

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 Definition	3
1.1	Introduction	3
1.2	Functional Requirements	3
1.2.1	Information Management	4
1.2.2	Point of Sale Transactions	8
1.2.3	Reporting	8
1.2.4	Login/User Access Levels	8
1.3	Non Functional Requirements	9
1.3.1	Operating System Independent	9
1.3.2	Architecture Independent	10
1.4	System Constraints	10
1.4.1	Interfaces	10
1.5	External Dependencies and Interfaces	10
1.5.1	Host Operating System	10
1.5.2	Input Processing Device	10
1.5.3	Output Processing Device	10
1.6	Preliminary Design	11
1.6.1	Major Modules	11
1.6.2	System Architecture	12
1.7	Preliminary Project Task Plan	12
1.7.1	Milestones	12
1.7.2	Team Member Roles and Responsibilities	13
2	Design	14
2.1	YardSale Database Design	14
2.1.1	Customer Management	17
2.1.2	Inventory Management	18
2.1.3	Transaction Handling	20
2.1.4	Shipping and Package Management	21
2.1.5	Employee Management	22
2.1.6	Security	23
2.2	YardSale Client Application Design	25
2.2.1	Main Menu	25
2.2.2	Sales Screen	26
2.2.3	Inventory Screen	27
2.2.4	Employee Management Screen	28
2.2.5	On-screen Calculator	29

3	Implementation	30
3.1	Database Level Implementation	30
3.1.1	Customer Management	30
3.1.2	Inventory Management	30
3.1.3	Transaction Handling	31
3.1.4	Employee Management	33
3.2	Application Level Implementation	34
3.2.1	YardCalc	34
3.2.2	YardDatabase	34
3.2.3	YardDBType	34
3.2.4	YardEmployee	35
3.2.5	YardInventory	35
3.2.6	YardException	35
3.2.7	YardLog	35
3.2.8	YardLogin	35
3.2.9	YardMain	35
3.2.10	YardSale	35
3.2.11	YardSaleScreen	35
3.2.12	YardSplash	35
4	Test Plan	36
4.1	Testing Methods	36
4.1.1	Passive Testing	36
4.1.2	Active Testing	36
4.1.3	Functional Requirements Testing	37

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 components of a YardSale system 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 Point of Sale Related Data
 - Inventory Data
 - * Add a product to the inventory
 - * Edit an existing inventory item
 - * Search for an inventory item by various criteria
 - * Remove an inventory item

- * Add an inventory item to a transaction
- Customer Data
 - * Add and remove customers
 - * Search for customers by various criteria
 - * Edit customer information
 - * Associate a customer with a transaction
- Employee Data
 - * Add and remove employees
 - * Associate an access level with each employee
 - * Associate an employee with a transaction
 - * Associate Login information with each employee
- Transaction Data
 - * Add new transaction information
 - * Search for transactions by various criteria
 - * Associate values with a transaction
- Perform Point of Sale Transactions
 - Allow sale and return of inventory items between customers and employees
 - Calculate the tax on an item being sold
 - Record all transaction information
- Provide Detailed Reporting
 - Allow a manager to produce reports related to the Point of Sale information recorded by the system.
- Allow for User Login with Differing Access Levels
 - Administrator (Super User) Access
 - Manager Access
 - Sales Associate Access

Each of these tasks is described in-depth in the corresponding subsection. 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 the four following subsections:

- Inventory Data
- Customer Data
- Employee Data
- Transaction Data

Inventory Data

YardSale will store data pertaining to all inventory items currently in-stock and on-order. The following fields will be recorded for each item:

- SKU Number
- Barcode Number
- Item Name
- Corresponding Department
- Item Type
- Retail Price
- Wholesale Price
- Weight (in pounds)
- Associated Vendor
- Associated Tax Type
- Item Description
- Shipping Flags
 - Freight Only?
 - Oversized?
- Number In-Stock
- Number On-Order
- Reorder Level

YardSale shall provide the following functionality to all inventory data in the system:

- **Add New Inventory:** new inventory items shall be able to be added to the current inventory.
- **Search Inventory:** all inventory items shall be available to a search function; items retrieved based on specified criteria.
- **Modify Inventory Item:** all inventory items shall be able to have any field modified and saved.
- **Remove Inventory:** existing inventory items shall be able to be removed from the list of current inventory items.
- **Associate Inventory Item with a Transaction:** each inventory item shall be able to be associated with a transaction in which the item's inventory status was altered (*see TRANSACTION DATA*).

