

INTERIM REPORT

YardSale The Open Source Point of Sale Solution



A.S. Logic Systems Co.
Jesse Lovelace
Adam Parrish
Mike Swigon
Jay Johnston
John Lamb
Cameron Watts

April 7, 2004

Contents

1	Requirements De nition	3
1.1	Introduction	3
1.2	Functional Requirements	3
1.2.1	Information Management	4
1.2.2	Employee Data	4
1.2.3	Transactions	5
1.2.4	Reporting	5
1.2.5	User Level Access Rights	5
1.3	Non Functional Requirements	6
1.3.1	Cross Platform	6
1.3.2	Cross Architecture	6
1.4	System Constraints	6
1.4.1	Interfaces	6
1.5	External Dependencies and Interfaces	7
1.6	Preliminary Design	7
1.6.1	Major Modules	7
1.6.2	System Architecture	8
1.7	Preliminary Project Task Plan	8
1.7.1	Milestones	8
1.7.2	Team Member Roles and Responsibilities	9
2	Design	10
2.1	YardSale Database Design	10
2.1.1	Customer Management	12
2.1.2	Inventory Management	13
2.1.3	Transaction Handling	15
2.1.4	Employee Management	16
2.1.5	Security	18
2.2	YardSale Client Application Design	19
2.2.1	Main Menu	19
2.2.2	Sales Screen	20
2.2.3	Inventory Screen	21
2.2.4	Employee Management Screen	22
2.2.5	On-screen Calculator	23
3	Implementation	24
3.1	Database Level Implementation	24
3.1.1	Customer Management	24
3.1.2	Inventory Management	24

3.1.3	Transaction Handling	26
3.1.4	Employee Management	27
3.2	Application Level Implementation	28
3.2.1	YardCalc	28
3.2.2	YardDatabase	28
3.2.3	YardDBType	29
3.2.4	YardEmployee	29
3.2.5	YardInventory	29
3.2.6	YardException	29
3.2.7	YardLog	29
3.2.8	YardLogin	30
3.2.9	YardMain	30
3.2.10	YardSale	30
3.2.11	YardSaleScreen	30
3.2.12	YardSplash	30
4	Test Plan	31
4.1	Passive Testing	31
4.2	Active Testing	31
4.2.1	Testing Mains	31
4.2.2	Warm-Body Testing	31
4.2.3	Verbose Program Information	32

Chapter 1

Requirements Definition

1.1 Introduction

YardSale is an open source Point of Sale system that is being developed by A.S Logic Systems. Current implementations of Point of Sale systems are fraught with a number of issues including often being extremely overpriced, difficult to administer, dated in their functionality, and lacking necessary operations.

The power of an open source system such as YardSale is the open way in which it is designed. A user of the YardSale system can take off the shelf computer hardware (in many cases older hardware will perform just as well) and create a point of sale with minimal configuration. In addition, supporting new hardware is trivial since the YardSale interface specification is in no way hindered by restrictive close-source licenses.

The name "YardSale" was conceived to emphasize the versatility and simplicity of the system. The parts of a YardSale point of sale can be older, used hardware and the underlying software can be completely free.

Being that YardSale is an open source project the initial market will target small, locally owned retail stores. This type of market will allow for extensive on-site research, as many of these businesses should be willing to work with us to improve on the functionality of their existing POS system. In addition, because A.S. Logic has decided to take the open source route, the software itself will be freely available to all who wish to use it, the only expense that will come into play is if the company wishes to enlist our services in setup, troubleshooting, support, expansion, or customization.

However, we believe that YardSale can be easily used by those who do not wish to purchase support contracts due to the clear documentation and well-written interfaces.

1.2 Functional Requirements

This section entails all of the minimal requirements set by the AS Logic Systems development team for the Open Source Point of Sale, hereafter referred to as YardSale.

YardSale should accomplish the following Point of Sale operations:

Manage Inventory

- { Add a product to the inventory
- { Edit an existing inventory item
- { Remove an inventory item
- { Associate attributes such as price, quantity, and tax with an inventory item
- { Search for an inventory item by various criteria
- { Add an inventory item to a transaction (see Transaction)

Manage Customers

- { Add and remove customers
- { Edit customer information such as phone number and address
- { Associate a customer with a transaction (see Transaction)

Manage Employees

- { Add and remove employees
- { Associate a level with employees such as "manager" and "sales associate"

Perform Point of Sale Transactions

- { Allow an "seller/employee" to sell and inventory item to a "buyer/customer"
- { Allow a customer to return an item to the seller
- { Calculate the tax on an item being sold

Each of these tasks is described in depth in their corresponding sub sections. The functionality outlined above is designed to be as extensible as possible; with the correct implementation of these requirements the YardSale System will support the expansion of functionality with great ease.

1.2.1 Information Management

YardSale will store information about all aspects of program operation data. The data that is stored can be broken down into four main subsections

Employee Data

Customer Data

Inventory Data

Sales Data

1.2.2 Employee Data

For YardSale to properly manage

YardSale will store information about all aspects of program operation data. This is inclusive of Employee, Customer, Inventory, and Sales data. Employees shall have all pertinent contact information such as Name, Address, Phone Numbers, and Email address as well as an Employee Number and possibly a Social Security number saved as a record. Customer will have the same contact information stored as well. Inventory Items will have a Name and SKU number associated with every record, and other pertinent information about Items such as their cost, resale value, and tax rate will also be stored. Sale transactions shall be saved by referencing the primary key of each employee, customer, and inventory item sold during a transaction.

Inventory Management

Customer Management

Employee Management

Among the more specific functionality of each of these sections will be to allow the user to retrieve any item stored in either inventory, customer, or employee tables given a search criteria.

1.2.3 Transactions

Transactions are the key to any point of sale system. The obvious function that a point of sale should perform is selling an item. However, selling an item is not simply adding up a price. Most sales require information about what is being sold, who is selling it, and to whom the items are being sold. These attributes are contained in a YardSale transaction.

YardSale will create a transaction which contains one or more items either being sold or returned. Each transaction will have an employee and a customer associated with it. Since tax is extremely important to any transaction, tax must be added to each item since different items require different taxes as mandated by local, state, and national government.

