Windows applications.

| **File** | **Edit** | **Options** | **Help** |
|---|---|---|---|
| View Old Shopping List | Cut (X) | Scan | SmartFridge Help |
| Print | Copy (C) | Shop | About SmartFridge |
| Quit | Paste (V) | Manage Inventory | |
| | | Recipe | |
| | | Custom Codes | |
| | | Settings | |

**Figure 3: Pull-down menus**

**Table 9: File Menu**

| Command | Hotkey | Action |
|---|---|---|
| View Old Shopping List | | Invokes the View Old Shopping list window, allowing the user to see what has been ordered previously |
| Print | | Invokes the print dialog box that allows the user to select printer settings. This is used to print a paper copy of the current shopping list |
| Quit | | Exits the program. If any data is unsaved, it will ask for confirmation. |

**Table 10: Edit Menu**

| | | |
|---|---|---|
| Cut | Ctrl + X | This option will be enabled only when the user has highlighted texts within a text edit area. It will remove the highlighted text and add it to the clipboard. |
| Copy | Ctrl + C | This option will be enabled only when the user has highlighted texts within a text edit area. It will add the highlighted text to the clipboard. |
| Paste | Ctrl + V | This option will be enabled only when the user is entering text into a text edit area. It will paste the text from the clipboard into the text area. |

**Table 11: Options Menu**

| | | |
|---|---|---|
| Scan | | Activates the Scan workspace |
| Shop | | Activates the Shop workspace |
| Manage Inventory | | Activates the Manage Inventory workspace |
| Recipes | | Activates the Recipes workspace |
| Settings | | Invokes the Settings dialog, used for adjusting user preferences |
| Custom Codes | | Invokes the Custom Codes dialog, used for entering custom codes and printing custom UPC labels. |

**Table 12: Help Menu**

| | | |
|---|---|---|
| SmartFridge Help | | Opens a help window |
| About | | Invoke the About window that displays the product information and version number |