Customer Data

YardSale will store data pertaining to all customers of the system's business. The following fields will be recorded for each customer:

- Customer Account Number
- First Name
- Middle Name
- Last Name
- Address
- Phone Number
- City
- State
- Zip Code
- Credit Card Number
- Credit Card Expiration Date
- Name on Credit Card
- Signature

YardSale shall provide the following functionality to all customer data in the system:

- **Add Customer:** new customers shall be able to be added to the current customer list.
- **Search Customers:** all customers shall be available to a search function; customers displayed based on specified criteria.
- **Modify Customer:** all customers shall be able to have any field modified and saved.
- **Associate Customer with a Transaction:** customers shall be able to be associated with a transaction in which they altered the state of some inventory item (*see TRANSACTION DATA*).

Employee Data

YardSale will store data pertaining to all employees currently employed by the business. The following fields will be recorded for each employee:

- Social Security Number
- Employee ID Number
- First Name
- Middle Name
- Last Name
- Address

- Phone Number
- City
- State
- Zip Code
- Signature
- Access Control Level (*see section 1.2.4*)

YardSale shall provide the following functionality to all employee data in the system:

- **Add New Employee:** new employees shall be able to be added to the current employment list.
- **Remove Employee:** existing employees shall be able to be removed from the list of current employees.
- **Associate Employees with an Access Control Level:** each existing employee shall be able to be associated with a predefined access level to the system.
- **Associate Employee with a Transaction:** employees shall be able to be associated with a transaction in which they altered the state of some inventory item (*see TRANSACTION DATA*).

Related to Employee Information, the system shall record information pertaining to user logins. This information should be inherent within the system and store the following information upon login and logout of each user:

- Employee ID Number
- Date and Time Logged In
- Date and Time Logged Out

Transaction Data

YardSale will store data pertaining to all transactions made by the system. The following fields will be recorded for each transaction:

- Employee ID Number (external reference)
- Inventory SKU Number (external reference)
- Customer Account Number (external reference)
- Transaction ID Number
- Sale Price
- Sale Quantity
- Comments

YardSale shall provide the following functionality to all transaction data in the system:

- **Add New Transaction:** new transaction data shall be able to be added to the current transaction list. This should occur automatically with the completion of any transaction.
- **Search Transactions:** all transactions shall be available to a search function; transactions displayed based on specified criteria.
- **Associate Values with a Transaction:** externally referenced values; such as employee ID, SKU, and customer account numbers, shall be able to be referenced and saved with each transaction.

1.2.2 Point of Sale Transactions

YardSale shall allow for Point of Sale Transactions of inventory items to be made between employees and customers. Each transaction shall satisfy the following criteria:

- Must contain one or more inventory items to be either sold or returned.
- Sales transaction functionality shall be available to all current employees in the system, regardless of access control level.
- Return transaction functionality shall only be available to employees with appropriately specified access control level.
- Must associate proper tax type/information with each item of a transaction.
- Completion of a transaction shall create new Transaction Data to be stored by the system; including the following external data:
 - **Customer Information:** Transactions shall be linked with an associated customer stored in the system.
 - **Employee Information:** Transactions shall be linked with an associated employee stored in the system.
 - **Inventory Information:** all information pertaining to the inventory items altered during the transaction (either added or removed).

see TRANSACTION DATA of section 1.2.1

1.2.3 Reporting

YardSale shall provide detailed reporting mechanisms to managers of the system. These reports shall be automatically generated by the system from the data stored by the Information Management System. Reporting shall be available via an easily accessible template to users with the appropriate user access level. The following reports shall be available by default:

- **Hourly Reporting for Employees:** this report will provide an exact output of total hours each employee was logged into the system.
- **Total Transactions by Time Period:** this report will provide an output of all transaction information stored during the specified time period.
- **Sales Transactions by Employee:** this report will provide an output of all sales transactions made by the selected employee(s), or all employees.