Once a transaction is created, it must be stored by the information management system so that a record of the transaction can be recalled for returns and reports.

1.2.4 Reporting

There will be a wide variety of reporting features available also as a result of the database backend. The following basic reporting functionality will be available in the first iteration of YardSale.

- Payroll for Employees

- Top Sales for Inventory Items

- Top Sales for Employees

- Top Sales for Customer

- Revenue Reporting given any time frame

- Hourly Employee Log Reporting

All of the reports will be output to a pdf via L^AT_EXtype setting. This will provide ease of portability and a wide range of flexibility in reporting.

1.2.5 User Level Access Rights

YardSale has three different levels of user interactivity. There are Managers, Sales Associates, and Administrative users.

Sales Associate Functionality

Sales associates are likely going to be the primary users of the YardSale system. However they will have a limited set of functionality when they login. Their privileges will be limited to basic customer management and transaction processing. They have no real need for full inventory capabilities nor do they need the ability to manage or browse employee information.

When a sales associate logs into YardSale they will see the following options:

- Transaction Processing

- Customer Management

Manager Functionality

The managers primary function in YardSale is to make sure that all of the inventory items, customers, and employees are correctly maintained. They are also the only primary user on the system allowed to run reporting functions.

When a manager logs in to YardSale they will have a full set of functionality available to them from the main screen inclusive of the following:

- Inventory Management
- Transaction Processing
- Customer Management
- Employee Management
- Reporting

Administrative Functionality

The administrative user is basically a built in manager user. This user will have full rights to the system and the password will be able to unlock other accounts in the system that are locked due to loss of password. The administrative user will likely never be used in day to day use of YardSale, however it is provided as a safety to always allow the access of encrypted data in case of total loss of management capability.

1.3 Non Functional Requirements

1.3.1 Cross Platform

The YardSale software is to be developed using the wxWindows libraries for C++, which creates GUIs (Graphical User Interfaces) that are platform independent. The YardSale software will be able to run on any major operating system on today's market.

1.3.2 Cross Architecture

The main purpose of the YardSale software is to be very independent of all architectural aspects of the system, using minimal system resources. The idea is to be able to use any system to run the software appropriately.

1.4 System Constraints

1.4.1 Interfaces

The YardSale system must support the use of several peripheral interfaces to aid in the completion of POS transactions and other functionality outlined previously. The interfaces are outlined as follows:

Cash Drawers The system must support the use of a cash drawer for storage of money exchanged during transactions. It shall pop open upon the completion of a transaction or when prompted by the user.

Magnetic Card Scanners The system must support the use of a magnetic card scanner for automatic input of information. These scanners must correctly input information stored on both credit cards and employee access cards.

Barcode Scanners The system must support the use of a barcode scanner for automatic input of items for inventory and transaction processing.

Receipt Printers The system must support the use of a receipt printer. The printers must print specified information at the conclusion of each transaction and will also be used with the reporting functionality.

TouchScreen Monitor The system must support the use of TouchScreen monitors. The monitor shall act as both input and output devices for the system. The monitors must correctly read the commands given and relay them to the system.

1.5 External Dependencies and Interfaces

1.6 Preliminary Design

1.6.1 Major Modules

Database

The Database module works as the translator between the user interface and the database. It converts calls made by the GUI into SQL queries to be sent to the database. When the query results are returned from the database, the Database converts these items to a Database Type (discussed in next section) to be read by the GUI. OTL Libraries will be used in conjunction with ODBC to provide database connectivity with the user interface. OTL is a cross platform library for ODBC.

Database Types

The Database Types module is the superclass for all information that may be sent to the GUI from the database. It contains function calls for each of the following types:

Inventory Type: contains variables for all possible inventory management items stored in the database. Coordinates with the Inventory Management view in the GUI, which is also used by Transactions.

Customer Type: contains variables for all possible customer management items stored in the database. Coordinates with the Customer view in the GUI, which is also used by Transactions.

Employee Type: contains variables for all possible employee management items stored in the database. Coordinates with the Employee Management view in the GUI.

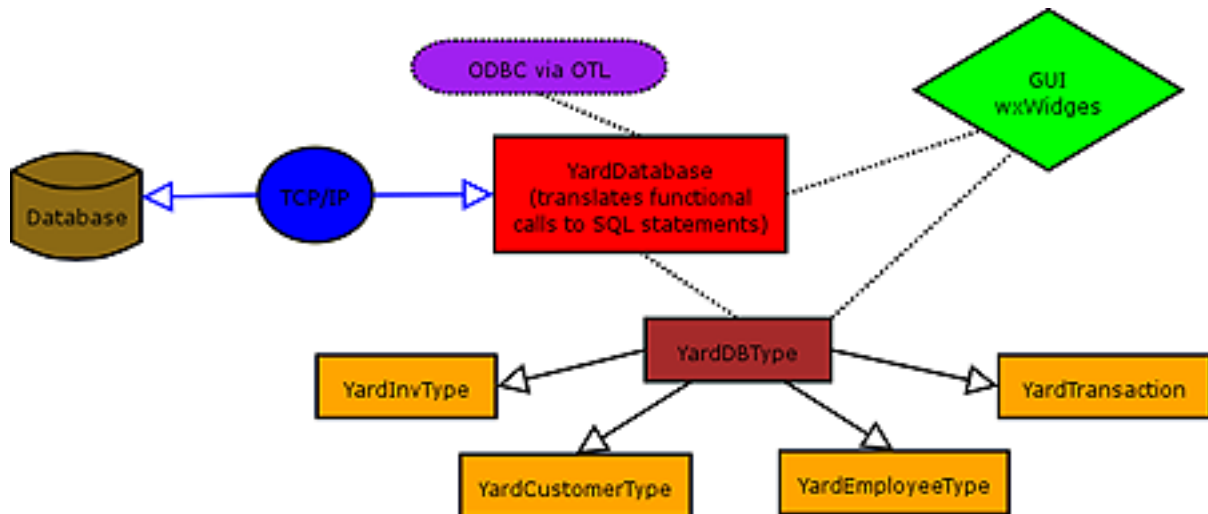
Yard Transaction: contains variables for all possible transaction items stored in the database. Coordinates with the Transaction view in the GUI.