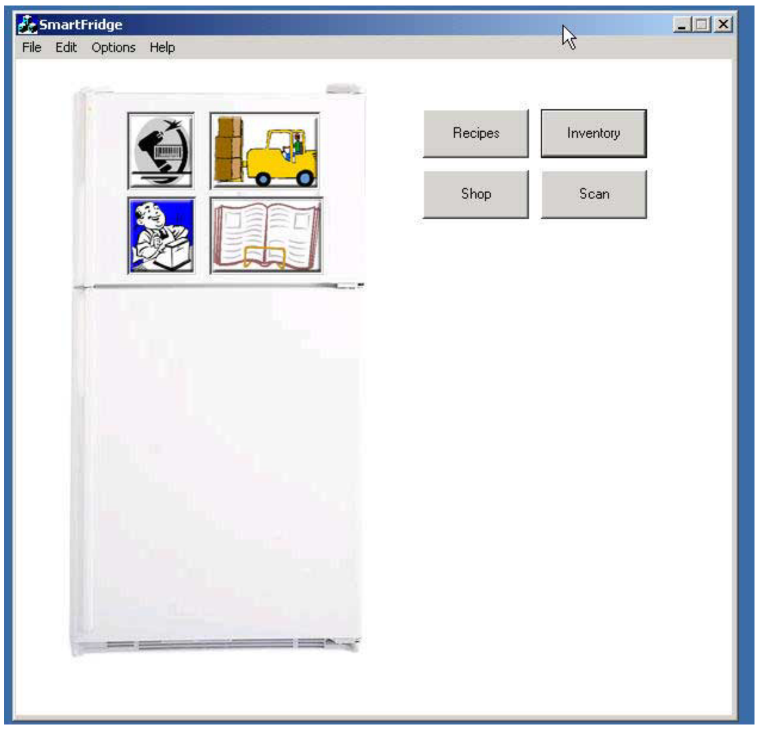


**Figure 4: Main Menu Screen**

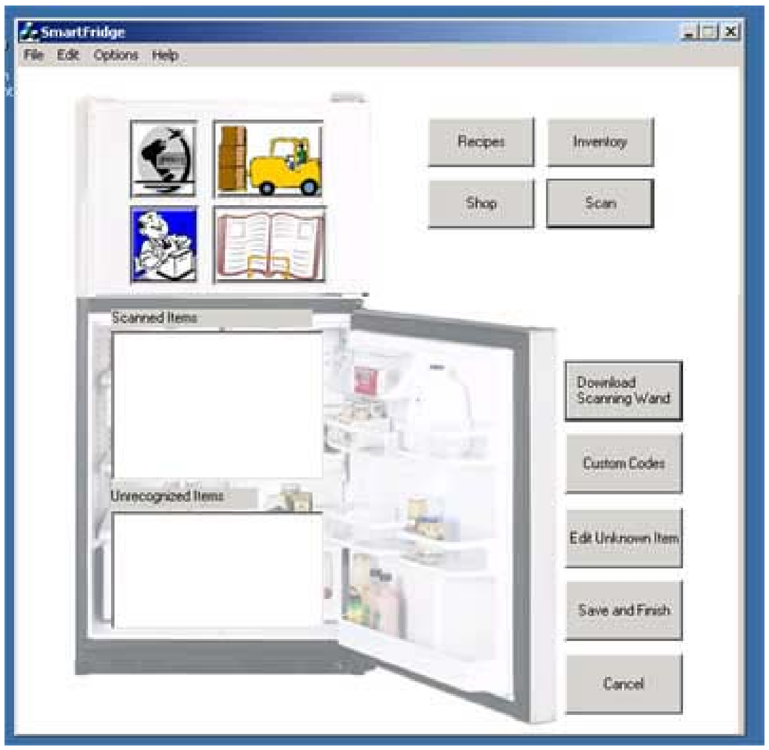


**Figure 5: The scan workspace**

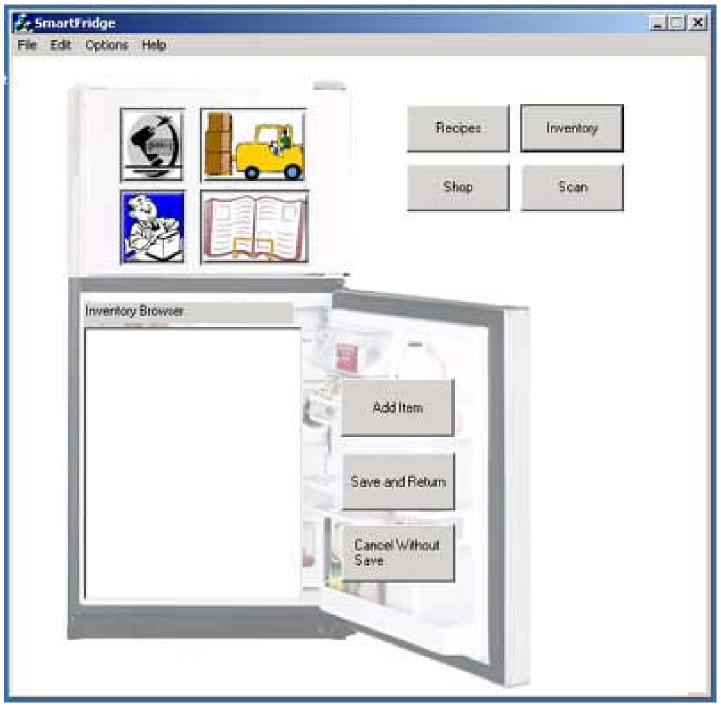


**Figure 6: The Inventory Workspace**

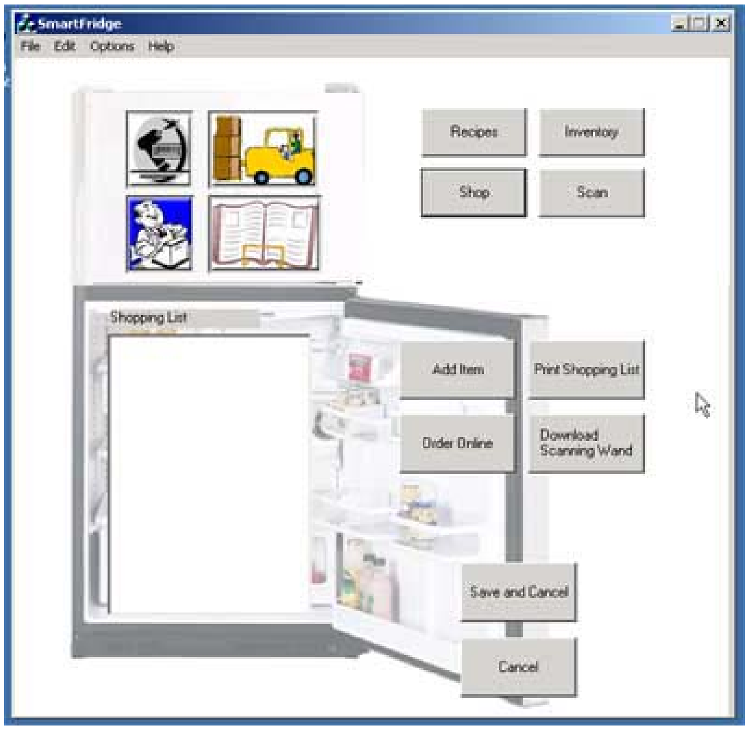


**Figure 7: The Shop Workspace**

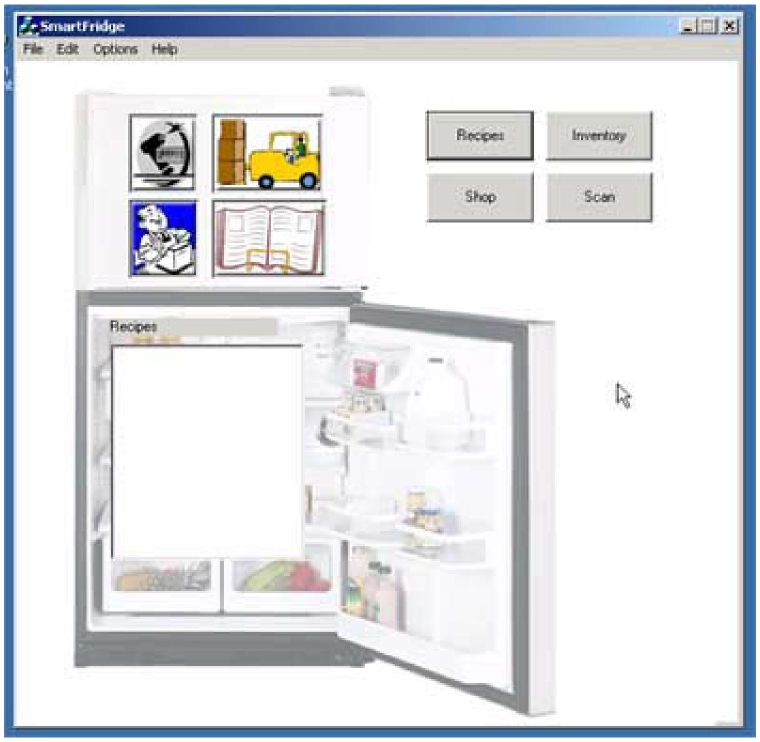


**Figure 8: Recipes Workspace**

Once the user is within a given workspace, there are many buttons available to perform various functions. The functions are outlined in Table 13 Table 16.