Each of these reports shall be made available for printing in a commonly used format, supported by all operating systems.

1.2.4 Login/User Access Levels

YardSale shall allow for users to log into the system with predefined system access. One of the following User Access Levels shall be associated with every user of the system:

Sales Associate Functionality

The Sales Associate User shall be the primary user of the YardSale system. A Sales Associate must be a valid employee of the business currently running the system. This access level is the lowest and most basic of the three User Access Levels, as its rights are given to all current employees by default. Sales Associates shall be given access to the following areas of the system:

- Customer Information Management
- Sales/Return Transaction Processing

Manager Functionality

The Manager User shall be given nearly full functionality to the YardSale system. A Manager must be a valid employee of the business currently running the system. This access level's functionality is designed to allow for maintenance of most areas of the system. Upon login, the following options shall be available to a Manager:

- Inventory Information Management
- Customer Information Management
- Employee Information Management
- Reporting Mechanisms
- Sales/Return Transaction Processing

Administrative Functionality

The Administrative User shall be given full functionality of the YardSale system, and all related, underlying systems (Information Management System, Operating System...). An Administrator need not be an employee of the business currently running the system. This will allow for valid entry of technicians and A.S. Logic Systems employees for servicing and setup of the system. The Administrative User is a "Super User" and should not be used in the day-to-day routines of the YardSale system. Upon login, the following options shall be available to an Administrator:

- Inventory Information Management
- Customer Information Management
- Employee Information Management
- Reporting Mechanisms
- Sales/Return Transaction Processing
- System Configuration

1.3 Non Functional Requirements

1.3.1 Operating System Independent

The YardSale system shall be designed to be compatible with all Operating System platforms available in today's marketplace.

1.3.2 Architecture Independent

The YardSale system shall be designed to be independent of all architectural aspects of the underlying system. This is the main aspect of YardSale, as it should be able to run appropriately on any system meeting the Operating System's Minimal Requirements.

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 Drawer:** 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 Scanner:** 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 Scanner:** The system must support the use of a barcode scanner for automatic input of items for inventory and transaction processing.
- **Receipt Printer:** 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. In addition, the system's user interface must be easily used with the TouchScreen monitor.

1.5 External Dependencies and Interfaces

1.5.1 Host Operating System

The YardSale system requires the use of a host operating system to run the YardSale software. The constraints on this operating system are minimal and nearly all currently used operating systems are supported by YardSale. YardSale shall be designed to function properly with the minimal system requirements of the host operating system.

1.5.2 Input Processing Device

In addition to the devices defined by the System Constraints (see *SYSTEM CONSTRAINTS* sec 1.4.1), some additional input devices are required for entering information into the system. Examples of such devices are a keyboard or mouse.

1.5.3 Output Processing Device

In addition to the devices defined by the System Constraints (see *SYSTEM CONSTRAINTS* sec 1.4.1), some additional output devices are required for retrieving information from the system. Examples of such devices are a monitor and printer.

1.6 Preliminary Design

1.6.1 Major Modules

Database

Information Management in YardSale will be handled through the use of a database. The database will utilize the querying power of mySQL4 incorporated with ODBC.

The Database module works as the translator between the user and the database. It converts calls made by the user 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 (see *DATABASE TYPES*) to be read by the user interface. OTL Libraries will be used in conjunction with ODBC to provide database connectivity with the user interface. OTL is a cross platform C++ library for ODBC.

Database Types

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

Inventory Information Type: contains variables for all fields defined by the Inventory Information Management section of this document. Each of these fields will be stored in the Inventory Table of the database. Coordinates with the Inventory Management view of the user interface, which is also referenced by Transactions.

Customer Information Type: contains variables for all fields defined by the Customer Information Management section of this document. Each of these fields will be stored in the Customer Table of the database. Coordinates with the Customer Management view of the user interface, which is also referenced by Transactions.