GUI wxWindows

The GUI wxWindows module defines all interfaces used by the GUI. It contains functions derived from the wxWindows libraries for C++. It also interacts with the Database module to display information sent from the database.

1.6.2 System Architecture

The figure shown below outlines the class dependencies and hierarchy of the modules described in section 1.5.1.



1.7 Preliminary Project Task Plan

1.7.1 Milestones

2004-03-05 FEATURE FREEZE - deadline for adding functional features to the design of the POS.

2004-03-07 ITERATION 1 - all requirements for first iteration to be completed and documented in the interim report.

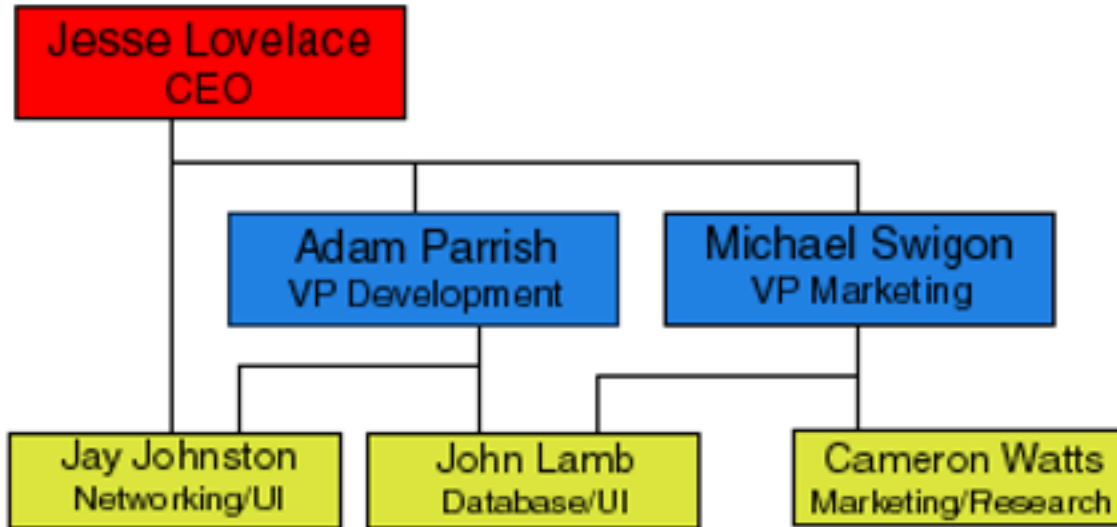
2004-04-01 ITERATION 2 - all secondary requirements to be completed for the second iteration.

2004-04-17 CODE FREEZE - deadline for beginning the coding of features; those not already implemented will be removed; testing only beyond this point

2004-04-27 FINAL DELIVERABLE - completion of prototype for presentation to the CSC department at the Posters and Pies event.

1.7.2 Team Member Roles and Responsibilities

The following figure displays a graphical hierarchy of the team's roles:



The team roles are defined as follows:

Jesse Lovelace As the CEO of A.S. Logic Systems, Jesse is in charge of all aspects of the project. Though he works very closely with both VPs, Jesse is the ultimate decision maker for the team. In addition to his roles as CEO, Jesse is also in charge of the design and implementation of the User Interface and all security aspects.

Adam Parrish Adam is the Vice President in charge of Development at A.S. Logic Systems. He works closely with both the CEO and VP of Marketing to see that YardSale's implementation is both correct and timely. In addition to these company roles, Adam is also the lead Database Programmer; seeing to it that the database is functioning properly and creating the SQL scripts for populating and querying the database.

Mike Swigon Mike is the Vice President of Marketing at A.S. Logic Systems. He works closely with both Adam and Jesse to both design the entire system and to insure that its implementation is correct. In addition to these responsibilities, Mike works with Adam in database setup and creation of SQL scripts for populating and querying the database.

John Lamb John's responsibilities fall primarily in interfacing with the database. He works closely with Jesse to develop a UI that can correctly and securely communicate information to and from the database; also implementing the SQL statements developed by Mike and Adam.

Jay Johnston Jay's responsibilities fall primarily in creation of the UI and networking the system during setup. He works closely with Jesse to develop modules for required functionality of the interface.

Cameron Watts Cameron's responsibilities fall primarily on marketing and system research. He works closely with the design group to ensure YardSale's functionality is top-of-the-line and user friendly.

Chapter 2

Design

The following sections are related to the design of YardSale in regards to the database level design as well as the client application level design. The first section provides a high level explanation of the database architecture using UML diagrams. The following section explains design related issues of the YardSale client program.

2.1 YardSale Database Design

The YardSale database was designed with security and efficiency in mind. The goal was, as it is in any database, to separate unrelated data and key similar items together with table relations. The global database model is seen in the diagram on the following page. Although it is not necessarily easy to read the table relations can be seen, and table specifics like field values, indexes, and foreign and primary keys can be seen in the following table diagrams.

2.1.1 Customer Management

Since customer management is basically a simple task it is maintained in only one table. There is no real need to have their data stored accross many di erent tables since all of the information is just personal data.

Customer Table

Customer_Table
+CUST_Account_Number: AutoIncrement INT
+CUST_First_Name: varchar(25)
+CUST_Middle_Name: varchar(25)
+CUST_Last_Name: varchar(50)
+CUST_Address: TEXT
+CUST_Phone_Number: varchar(20)
+CUST_City: varchar(50)
+CUST_Zip: varchar(12)
+CUST_Credit_Card_Number: varchar(20)
+CUST_CC_Exp_Date: varchar(6)
+CUST_Name_On_CC: TEXT
+CUST_Signature: TEXT
+CUST_Photo: TEXT
+Primary Key(CUST_Account_Number)

The above UML diagram shows the layout of the customer table. The object of this table is to manage customer personal information for later reference in transactions. By allowing this associativity YardSale will later be able to formulate reports on how much each of its customers spend for example. It also provides a rather in depth directory of all of a business's clients for use in any form they see t.