**Table 13: Scan Workspace Description**

| Control | Type | Resource ID | Function |
|---|---|---|---|
| Download Scanning Wand | Button | IDC_SCAN_DOWNLOAD | Downloads scanned codes from UPC wand |
| Custom Codes | Button | IDC_SCAN_CUSTOM | Invokes the Custom Codes dialog |
| Edit Unknown Item | Button | IDC_SCAN_EDITUNKNOWN | Invokes the Edit Unknown item dialog |
| Save and Cancel | Button | IDC_SCAN_SAVE | Saves all changes and exits the workspace |
| Cancel Without Save | Button | IDC_SCAN_CANCEL | Exits the workspace without saving changes |

| | | | |
|---|---|---|---|
| Scanned Items List | List | IDC_SCAN_LIST | List of scanned items |
| Unrecognized Items List | List | IDC_SCAN_UNREC | List of items that were scanned but not recognized |

**Table 14: Inventory Workspace**

| Control | Type | Resource ID | Function |
|---|---|---|---|
| Add Item | Button | IDC_INV_ADD | Invokes the Add Item dialog |
| Save and Return | Button | IDC_INV_CANCEL | Saves all changes and exits the workspace |
| Cancel Without Save | Button | IDC_INV_SAVE | Exits the workspace without saving changes |
| Inventory List | List | IDC_INV_LIST | List of items in inventory |

**Table 15: Shop Workspace**

| Control | Type | Resource ID | Function |
|---|---|---|---|
| Add Item | Button | IDC_SHOP_ADD | Invokes the Add Item dialog so the user can search for new items |
| Order Online | Button | IDC_SHOP_ORDER | Invokes the Order Online dialog |
| Print Shopping List | Button | IDC_SHOP_PRINT | Invokes the Print dialog so the user can set printer settings |
| Save and Cancel | Button | IDC_SHOP_SAVE | Saves all changes and exits the workspace |
| Cancel Without Save | Button | IDC_SHOP_CANCEL | Exits the workspace without saving changes |
| Shopping List | List | IDC_SHOP_LIST | List of all items on the current shopping list |

**Table 16: Recipes Workspace**

| Control | Type | Resource ID | Function |
|---|---|---|---|
| Recipe Navigator | Tree | IDC_REC_LIST | Allows the user to navigate the recipes and add recipes and new categories of recipes. |

### 5.1.2 Dialogs

Many simple dialog windows will be used for collecting data from the user. Each dialog will consist of text entry boxes to collect the required data, and OK and Cancel buttons. If the user clicks on cancel, the information will not be recorded. If the user selects OK, the dialog will check the values to make sure they are within specified values and return the user data to the main application window.

## *5.2 Low-Level User Interface Design*

The SmartFridge software application will incorporate the Microsoft Foundation Classes (MFC) included with Visual C++. The framework classes included in this package provide the basic functionality for a Windows application from which the SmartFridge application will be created. The advantages of using the MFC is that it provides a powerful library of pre-built functions that are useful for generating the look-and-feel of a Windows application, and is highly suited to integration with the Access database software we will be using. However, it will be necessary to extensively customize the functionality to suit the SmartFridge application needs, requiring significant low-level coding to extend the pre-built libraries. The various user interface modules and their hierarchy is depicted in Figure 9.