Employee Information Type: contains variables for all fields defined by the Employee Information Management section of this document. Each of these fields will be stored in the Employee Table of the database. Coordinates with the Employee Management view of the user interface.

Transaction Information Type: contains variables for all fields defined by the Transaction Information Management section of this document. Each of these fields will be stored in the Transaction Log Table of the database. Information is automatically updated upon the conclusion of a Transaction in the user interface.

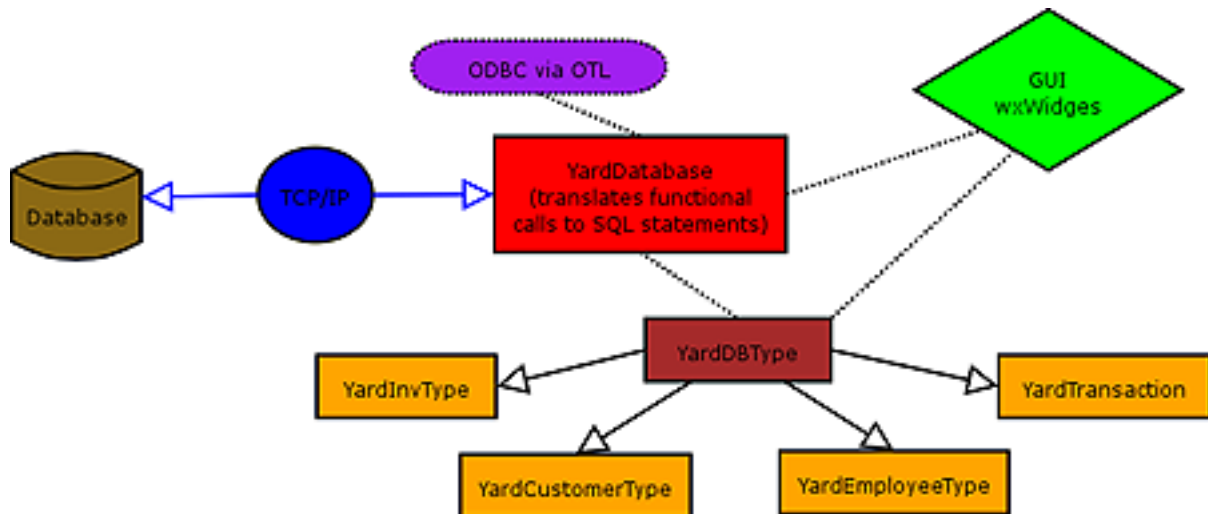
GUI wxWidgets

The User Interface for YardSale will be designed using the wxWidgets libraries for C++.

The GUI wxWidgets module defines all screens and windows used by the user interface. It contains functions derived from the wxWidgets libraries and specifies which functions to call for each action within the user interface. 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.

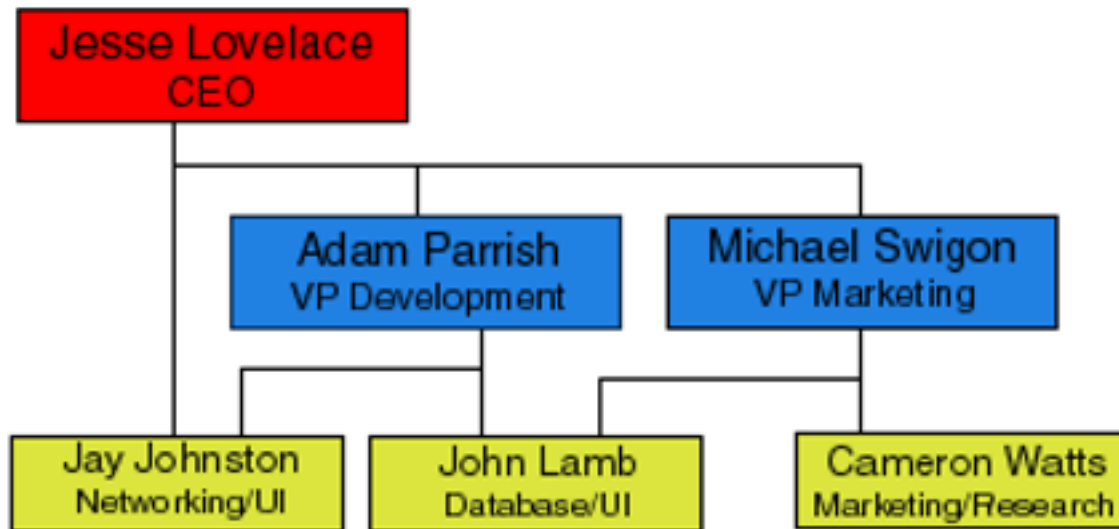
2004-04-01 ITERATION 2 - all secondary requirements to be completed for the second iteration.