The data stored begins with the customer account number which is just an arbitrarily assigned number that is managed by the database management system. It is also the primary key for the table and is therefore unique. First, Middle, and Last names are all stored in seperate elds of their own, as is the Address information. Credit information can also be stored about users but is optional, and is also planned to be stored in the eld as an encrypted string of data so that underprivileged users can not access it. Two interesting features about this table is the ability to store a link to a photograph and signature for each customer. That way it will aid in positively identifying a customer when they are checking out with check or credit.

2.1.2 Inventory Management

YardSale's Inventory Management scheme spans over three tables. The main table that stores actual inventory description information is the Inventory Table itself. The other two tables are used to support the main table. They are the Tax table and the Vendor Table. The tax table is basically a storage area for different tax types, and the Vendor Table is a storage table for Vendor information or Inventory Supplier information.

Inventory Table

Inventory_Table
+INV_SKU_Number: AutoIncrement INT +INV_Bar_Code_Number: varchar(30) +INV_Item_Description: TEXT +INV_Item_Department: varchar(30) +INV_Quantity_On_Hand: INT +INV_Quantity_On_Order: INT +INV_Reorder_Level: INT +INV_Reorder_Quantity: INT +INV_Item_Type: varchar(20) +INV_Item_Weight: FLOAT +INV_Tax_Type: REFERENCES TAX_Tax_ID FLOAT +INV_REF_Vendor_ID: REFERENCES Vendor_Table.Vendor_ID INT +INV_Retail_Price: MONEY +INV_Wholesale_Price: MONEY +INV_Bulk_Price: TEXT +INV_Date_Last_Received: DATETIME +INV_Weight_Pounds: WEIGHT +INV_Oversized_Flag: Boolean +INV_Ship_By_Freight: Boolean +PrimaryKey(INV_SKU_Number)

This table being the main Inventory storage structure contains all information needed about any inventory item. The record primary key is the SKU number which is a user defined number or character string. These are often used in small businesses to internally key their inventory. Manufacturers will often have a bar code associated with each item they produce as well so there is a field available for that as well. Each item can be briefly described, and associated with a department for further subcategorizing. The number of any particular item is maintained as well as how many of the item are on order. There is a field for storing the number at which an item should be reordered, and also how many to reorder at that time. There are varying description fields such as item type and weight as well. Three pricing fields are supplied. The first two are statically maintained as retail and wholesale price. The third price type varies on the number of items being purchased. This field is maintained in XML format so that as many different pricing levels as are needed can be defined. When an item is received it updates the field corresponding to last received. Some items are oddly shaped or are overly heavy, either of these two options could cause the oversized flag to be set, and if the oversized flag is set a ship by freight option would also be set, but they are mutually exclusive and the ship by freight can be set without the oversized flag being set.

Tax Table

Tax_Table
+Tax_ID: INT Autoincrement
+TAX_Name: varchar(20)
+TAX_Percent: FLOAT
+PrimaryKey(Tax_ID)

The tax table is just a support table for the inventory items. It is referenced by ID depending on the desired taxing an item should have. Using a table allows for user definable tax types, and allows the database to be flexible to tax changes. All that the table needs is a Tax Name and a percentage for taxing items. The ID field is managed internally by the database management system and is also the primary key for the table.

Vendor Table

Vendor_Table
+VND_ID: AutoIncrement INT
+VND_Name: varchar(255)
+VND_Address: TEXT
+VND_City: varchar(50)
+VND_State: varchar(50)
+VND_Zip_Code: varchar(12)
+VND_Phone_Number: varchar(20)
+VND_Sales_Representative: TEXT
+VND_Specialty: TEXT
+VND_Email_Address: varchar(255)
+VND_Home_Page: TEXT
+PrimaryKey(VND_ID)

The Vendor Table is used also as a supporting table for the inventory. This information pertains to the supplier of the items being sold. When an item reaches its reorder level in the Inventory Table, the information in this table would be used to make the order to resupply. The table is keyed by a unique ID that is managed by the database management system. The company name, address, and pertinent contact information is maintained along with a sales representative's name. There are optional fields for company specialty, email address, and homepage as well.

2.1.3 Transaction Handling

Transaction Handling is a process that has data spanning two tables with two more supporting tables. Transaction handling is split into two sections. The first section being the actual day to day transaction, and then the added functionality of packaging the items sold during the transaction.

Transaction Table

Transaction_Log_Table
+TRANS_Ref_EMP_ID_Number: INT +TRANS_Ref_Item_SKU_Number: varchar(10) +TRANS_Ref_Cust_Account_Number: INT +TRANS_Item_Sale_Price: DECIMAL(10,2) +TRANS_ID: INT NOT NULL +TRANS_Quantity: INT +TRANS_Comment: TEXT
+Primary Key(Ref_Emp_ID_Number, Ref_Item_SKU_Number, Ref_Cust_Account_Number, Item_Sale_Price, ID, Quantity) +INDEX trans_id(TRANS_ID) +INDEX emp_id(TRANS_REF_EMP_ID_Number) +INDEX sku_num(TRANS_REF_INV_SKU_Number) +INDEX cust_acct(TRANS_REF_CUST_Account_Number) +FOREIGN KEY(TRANS_REF_EMP_ID_Number): REFERENCES Employee_Table(EMP_ID_Number) +FOREIGN KEY(TRANS_REF_INV_SKU_Number): REFERENCES Inventory_Table(INV_SKU_Number) +FOREIGN KEY(TRANS_REF_CUST_Account_Number): REFERENCES Customer_Table(CUST_Account_Number)

The Transaction Log Table is used to store information that links Customers, Employees, and inventory items. Each entry in the transaction table represents an item sold during a transaction. Since the key is not a single ID number it is the combination of the Customer, Employee, Item, Quantity and Price, the ID can be used to represent the overall transaction. Any row that contains an equivalent ID belongs to the same transaction.