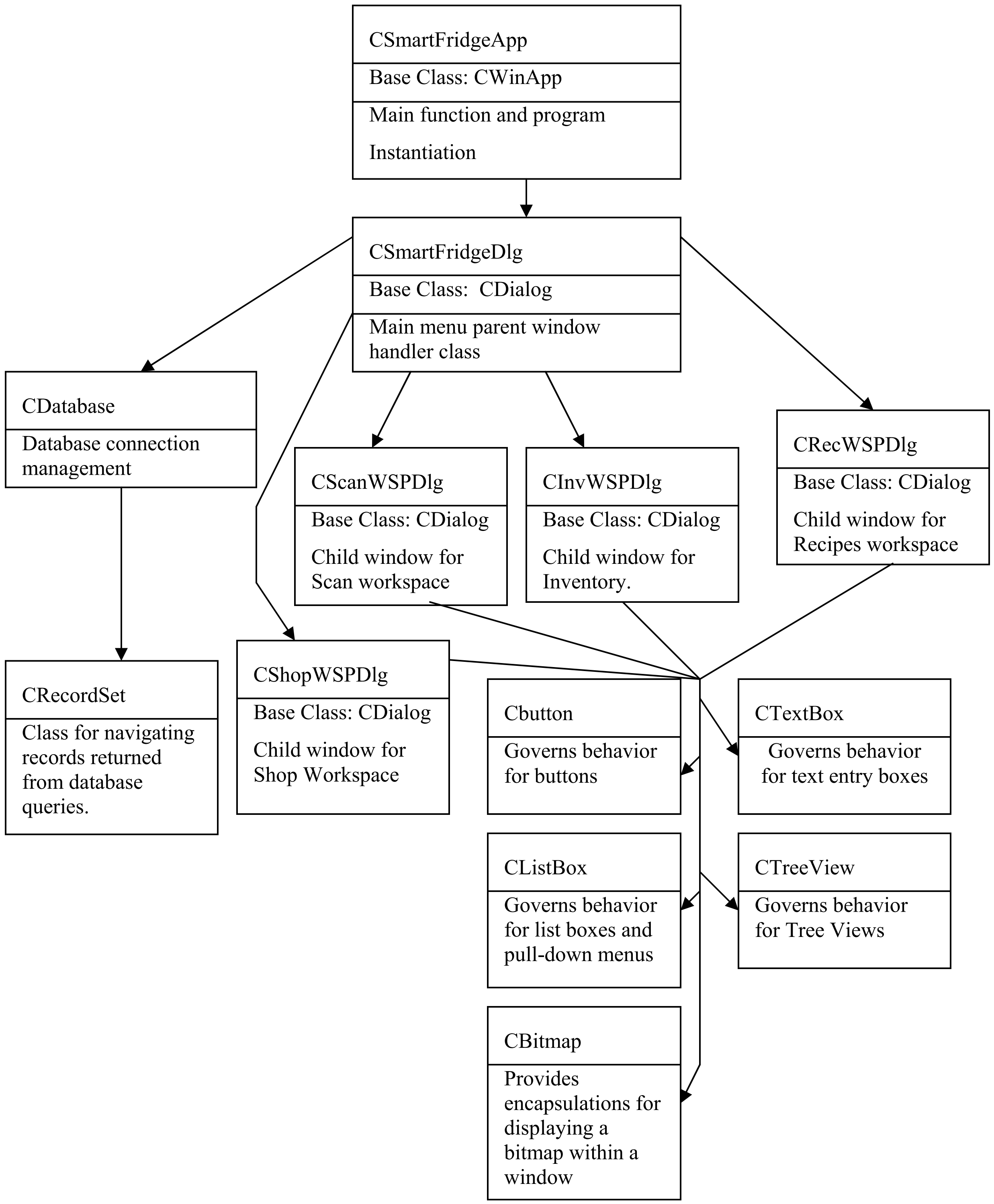

CSmartFridgeApp
Base Class: CWinApp
Main function and program Instantiation
CSmartFridgeDlg
Base Class: CDialog
Main menu parent window handler class
CDatabase
Database connection management
CScanWSPDlg
Base Class: CDialog
Child window for Scan workspace
CInvWSPDlg
Base Class: CDialog
Child window for Inventory.
CRecWSPDlg
Base Class: CDialog
Child window for Recipes workspace
CRecordSet
Class for navigating records returned from database queries.
CShopWSPDlg
Base Class: CDialog
Child window for Shop Workspace
Cbutton
Governs behavior for buttons
CTextBox
Governs behavior for text entry boxes
CListBox
Governs behavior for list boxes and pull-down menus
CTreeView
Governs behavior for Tree Views
CBitmap
Provides encapsulations for displaying a bitmap within a window

**Figure 9: User Interface Module Diagram**

## 5.2.1 Application Characteristics

The software will be implemented as a Dialog-based executable framework constructed using the Visual C++ AppWizard template. The template will allow the Visual C++ ClassWizard to be used to generate the message-handling code necessary for the application to call the correct methods in response to user events such as clicking on a button or selecting an item from the menu. This greatly simplifies the amount of low-level coding required for the creation of a Windows application.

### Main Menu

The main menu will provide a standard menu, status bar, as well as minimize and close buttons. It will display the image of a refrigerator, with picture buttons allowing the user to move between the four major workspaces: scan, inventory management, shop, and recipes. To create this affect it will be necessary to modify the standard CDialog class by creating five subclasses. The class CSmartFridgeDlg will function as the parent window, and will be a modal dialog box with an attached menu, status bar, minimize and maximize button. The four workspaces will each be contained in separate frameless and modeless dialogs that will be embedded in the parent window. This will make them seem like part of the main dialog window. Only the controls will be visible, and the child windows will automatically move with the parent. Each child window can be hidden or shown to create the appearance that the user is flipping between different pages on the main menu, such as on a tabbed property sheet common in many Windows application. The reason we did not use the standard CPropertySheet class is that we wanted to have the ability to customize the size, shape, position, and graphic of the buttons switching between pages, and the CPropertySheet class limits the buttons to appearing as tabs. To create the modeless dialog classes it is necessary to overwrite the ::OnOK, OnCancel, and OnPostNCDestroy methods from the CDialog class.

### Tree & List Views

The Tree views will be derivatives of the CTreeView and CListView classes. The classes will be extended to allow icons and buttons to be displayed along with the entries.

### Printing

Printing will be accomplished by overriding the CDialog::OnPrint method and creating a CPrintDC Device context. The device context will give access to all drawing functions including pixel-by-pixel device control using the CPrintDC::bitBlt method.

### Serial Communications

Serial communications will be implemented using the Win32 API. The Win32 API provides several methods used for interfacing with the comm serial ports:

- **CreateFile**: Opens a communications port.
- **ReadComm**: Reads data from the communications port.
- **WriteComm**: Writes data to the communications port.

- **WaitCommEvent**: Waits for a specified comm port event to occur and activates a specified method.
- **SetCommMask**: Sets the communication port properties.

## *5.3 Database Overview*

Although there is a tremendous amount of programming required to implement the SmartFridge software user interface, the heart of the software application is the database. The database consists of tables that contain a variety of information about the different products and recipes.

### 5.3.1 General Description of Database implementation

The actual database will be constructed using Microsoft Access, and the software will add and retrieve items from the database by sending SQL commands to the Microsoft Jet database engine. Table 17 contains a list of the primary tables used in the SmartFridge Database, and a brief description of the information that each table contains.

**Table 17: Description of Primary Tables for the SmartFridge Database**

| Table Name | Description of Primary Tables |
|---|---|
| ProductLibrary | Contains all product information, identified by barcode number. |
| GrocerProducts | Contains a list of the barcodes for the products that the online grocer stocks. |
| PersonalInventory | Lists the products in the user's virtual inventory, as well as the quantity |

Table 18 contains a list of the Tables used in the various workspaces and a brief description of each.

**Table 18: Description of Workspace Tables for the SmartFridge Database**

| Table Name | Description of Workspace Tables |
|---|---|
| ShoppingList | All items that have been scanned out or added by the user in the last month. When any item on the Shopping List is scanned in, that item is removed from the ShoppingList table. |
| UnknownItems | All unknown items that were scanned in and have not been identified. |
| Recipes | All of the recipes in the SmartFridge Database. Each record in the table contains a single recipe, as well as a unique Recipe Code identifier. |

| | |
|---|---|
| IngredientsList | Contains all of the ingredients and amounts corresponding to each recipe |

In addition to the tables listed above, there will be a number of tables that are used to interpret numerical codes to text descriptions. These tables are described in Table 19: Description of Translation Tables for the SmartFridge Database.