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. A database was chosen to store the data for the application since we require persistent data storage. A database also is a robust method for storage, and easily queried as opposed to flat text files or binary file storage methods. There are also built in user access rights which will aid in security for YardSale.

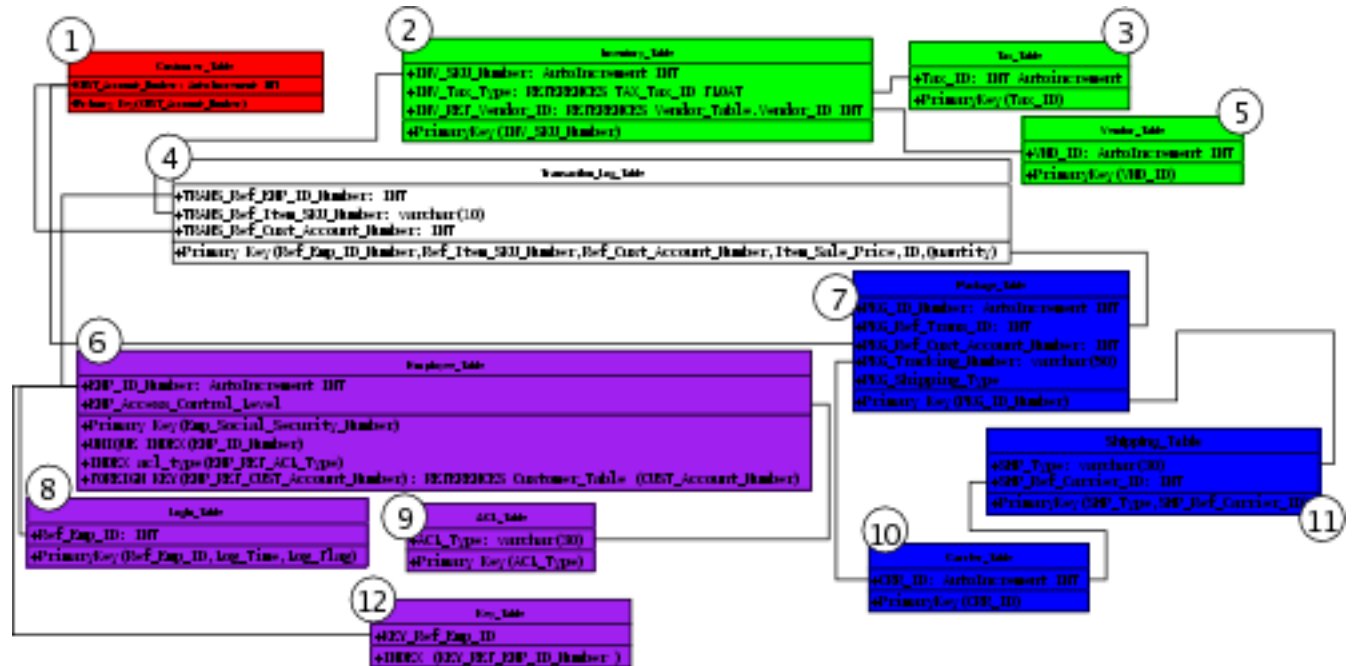
Our Chosen Relational Database Management System for development is MySQL 4 although since all of our queries are based on the SQL standard it should be easy to allow the use of any SQL Compliant RDBMS. For Application to Database connectivity we plan to use OTL in conjunction with ODBC. OTL is a set of C++ libraries which provide a higher level SQL layer between the actual database and the application. ODBC is used to interface with the actual database on an operating system and network level.