Package Table

Package_Table
+PKG_ID_Number: AutoIncrement INT +PKG_Ref_Trans_ID: INT +PKG_Ref_Cust_Account_Number: INT +PKG_Ref_Carrier_ID: INT +PKG_Tracking_Number: varchar(50) +PKG_Shipping_Type
+Primary Key(PKG_ID_Number) +INDEX trans_id(PKG_REF_TRANS_ID) +INDEX cust_acct(PKG_REF_CUST_Account_Number) +INDEX carr_id(PKG_REF_CRR_ID_number) +INDEX shp_type(PKG_REF_SHP_Shipping_Type) +FOREIGN KEY(PKG_REF_TRANS_ID): REFERENCES Transaction_Log_Table(TRANS_ID) +FOREIGN KEY(PKG_REF_CUST_Account_Number): REFERENCES Customer_Table(CUST_Account_Number) +FOREIGN KEY (PKG_REF_CRR_ID_Number): REFERENCES Carrier_Table(CRR_ID) +FOREIGN KEY(PKG_REF_SHP_Shipping_Type): REFERENCES Shipping_Table(SHP_Type)

The Package Table is used to store information about a package. There is a link to a transaction ID so that items can be associated with a package. There is also a link to a customer from the package table. The reason there is a link from the package table is because one customer may wish to buy something for another so the customer who made the transaction will not necessarily be the same as the one who will receive the package. There is also a reference to a carrier and a shipping type. A tracking number field is provided for when the tracking number is issued by the Carrier.

Carrier Table

Carrier_Table
+CRR_ID: AutoIncrement INT
+CRR_Name: varchar(50)
+CRR_Pickup_Location: TEXT
+CRR_Phone_Number
+PrimaryKey(CRR_ID)

The Carrier table is a support table for the Package table. It provides information about the different shipping services. The information maintained here is just what is necessary to get a package shipped. There is a phone number, pickup location and an ID associated with each entry.

Shipping Table

The Shipping Table is also a support table for the Package Table. This table is a list of all of the different methods of shipping available associated with the Carrier Table. Since each Carrier can have multiple shipping methods they can all be easily added and deleted if they ever change via this table. The table just contains the name for the type of shipping, the carrier, and how much it costs.

Shipping_Table
+SHP_Type: varchar(30)
+SHP_Ref_Carrier_ID: INT
+SHP_Cost: DECIMAL(7,2)
+PrimaryKey(SHP_Type, SHP_Ref_Carrier_ID)
+INDEX crr_id(SHP_REF_CRR_ID)
+INDEX shp_type(SHP_Type)
+FOREIGN KEY(SHP_REF_CRR_ID): REFERENCES Carrier_Table(CRR_ID)

2.1.4 Employee Management

The Employee Management section is actually two sections. There is the Employee portion which spans two tables, the Employee Table for employee information, and the Login Table which keeps track of the hours an employee works between logging in and out of the clients. The other section is the security aspect of the program as it relates to employees. It also spans two tables; the ACL Table and the Key Table.

Employee Table

Employee_Table
+EMP_Social_Security_Number: varchar(13) +EMP_ID_Number: AutoIncrement INT +EMP_First_Name: varchar(25) +EMP_Middle_Name: varchar(25) +EMP_Last_Name: varchar(50) +EMP_Address: TEXT +EMP_Phone_Number: varchar(20) +EMP_City: varchar(50) +EMP_Zip: varchar(12) +EMP_Picture: TEXT +EMP_Signature: TEXT +EMP_Access_Control_Level +Primary Key(EMP_Social_Security_Number) +UNIQUE INDEX(EMP_ID_Number) +INDEX acl_type(EMP_REF_ACL_Type) +FOREIGN KEY(EMP_REF_CUST_Account_Number): REFERENCES Customer_Table (CUST_Account_Number)

The Employee Table keeps track of all pertinent information about employees that would be needed by an employer. Basic personal information such as First, Middle, and Last name as well as Contact information are maintained here. Each employee in the table has a unique employee identification number associated with them to avoid having to use the Social Security Number as a key. This also allows data such as the Social Security number to be encrypted to alleviate underprivileged users from seeing it. Each employee also has a field for a picture link and signature link to help with positive identification. The other field in this table is the password field which is used to store a user's password to allow a user to login, and which is also used to decrypt the keys from the key table. Each user is also associated with a ACL entry in the ACL Table.

Login Table

Login_Table
+Ref_Emp_ID: INT +Log_Count: INT +LOG_In_Time: DATETIME +LOG_Out_Time: DATETIME +PrimaryKey(Ref_Emp_ID, Log_Time, Log_Flag)

The Login Table tracks the amount of time an employee has been logged in based on when they logged into their first client to when they logout of their last client. The structure is very simple. It references an employee ID and has a Count field. When the count is greater than zero the user is logged in and upon first entry a value will be inserted into the Log In Time field. When the value of Count reaches zero again a value will be inserted into the Log Out Time field.

2.1.5 Security

Access Control List Table

ACL_Table
+ACL_Type: varchar(30)
+ACL_Description: TEXT
+Primary Key(ACL_Type)

The ACL table which is short for Access Control List Table manages which user types have access to different levels of program use. All this table contains is a Name of a user type and a description of their functionality. Currently we foresee only four user types and will likely not have an interface to manage this table.

Key Table

Key_Table
+KEY_ID: INT AutoIncrement
+KEY_UserEmp_ID
+KEY_Keyin: TEXT
+INDEX (KEY_KeyID_Number)
+FOREIGN KEY (KEY_KeyID_Number) : REFERENCES Employee_Table (EMP_ID_Number)

The Key Table is used to manage the keys used to unencrypt the data stored in the database. The entry in this table will reference a valid user in the Employee Table and the text field will contain encrypted data that the user's password was used to encrypt. When the password is used to decrypt this data the user can obtain the keys used to encrypt global database data. A few example fields that are going to be stored as encrypted data are Credit Card Numbers and Expiration Date pairs as well as Social Security Numbers. This will add a level of database security that will prevent or deter malicious users from stealing valuable information from the database.

2.2 YardSale Client Application Design

The YardSale client is a modular structure designed to be flexible enough to accommodate any style of business. Each screen uses a bottom-oriented toolbar containing access to the on-screen calculator and keyboard, as well as the current time, an UNDO button, and a backwards navigation button.

2.2.1 Main Menu



The Main Menu is designed to allow users to quickly access any part of the YardSale client. Buttons are available to users depending on access level. The screen shown above displays an administrative user's access, as all options are currently available. The bottom toolbar does not contain the backwards navigation button, as this is the top-level screen. The buttons are enlarged to support touchscreen access.