There are a number of advantages to using numerical codes instead of text descriptions. First of all, in most cases the amount of space required to hold a numeric code is less than that required to hold a string. In large databases, saving a few bytes of space in each row of a table makes a tremendous difference in both the size of the database and the efficiency. Secondly, it is much quicker to enter the information if it is only a code – the same words or descriptions do not have to be entered in each field of multiple tables. As well, typos when entering category descriptions would confuse the computer program. Finally, if changes need to be made to the description of a category, the only change that is necessary is in the look up table. The alternative would be to change each record that refers to the category that needs to be changed.

**Table 19: Description of Translation Tables for the SmartFridge Database**

| Table Name | Description of Translation Tables |
|---|---|
| CategoryCode | Contains codes for each category and subcategory, as well as descriptions of the category. Each record will also contain the parent category and the children categories. |
| RecipeCategory | Contains codes for recipe categories and subcategories, as well as a description of the category. |
| UnitsCode | Contains the descriptions of unit types and the corresponding number for each type. |

## 5.3.2 Implementation Decisions

In the design of the database structure, it was necessary to make decisions regarding the method of implementation. This section describes a few of the major decisions made and our motivation for using the method that we chose.

The first implementation decision made was regarding the method of storing temporary data, for example, in the Add/Remove ingredients window. The temporary data could be stored in a separate table, or it could be stored by the program using dynamic variables. If the table method was used, the data would be written to the database right away, and thus the data would not be lost in the event of a computer crash. However, dynamic variables would be less complicated to program, and can be adjusted quickly by the software. Since in the event of a computer crash,

1. the amount of data that would be lost would not require a significant amount of work to re-enter, and
2. the crash would most-likely occur while the data was being entered into the database,

It was decided that for the given requirements, the implementation using dynamic variables is a superior choice.

Another important decision was the implementation of the CategoryCode table. The actual implementation used was based on using linked lists to navigate through categories with the same parent and is discussed in section 5.3.3. An alternative implementation is to use a simple table that consists of a 10 digit CategoryCode field and a Description field. The description field is the same in each implementation.

The CategoryCode field for the alternative CategoryCode table contains a 10 digit number and the information is grouped into five sections of two digits each. The first two MSBs represent the main category, the next two MSBs are the first sub-category, and so forth. If the category code for a group of two bits is 00, then the software knows that the product with this CategoryCode belongs at this level. As an example, suppose you want to put an item into "category one", "sub-category one". The code would be: 0101000000. The next sub-category added to "category one" would be "sub-category two" and items in this category would have a code 0102000000. Using this method of implementing the CategoryCode would allow 99 different main categories, 99 sub-categories for each category, etc… If all possible categories were used, there would be a total of $99^5$ categories, or $9.5 \times 10^9$ categories, much more than is likely to be required.

The 10 digit CategoryCode method has the drawback that the code will take up more room than necessary. A 10 digit decimal number requires 34 bits of storage, or about 4 bytes. The CategoryCode is referenced in a number of different locations, including the ProductLibrary table. However, the Product Library can contain as many as 600000 items. A typical set of categories would probably not contain more than 1000 different categories and sub-categories. Therefore the amount of space required to hold a linked-list CategoryCode is only 10 bits, or just under half the required space to hold the 10 digit CategoryCode.

In the case of a very large database with a lot of categories and sub-categories, the 10 digit CategoryCode might be preferred, as the access time is always the same. The speed of access for the linked-list CategoryCode is slower, as it is necessary for the software to search through many different records to determine the category level.

We decided to implement the CategoryCode table using the linked-list style CategoryCode because of the additional flexibility and speed of access for our expected tree size and database size. As well, it is conceivable that a particular category could have more than 100 sub-categories, and therefore the 10 digit CategoryCode would not be sufficient. Finally, using the linked-list style CategoryCode allows the user to create more than five levels of sub-categories, and therefore requirement 115 from the functional specifications is exceeded.

### 5.3.3 Detailed Description of Database Tables

This section discusses each database table in detail. The fields that each table contains are listed in a data table and a more detailed description of each field is displayed below the data table in bulleted form. The format of each bullet is as follows:

- FieldName (type of data that must be in field, [default = ,] [*primary*]): Description of field

The type of data that must be in each field is as described in the *Comparison or mapping of data types between an Access database and Access* portion of appendix A. The data types that are used are those for Access project (SQL server).

The optional default parameter is to allow the programmers to provide a default value for the field.

An important requirement for each table is that it contains at least one field that uniquely identifies each record. When the table is defined, the field with the unique identifier must be specified and is referred to as the primary key. The field(s) in each table that is (are) bold describe the primary key.

## 5.3.4 Primary Tables

### ProductLibrary Table

The ProductLibrary table contains all of the information about every product the SmartFridge Software can recognize.

**Table 20: ProductLibrary Table**

| ***Barcode*** | *ProductDescription* | *Size* | *UnitsCode* | *Manufacturer* | *CategoryCode* |
|---|---|---|---|---|---|

- Barcode (int, *primary*): The barcode number from the package of the item being scanned.
- ProductDescription (varchar(*n*)): The name of the product as well as a brief description.
- Size (float): size of the product package. Note that various sizes of packages of the same product will have different barcodes.
- UnitsCode (int): The code corresponding to the type of units for the item. Note that the units-code-to-unit translation can be determined by using the UnitsCode table.
- Manufacturer (varchar(*n*)): The manufacturer of the product. The SmartFridge software does not refer to the manufacturer field of the ProductLibrary table, and therefore it does not matter if the value in this field is NULL.
- CategoryCode (int): Contains the code for the category that the product best fits into. See description of the CategoryCode table for more information.

### GrocerProducts Table

The GrocerProducts table lists all of the products that the online grocer associated with Home Gizmos has in stock. Since Home Gizmos has not formed any strategic alliances with online grocers at this time, the GrocerProducts table contains an arbitrary subset of the product library for the purpose of testing and development.

**Table 21: GrocerProducts Table**

| ***Barcode*** | *Price* |
|---|---|

- Barcode (int, *primary*): The barcode of a product that the online grocer stocks.
- Price (money): The price that the online grocer charges for the product.