The following first section provides a high level explanation of the database architecture using UML diagrams. The latter 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.

YardSale Global Database Diagram



The database is broken down into the following sections in the above color coded diagram. Each color corresponds to a section in the latter portion of the Database Design section.

Customer Management Red

Inventory Management Green

POS Transaction Handling White

Shipping Management Blue

Employee Management Purple

The following is a legend that describes the database layout diagram. Each number corresponds to a numbered table in the diagram. Each table lists it's key as well.

1. Customer Table
 - Primary Key: Customer Account Number
2. Inventory Table
 - Primary Key: Inventory SKU Number
3. Tax Table
 - Primary Key: Tax ID

4. Transaction Log Table

- Primary Key: Key composed of all referenced keys

5. Vendor Table

- Primary Key: Vendor ID

6. Employee Table

- Primary Key: Employee Social Security Number

7. Package Table

- Primary Key: Package ID Number

8. Login Table

- Primary Key: Key composed of all referenced keys

9. ACL Table

- Primary Key: ACL Type

10. Carrier Table

- Primary Key: Carrier ID

11. Shipping Table

- Primary Key: Key composed of all referenced keys

12. Key Table

- Primary Key: Key ID

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 across many different 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 fit.

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 separate fields 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 field 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 the 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 subcatagorizing. 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
<pre> +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) </pre>

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.

2.1.4 Shipping and Package Management

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)

2.1.5 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 users 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
+Primary Key(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.

2.1.6 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_User_Emp_ID
+KEY_Key: TEXT
+INDEX: (KEY_User_Emp_ID,Key)
+FOREIGN KEY (KEY_User_Emp_ID) : 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

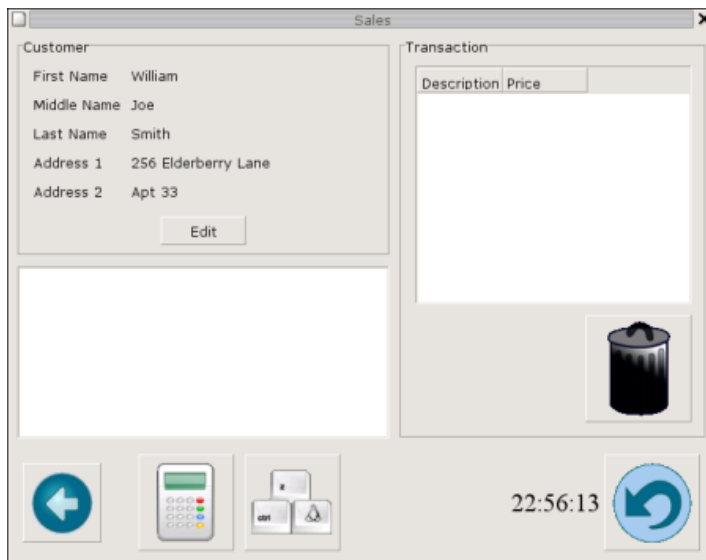
The YardSale client is a modular structure designed to be flexible enough to accommodate any style of business. Each screen uses a bottom-oriented toolbar containing access to the on-screen calculator and keyboard, as well as the current time, an UNDO button, and a backwards navigation button.