2.2.2 Sales Screen

The screenshot shows a window titled "Sales" with a close button (X) in the top right corner. The window is divided into several sections:

- Customer Section:** Located on the top left, it contains fields for "First Name" (William), "Middle Name" (Joe), "Last Name" (Smith), "Address 1" (256 Elderberry Lane), and "Address 2" (Apt 33). Below these fields is an "Edit" button.
- Transaction Section:** Located on the top right, it features a table with two columns: "Description" and "Price". The table is currently empty.
- Inventory Item List:** A large empty rectangular area is located below the Customer section.
- Bottom Bar:** Contains several icons and a timestamp. From left to right: a blue circular arrow icon, a calculator icon, a keyboard icon, a trash can icon, the timestamp "22:56:13", and a blue circular refresh icon.

The Sales Screen is designed to provide much information and functionality, in an easy-access and user-friendly manner. It displays information about the current customer, which may be edited to make any corrections or updates. Below this is an inventory item list, which expands into a tree form to list similar products. This section may be used in the case of a barcode scanner malfunction, or to give information about items similar to the ones being purchased. To the right of the screen is the transaction shopping cart, which displays names and prices of items currently entered into the system to be purchased during this transaction. When items are present in this list, a running total (including tax) is displayed at the bottom of the list. The trash can icon is used to remove unwanted items from the list.

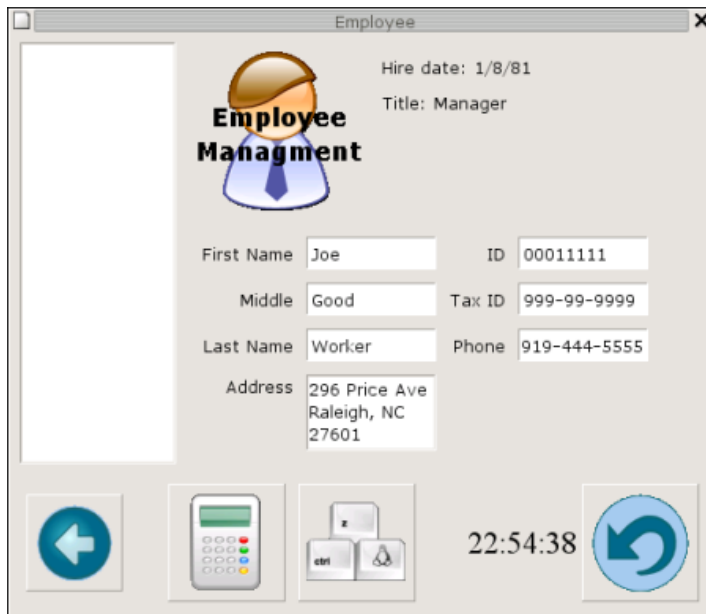
2.2.3 Inventory Screen

The screenshot shows a software window titled "Inventory" with a close button (X) in the top right corner. The window is divided into several sections:

- Form Fields:** On the left, there are input fields for "SKU", "Name", "Department", "Item Type", "Price", "Wholesale", "Weight (lbs)", "Vendor", and "Tax Type" (which has a dropdown menu labeled "ChoiceIter"). On the right, there are fields for "Bar Code", "Description" (a larger text area), "On Hand" (a numeric field with up/down arrows), "On Order" (a numeric field with up/down arrows), and "Reorder Level" (a numeric field with up/down arrows). There are also checkboxes for "Freight Only" and "Oversized".
- Buttons:** A vertical column of buttons is on the right side, including "New Item", "Search", "Clear", "Save", "Cancel", and "Clear Results".
- Table:** Below the form fields is a table with the following headers: "SKU", "BarCode", "Description", "Department", "On Hand", "Retail", and "Wholesale". The table body is currently empty.
- Footer:** At the bottom of the window, there are three icons on the left: a blue circular arrow pointing left, a calculator icon, and a printer icon. In the center, the time "22:57:34" is displayed. On the right, there is a blue circular arrow pointing right.

The Inventory Screen can display information pertaining to all inventory items currently stored in the database. There are fields available for every possible value of an inventory item, which may be edited and saved. In addition to these characteristics, the Inventory Screen has many other functions which it can perform. It can be used to add items to the inventory by filling in the appropriate fields and clicking the 'New Item' button. The Search function is used by entering the desired information into the appropriate fields and clicking the 'Search' button. This will cause all inventory items present in the database that satisfy the criteria to be displayed in the window at the bottom of the screen. For error checking purposes, a 'Save' and a 'Cancel' button are used when exiting this screen. This screen should only be available to manager-access-level users and higher.

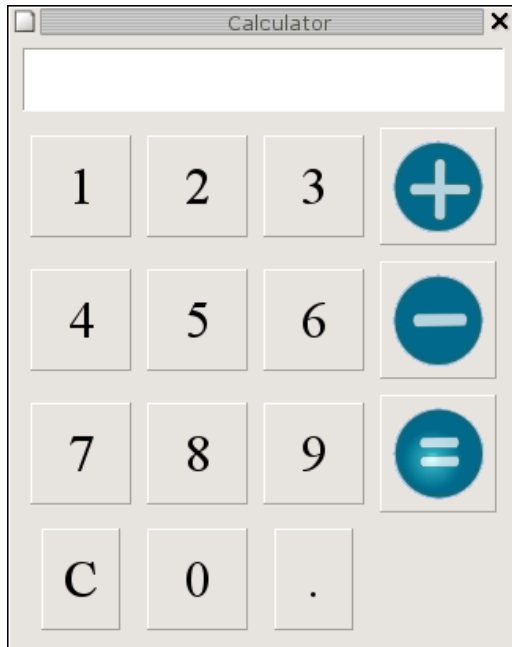
2.2.4 Employee Management Screen



The screenshot shows a software window titled "Employee" with a close button (X) in the top right corner. On the left is a large empty white rectangular area. To its right is a graphic of a person in a blue shirt and tie with the text "Employee Management" overlaid. Further right, the text "Hire date: 1/8/81" and "Title: Manager" is displayed. Below this is a form with the following fields: "First Name" with the value "Joe", "Middle" with "Good", "Last Name" with "Worker", and "Address" with "296 Price Ave", "Raleigh, NC", and "27601". To the right of these are "ID" (00011111), "Tax ID" (999-99-9999), and "Phone" (919-444-5555). At the bottom, there is a row of icons: a blue circular arrow pointing left, a calculator icon, a printer icon, a clock showing "22:54:38", and a blue circular arrow pointing right.