### PersonalInventory Table

The PersonalInventory table contains a list of all of the products in the user's virtual inventory. To keep track of the date that each product was scanned in, each record of the PersonalInventory table contains the number of instances of a particular item scanned on a particular date. For example, if item number X was scanned on January 1st, the record would be X, 01/01/02, 1. However, if on the same day a second instance of item X was scanned, the record would be updated to X, 01/01/02, 2. If on January 2nd, another instance of item X was scanned in, a new record would be added to the table, and this record would contain X, 02/01/02,1. Using this method, it is possible to track the date that each item in the virtual inventory was scanned in.

For the PersonalInventory table, the primary key is actually a compound primary key that consists of the Barcode field and the Date field. That is, the combination of these two fields must be unique.

**Table 22: PersonalInventory Table**

| ***Barcode*** | ***Date*** | *Quantity* |
|---|---|---|

- Barcode (int, *compound primary*): The barcode of the product scanned in on the date in the date field.
- Date (datetime, *compound primary*): The date that the product in the barcode field was scanned in.
- Quantity (int): The number of items of the type in the barcode field and scanned in on the date in the date field.

## 5.3.5 Workspace Tables

### ShoppingList Table

The ShoppingList table keeps track of the items in the user's shopping list[1]. The shopping list adds items to the ShoppingList table when they are scanned out. If one of the items on the shopping list is scanned in, the SmartFridge software will remove the item from the ShoppingList table, or adjust the quantity if required.

[1] Refer to section 5.4, requirements 69-71 in the Functional Specifications document.

**Table 23: ShoppingList Table**

| ***Barcode*** | *Quantity* | *CheckBoxStatus* | *DateAdded2List* |
|---|---|---|---|

- Barcode (int, *primary*): The products in the user's Shopping List.
- Quantity (int): The quantity of the product identified by the barcode field.
- CheckBoxStatus (bit, default = 1): If the check box next to the item is active for the record, the field value will be 1 and the Print Shopping List button or Order Online button will process the item[2]. Otherwise, the check box is inactive, and the field value will be 0.
- DateAdded2List (datetime): Contains the most recent date that the product with the barcode given in the record was modified.

## UnknownItems Table

The UnknownItems table contains a list of all products scanned in without a barcode recognized by the SmartFridge software to be in the Product Library[3].

**Table 24: UnknownItems Table**

| ***Barcode*** | *Date* |
|---|---|

- Barcode (int, *primary*): The barcodes of the unknown products.
- Date (datetime): The date that the unknown product was most recently scanned in. It is not necessary to keep track of the older dates that the unknown item was scanned as a more recent date is more useful in helping the user decide what the unknown item might be.

## Recipes Table

The Recipes table stores all of the recipes in the SmartFridge database. Each record corresponds to a different recipe. To find the ingredients and amounts required for each recipe, it is necessary to look them up using the IngredientsList table.

**Table 25: Recipes Table**

| ***RecipeCode*** | *RecipeName* | *RecipeDescription* | *Directions* |
|---|---|---|---|
| *AdditionalIngredients* | *RecipeCategory* | | |

- RecipeCode (int, *primary*): A code representing a particular recipe.

[2] Refer to section 5.4, requirements 70, 73-74 in the Functional Specifications document.
[3] Refer to section 5.3.3 in the Functional Specifications document.

- RecipeName (varchar(*n*)): The text name of the recipe.
- RecipeDescription (varchar(*n*)): A text description of the recipe.
- Directions (text): The text directions of how to make the recipe.
- AdditionalIngredients (text): A text description of the ingredients required for the recipe that are not in the Product Library. The ingredients are comma delimited. Ingredients appearing in the AdditionalIngredients field are not searchable by the database.
- RecipeCategory (int): The RecipeCategory contains a number that represents the category that the recipe would fall into. The recipe category can be decoded from the number using the RecipeCategory table.

### IngredientsList Table

The IngredientsList Table associates the ingredients in a recipe with the appropriate Recipe Code. By having a separate table for Recipe/Ingredients translation, it is possible to avoid having the information in the Recipes table repeated unnecessarily. Space is saved, and a more efficient design results.

**Table 26: IngredientsList Table**

| ***RecipeCode*** | ***Ingredient*** | *Amount* | *Units* |
|---|---|---|---|

- RecipeCode (int, *compound primary*): The code corresponding to the Recipe that the ingredient in the Ingredient field belongs to.
- Ingredient (int, *compound primary*): Contains the Barcode for the ingredient that is in the recipe.
- Amount (float): Contains the amount of the ingredient contained in the same record.
- Units (int): Contains the UnitsCode of the units of measurement for the record.

## 5.3.6 Translation Tables

### CategoryCode Table

The CategoryCode table is used to determine the category that a product belongs to and to organize the categories into major and sub-categories. Each category, be it major or minor, is listed in the CategoryCode table as its own record.

To list out the categories in a tree organization form, each category with the ParentCode field as Null is listed out for the main categories. Next, for the first main category, the record with the CategoryCode from the OldestChild field is displayed. The next category at the same level is found by looking in the NextOldestSibling field of the current record. Once a category is reached that has a Null value in the NextOldestSibling, the software knows that the last sub category has been reached. Sub-sub-categories are displayed in a similar manner, and the entire procedure is repeated for the other main categories.

The overall effect of this structure is to have a tree-type hierarchy that connects sub-categories using single linked lists.

**Table 27: CategoryCode Table**

| ***CategoryCode*** | *Description* | *ParentCode* | *OldestChild* | *NextOldestSibling* |
|---|---|---|---|---|

- CategoryCode (int, *primary*): Used to uniquely identify each category, no matter its level in the hierarchy.
- Description (varchar(*n*)): A brief text description/title for the category.
- ParentCode (int): Used to look up the CategoryCode of the parent category. If this field is Null, then the category belongs to the top most level of the hierarchy.
- OldestChild (int): Gives a starting point for listing the children categories of the category associated with the current record. To list the children, you go to the oldest child of the category and travel through the children from oldest to youngest by referring to the NextOldestSibling field. If the value of the OldestChild field is Null, then there are no children categories associated with the current record.
- NextOldestSibling (int): Used for listing the sub-categories that share a common parent and that are at the same level. If the NextOldestSibling field of a record is Null, then there are no more sub-categories at the same level.

## RecipeCategory Table

The RecipeCategory table has the same structure as the Category Code table; however the categories are more suitable for sorting recipes.

**Table 28: RecipeCategory Table**

| ***RecipeCategory*** | *CategoryName* | *ParentCode* | *OldestChild* | *NextOldestSibling* |
|---|---|---|---|---|

- RecipeCategory (int, *primary*): The code representing a particular category of recipe.
- CategoryName (varchar(*n*)): A text description of the recipe category.
- ParentCode (int): Used to look up the RecipeCode of the parent category. If this field is Null, then the category belongs to the top most level of the hierarchy
- OldestChild (int): Gives a starting point for listing the children categories of the Recipe category associated with the current record. If the value of the OldestChild field is Null, then there are no children recipe categories associated with the current record.
- NextOldestSibling (int): Used for listing the sub-categories that share a common parent and that are at the same level. If the NextOldestSibling field of a record is Null, then there are no more sub-categories of recipes at the same level

### UnitsCode Table

The UnitsCode table maps the UnitsCode to the text description of the unit.

**Table 29: UnitsCode Table**

| ***UnitsCode*** | *UnitDescription* |
|---|---|

- UnitsCode (int, *primary*): The code representing the units
- UnitDescription (varchar(*n*)): A text description of the unit

## *5.4 Accessing the Database using SQL*

Once the structure of the database is established, it is necessary to create a method of accessing the tables to access, add and remove information. The method for meeting this requirement is to use SQL, structured query language programming.

As mentioned in Section 5.2, the SmartFridge user interface is to be created using Microsoft Visual C++ 6.0. Using Visual C++, it is straightforward to create SQL commands using the Microsoft Foundation Classes *Cdatabase* and *CRecordSet*.

The SQL commands that we will be using belong to the SQL subset of languages called DML (Data Manipulation Language). DDL, Data Definition Language, will not be used as the creation of tables is to be programmed using Microsoft Access. DML is used to view records and fields from specific tables, as well as to delete and create records or fields.

The most common SQL command we will be using is the SELECT command. The basic structure of the SELECT command is as follows:

SELECT FieldNames AS alias FROM TableName WHERE condition ;

Using this command and several variants, it is possible to view any information that may be required from the database.

# 6 Conclusion

In this document, we have outlined many specific components of the design that have already been decided. From this point, we can determine what is left to be worked out. We have already chosen and obtained all of our important components, as well as begun work on the GUI. The integration process has begun to some extent on the hardware side, where we have interfaced the scanning wand to the microprocessor. Now we need to improve the reliability of our scans, and work out our serial interface. The next step would be to move our components off the evaluation board, and onto a breadboard, and finally a printed circuit board. On the software side, the database is nearly complete, and the GUI has been started. We feel confident that we will be able to complete a working prototype by December.

# A Appendix: Data Types used by Microsoft Access

This appendix contains information on Data Types used by Microsoft Access.

Firstly, a modified list of descriptions of the Microsoft Access Data Types is displayed as taken from *Aitken, Peter, Jennifer Fulton, Sue Plumley, Faithe Wempen "Microsoft Office Professional: 6 in 1"QUE Publishing: 1997. Indiana, USA,* page 606.

**Data Types**

| | |
|---|---|
| **Text** | Plain, ordinary typed text, which can include numbers, letters, and symbols. A Text field can contain up to 255 characters. |
| **Memo** | More plain, ordinary text, with a maximum field length of 64000 characters. |
| **Number** | A plain, ordinary number (not a currency value or a date). Access won't allow any text in a Number field. |
| **Date/Time** | Just that – a date or a time. |
| **Currency** | A number formatted as an amount of money. |
| **AutoNumber** | A number that Access automatically fills in for each consecutive record. |
| **Yes/No** | The answer to a true/false question. It can contain either of two values, which might be Yes or No, True or False, On or Off. |

**Formatting Options**

| | |
|---|---|
| **Field Size** | The maximum number of characters a user can input in that field. |
| **Default Value** | If a field is usually going to contain a certain value, you can enter it here to save time. It will always appear in a new record, and you can type over it in the rare instances when it doesn't apply. |

---

Secondly, a comparison of the data types used by Access database and Access project is displayed. This comparison is taken from the Microsoft Access XP help wizard.

Comparison or mapping of data types between an Access database and Access project

The following table compares data types between a Microsoft Access database and a Microsoft Access project.

| Microsoft Access data type | SQL Server data type |
|---|---|
| Yes/No | bit |

| | |
|---|---|
| Number (Byte) | tinyint |
| Number (Integer) | smallint |
| Number (Long Integer) | int |
| Number (Single) | real |
| (no equivalent) | bigint |
| Number (Double) | float |
| Currency | money<br>smallmoney |
| Decimal/numeric | decimal<br>numeric |
| Date/Time | datetime<br>smalldatetime |
| AutoNumber (Increment) | int (with the **Identity** property defined) |
| Text (n) | varchar(n)<br>nvarchar(n) |
| Memo | text |
| OLE Object | image |
| Replication ID (also called globally unique identifier (GUID)) | uniqueidentifier (SQL Server 7.0 or later) |

| | |
|---|---|
| Hyperlink | char, nchar, varchar, nvarchar (With the **Hyperlink** property set to Yes) |
| (no equivalent) | varbinary |
| (no equivalent) | smallint |
| (no equivalent) | timestamp |
| (no equivalent) | char<br><br>nchar |
| (no equivalent) | sql_variant |
| (no equivalent) | user-defined |