The Employee Management Screen is designed to store and display information pertaining to all employees currently in the system. There are fields available for every possible value of an employee item in the database. Information is displayed by entering the desired employee's ID number and pressing enter. This screen should be carefully guarded, as sensitive information is displayed. It should only be available to manager-access-level users and higher.

2.2.5 On-screen Calculator



The On-Screen Calculator is designed to provide the user with an easy-access, user-friendly calculator and number pad. It is available to all users from each screen in the system, via the tool bar at the bottom of the screen. It provides the functionality of a basic calculator (add and subtract, multiply and divide to be added at a later date). Numbers entered and the results are displayed in the box at the top of the calculator.

Chapter 3

Implementation

3.1 Database Level Implementation

3.1.1 Customer Management

```
DROP TABLE IF EXISTS CustomerTable;
```

```
CREATE TABLE CustomerTable(  
    CUSTAccountNumber INT AUTOINCREMENT NOT NULL,  
    CUSTFirst_Name varchar(25),  
    CUSTMiddle_Name varchar(25),  
    CUSTLast_Name varchar(50),  
    CUSTAddress TEXT,  
    CUSTPhone varchar(20),  
    CUSTCity varchar(50),  
    CUSTZip varchar(12),  
    CUSTCredit_CardNumber varchar(20),  
    CUSTCCExpDate varchar(6),  
    CUSTNameOnCC TEXT,  
    CUSTSignature TEXT,  
    CUSTPhoto TEXT,  
    Primary Key(CUSTAccountNumber)  
)type=InnoDB
```

3.1.2 Inventory Management

```
DROP TABLE IF EXISTS InventoryTable;
```

```
CREATE TABLE InventoryTable(  
    INV_SKUNumber varchar(10),  
    INV_Bar_CodeNumber varchar(30),  
    INV_Item_Description TEXT,  
    INV_Item_Department varchar(30),
```

```

INV_Quantity_OnHand INT,
INV_Quantity_OnOrder INT,
INV_Reorder_Level INT,
INV_Reorder_Quantity INT,
INV_Item_Type varchar(20),
INV_REETAXTax_Type INT NOT NULL,
INV_REEVNDVendor_ID INT NOT NULL,
INV_Retail_Price DECIMAL(7,2),
INV_Wholesale_Price DECIMAL(7,2),
INV_Bulk_Price TEXT,
INV_Date_Last_Received DATETIME,
INV_Weight_Pounds FLOAT,
INV_Oversized_Flag enum('T','F'),
INV_Ship_By_Freight enum('T','F'),
INV_Comment TEXT,
Primary Key (INV_SKUNumber),
UNIQUE INDEX (INV_Bar_CodeNumber),
INDEX tax_id (INV_REETAXTax_Type),
INDEX vndid (INV_REEVNDVendor_ID),
FOREIGN KEY (INV_REETAXTax_Type)
REFERENCES TaxTable(TAX.ID),
FOREIGN KEY (INV_REEVNDVendor_ID)
REFERENCES VendorTable(VND.ID)
) type=InnoDB

DROP TABLE IF EXISTS TaxTable;

CREATE TABLE TaxTable(
    TAXID INT AUTOINCREMENT NOT NULL,
    TAXName varchar(20),
    TAXPercent FLOAT,
    Primary Key (TAX.ID)
)type=InnoDB

```

3.1.3 Transaction Handling

```
DROP TABLE IF EXISTS Transactionbog_Table;
```

```

CREATE TABLE Transactionbog_Table(
    TRANSREFEMBD_Number INT,
    TRANSREFINV_SKUNumber varchar(10),
    TRANSREFCUSTAccount_Number INT,
    TRANSale_Price DECIMAL(10,2),

```

```

TRANSID INT NOT NULL,
TRANSQuantity INT,
TRANSComment TEXT,
Primary Key(TRANSREFEMID_Number,
            TRANSREFINV_SKUNumber,
            TRANSREFCUSTAccount_Number,
            TRANSale_Price, TRANSID,
            TRANSQuantity),
INDEX trans_id (TRANSID),
INDEX emp_id (TRANSREFEMID_Number),
INDEX skunum (TRANSREFINV_SKUNumber),
INDEX custacct (TRANSREFCUSTAccount_Number),
FOREIGN KEY (TRANSREFEMID_Number) REFERENCES EmployeeTable(EMPID_Number),
FOREIGN KEY (TRANSREFINV_SKUNumber) REFERENCES InventoryTable(INV_SKUNumber),
FOREIGN KEY (TRANSREFCUSTAccount_Number) REFERENCES CustomerTable(CUSTAccount_Number)
)type=InnoDB

```

```

DROP TABLE IF EXISTS PackageTable;

```

```

CREATE TABLE PackageTable(
    PKGID_Number INT AUTOINCREMENT NOT NULL,
    PKGREETRANSID INT,
    PKGREECUSTAccount_Number INT,
    PKGREECRBD_Number INT,
    PKGTracking_Number varchar(50),
    PKGREESHPShipping_Type varchar(30),
    Primary Key(PKGID_Number),
    INDEX trans_id (PKG.REETRANSID),
    INDEX custacct (PKG.REECUSTAccount_Number),
    INDEX crr_id (PKG.REECRBD_number),
    INDEX shptype (PKG.REESHPShipping_Type),
    FOREIGN KEY (PKG.REETRANSID) REFERENCES TransactionLogTable( TRANSID ),
    FOREIGN KEY (PKG.REECUSTAccount_Number) REFERENCES CustomerTable( CUSTAccount_Number ),
    FOREIGN KEY (PKG.REECRBD_Number) REFERENCES CarrierTable( CRRID ),
    FOREIGN KEY (PKG.REESHPShipping_Type) REFERENCES ShippingTable( SHP.Type )
) type=InnoDB

```

```

DROP TABLE IF EXISTS CarrierTable;