# B Appendix: Wand Schematic

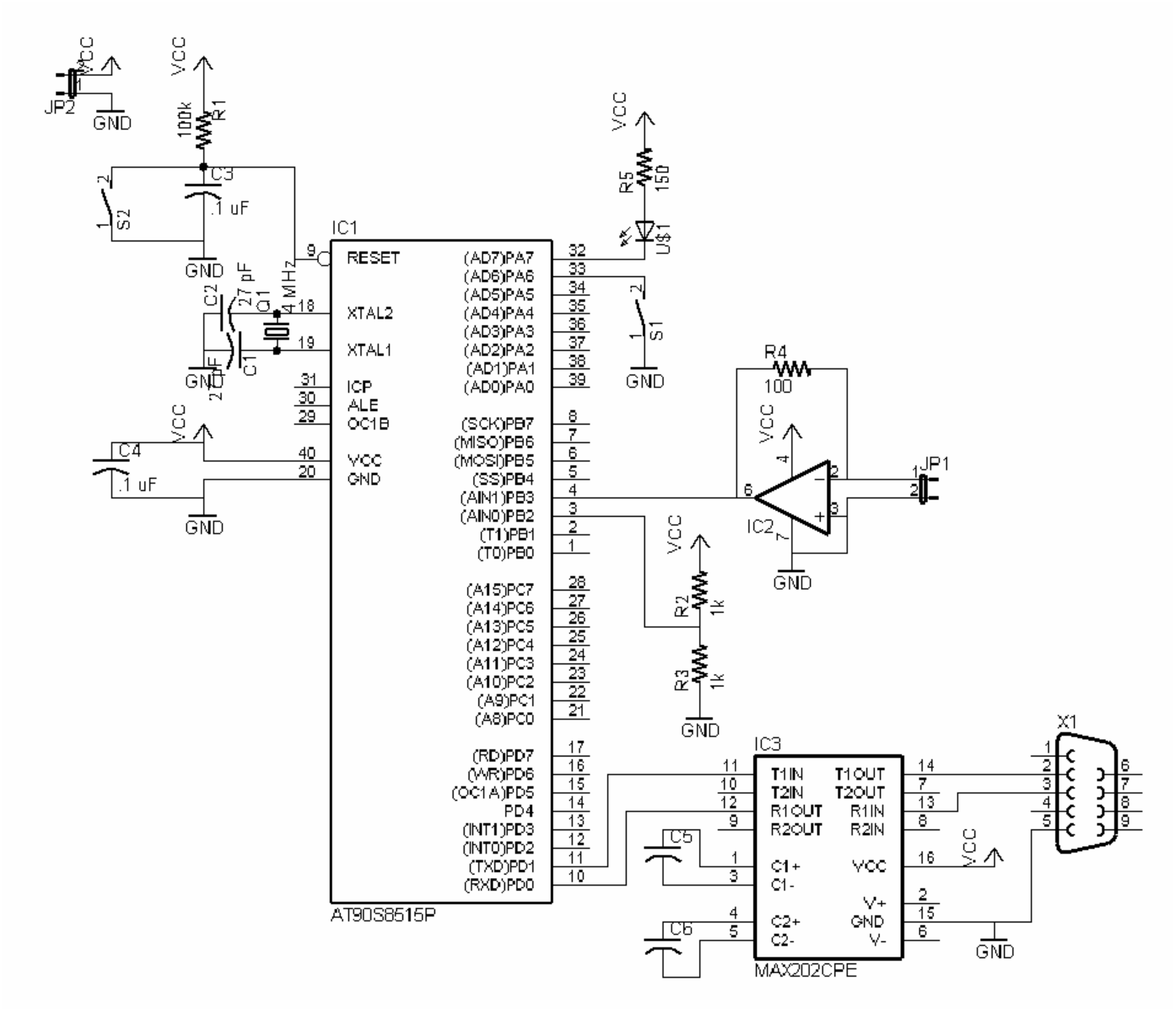


**Figure 10: Hardware Schematic**

## References

| | | |
|---|---|---|
| **Atmel** | Micro-controller manufacturer | http://www.atmel.com |
| **GNU GCC** | C Compiler | http://gcc.gnu.org |
| **Tysso** | Barcode scanner manufacturer | http://tysso.com |
| **Agilent** | Barcode lens/sensor manufacturer | http://agilent.com |
| **Quick.com** | Online Grocery website | http://quick.com |
| **David Vanhorn** | Information on bar codes | http://www.dvanhorn.org |

## Additional References :

Adams, Russ *"Bar Code FAQ page"* http://www.barcode-1.net/pub/russadam/faq.html Adams Communication

Aitken, Peter, Jennifer Fulton, Sue Plumley, Faithe Wempen *"Microsoft Office Professional: 6 in* 1"QUE Publishing: 1997. Indiana, USA

*"Decode the Barcode*" http://www.lascofittings.com/BarCode-EDI/Decode.htm Last edited September 24, 2001

*"How UPC bar Codes wor*k" http://www.howstuffworks.com/upc1.htm

*"ID numbers and Bar Code*s" http://www.uc-councilorg/main/ID_Numbers_and_Bar_Codes.html *Uniform Code Council Inc*

*"Microsoft Visual C++ 6.0: MFC Library Reference Part* 1" Microsoft Press:1998. Redmond, Washington.

*"Microsoft Office 97 Resource Kit",* Microsoft Press: 1997, Redmond, Washington.

Microsoft Access XP help wizard

Novalis, Susann *"Access 2000 VBA Handbook"* Sybex, San Fransisco, WA. Copyright 2000

*"Open Directory: Computers: Software: Barcode"* http://dmoz.org/Computers/Software/Bar_Code/

Ramalho, Jose A "Learn SQL in Three Days", 2001, Wordware Publishing, Inc. Plano, Texas

*"UPCDB.com"* http://www.rnrcomputing.com/upc/ Visited September, 2002 (contents have changed since then)