```

```

CREATE TABLE CarrierTable(
    CRRID INT AUTOINCREMENT NOT NULL,
    CRRName varchar(50),
    CRRPickup_Location TEXT,
    CRRPhone.Number varchar(20),

```

```

    Primary Key (CRRID)
)type=InnoDB

DROP TABLE IF EXISTS ShippingTable;

CREATE TABLE ShippingTable(
    SHPType varchar(30),
    SHPRECRID INT,
    SHPCost DECIMAL(7,2),
    Primary Key(SHPType, SHPRECRID, SHPCost),
    INDEX crr_id (SHP_REFCRID ),
    INDEX shptype (SHP_Type),
    FOREIGN KEY (SHP_REFCRID) REFERENCES CarrierTable( CRRID )
)type=InnoDB

```

3.1.4 Employee Management

```

DROP TABLE IF EXISTS EmployeeTable;

CREATE TABLE EmployeeTable(
    EMPSocial _Security _Number varchar(13) NOT NULL,
    EMPID _Number INT NOT NULL,
    EMPFirst _Name varchar(25),
    EMMiddle _Name varchar(25),
    EMELast _Name varchar(50),
    EMPAddress TEXT,
    EMPPhoneNumber varchar(20),
    EMPCity varchar(50),
    EMPZip varchar(12),
    EMPPicture TEXT,
    EMPSignature TEXT,
    EMREFACLType varchar(30),
    #EMREFCUSTAccount _Number INT,
    Primary Key (EMP_Social _Security _Number),
    UNIQUE INDEX(EMP _Number),
    INDEX acltype (EMP.REFACLType),
    #INDEX acctnumber (EMREFCUSTAccount _Number),
    FOREIGN KEY (EMPREFACLType) REFERENCES ACLTable(ACL_Type),
    #FOREIGN KEY (EMREFCUSTAccount _Number) REFERENCES CustomerTable(CUSTAccount _Number)
) type=InnoDB

DROP TABLE IF EXISTS KeyTable;

CREATE TABLE KeyTable(

```

```

KEYID INT AUTOINCREMENT,
KEYREFEMBED_Number INT,
KEYKeys TEXT,
PRIMARY KEY ( KEY ),
INDEX ( KEYREFEMBED_Number ),
FOREIGN KEY (KEYREFEMBED_Number) REFERENCES EmployeeTable( EMP.ID.Number )
)type=InnoDB

DROP TABLE IF EXISTS ACLTable;

CREATE TABLE ACLTable(
    ACLType varchar(30),
    ACLDescription TEXT,
    Primary Key (ACLType)
) type=InnoDB

```

3.2 Application Level Implementation

3.2.1 YardCalc

YardCalc is a generic on-screen calculator the user employs to enter prices. This calculator uses a stack-based method of storing numbers and operators.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.3

3.2.2 YardDatabase

YardDatabase is the main database backend which does all translation from OO calls to SQL/ODBC.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.6

3.2.3 YardDBType

YardDBType is the abstract base class for all database objects. All database types are assignable and contain a ToString() method to format the DB type to text.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.8

YardEmployeeType

YardEmployeeType is a subclass of YardDBType, which represents an employee record and contains functions for all possible items.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.11

YardInvType

YardInvType is a subclass of YardDBType, which represents an employee record and contains functions for all possible items.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.15

YardInvType::BulkPricing

BulkPricing is a C++ structure that associates a quantity with a percentage for bulk pricing.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.16

3.2.4 YardEmployee

YardEmployee is the employee management screen. Depending on access level, users may insert/modify employee information via this screen.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.10

3.2.5 YardInventory

YardInventory is the inventory management screen, which allows searching. Depending on access level, users may add inventory via the "New Item" button on this screen.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.14

3.2.6 YardException

YardException is the exception class from Crypto++.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.12

3.2.7 YardLog

YardLog is the logging widget, based on wxListCtrl and wxLog. This widget resets the default logging system and redirects all output to itself. Different icons represent what type of log message is being displayed.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.17

3.2.8 YardLogin

YardLogin is the customized login screen. The user will be asked for a username and password. Also, a quick select icon will allow the user to rapidly select his/her name.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.18

3.2.9 YardMain

YardMain is the main menu screen, which displays graphical buttons for accessing each part of the system.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.19

3.2.10 YardSale

YardSale is the main application object, which returns a reference to a YardDatabase object.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.20

3.2.11 YardSaleScreen

YardSaleScreen is main sale screen, which contains the current transaction information and an interface to add new items to the transaction. The payment screen can be accessed from YardSaleScreen.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.21

3.2.12 YardSplash

YardSplash is an eye-candy, startup splash screen that shows a progress bar.

SEE YARDSALE IMPLEMENTATION MANUAL SECTION 7.22

Chapter 4

Test Plan

The YardSale system is tested constantly throughout the development process. These tests are administered through two different methods: passive (automatic) testing, and active (user) testing.

4.1 Passive Testing

The passive testing of YardSale is accomplished by our auto build system. The auto build system pulls the latest source code from the repository and compiles it with the most strict settings. If the code fails the system reports the error on our website. The auto build system will also compile the code on six other architectures including Amd64, Sparc, Mac OSX, Linux, and Microsoft Windows. The extensive compilation of the source code ensures that no platform dependant source code violates the portability of the project.

4.2 Active Testing

There are several methods which are utilized by our development team to manually test the YardSale system throughout the development process.

4.2.1 Testing Mains

Each non-gui object in the YardSale system has an integrated main loop which allows the program to build test versions of each of the objects. These test objects then execute a series of tests to ensure that the object is working exactly to specification. If any of these tests fail the system will report the exact location of the failure.

4.2.2 Warm-Body Testing

Each GUI screen is tested by our virtual employees for both functionality and usability. The employees report back to the developers with any bugs they find through our bug tracking system. We also have weekly usability conferences where we discuss the merits of a particular interface.

4.2.3 Verbose Program Information

YardSale is designed to report back a large amount of valuable debug information to the developer and to keep the non-technical user informed about the status of the program. YardSale uses exceptions to ensure that single functions cannot violate the integrity of the system on a global level. In addition, all database interfaces have special logging information that reports the exact status of all database calls; this allows the developers to get instant access to valuable SQL and connection results.