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

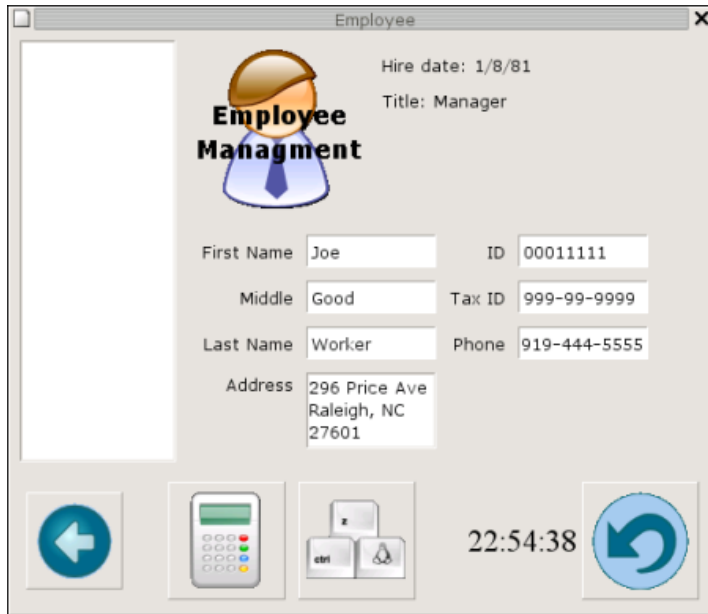


The Sales Screen is designed to provide much information and functionality, in an easy-access and user-friendly manner. It displays information about the current customer, which may be edited to make any corrections or updates. Below this is an inventory item list, which expands into a tree form to list 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 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.

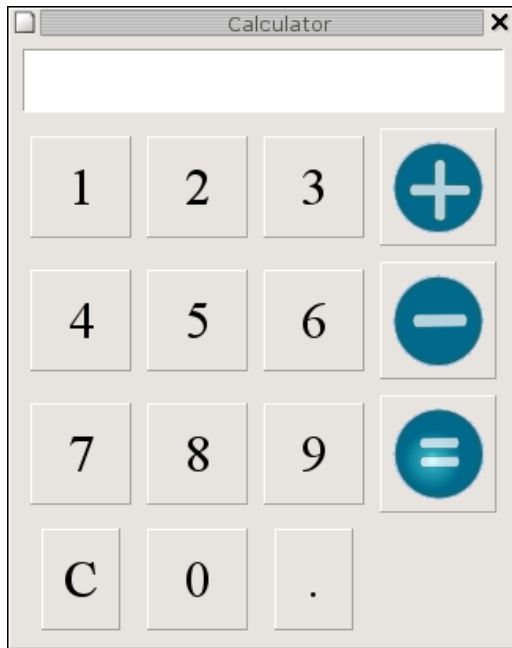
2.2.4 Employee Management Screen



The screenshot shows a 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 wearing a hard hat and a blue shirt with a tie, with the text "Employee Management" overlaid. Further right, the text "Hire date: 1/8/81" and "Title: Manager" is displayed. Below this, there are several input fields: "First Name" with "Joe", "Middle" with "Good", "Last Name" with "Worker", and "Address" with "296 Price Ave", "Raleigh, NC", and "27601". To the right of these are fields for "ID" (00011111), "Tax ID" (999-99-9999), and "Phone" (919-444-5555). At the bottom, there are four icons: a blue circle with a white left arrow, a calculator icon, a printer icon, and a blue circle with a white right arrow. In the center of the bottom row is a digital clock displaying "22:54:38".

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 Customer_Table;
```

```
CREATE TABLE Customer_Table(  
    CUST_Account_Number INT AUTO_INCREMENT NOT NULL,  
    CUST_First_Name varchar(25),  
    CUST_Middle_Name varchar(25),  
    CUST_Last_Name varchar(50),  
    CUST_Address TEXT,  
    CUST_Phone 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)  
)type=InnoDB
```

3.1.2 Inventory Management

```
DROP TABLE IF EXISTS Inventory_Table;
```

```
CREATE TABLE Inventory_Table(  
    INV_SKU_Number varchar(10),  
    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_REF_TAX_Tax_Type INT NOT NULL,
    INV_REF_VND_Vendor_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_SKU_Number),
    UNIQUE INDEX (INV_Bar_Code_Number),
    INDEX tax_id (INV_REF_TAX_Tax_Type),
    INDEX vnd_id (INV_REF_VND_Vendor_ID),
    FOREIGN KEY (INV_REF_TAX_Tax_Type)
    REFERENCES Tax_Table(TAX_ID),
    FOREIGN KEY (INV_REF_VND_Vendor_ID)
    REFERENCES Vendor_Table(VND_ID)
) type=InnoDB

DROP TABLE IF EXISTS Tax_Table;

CREATE TABLE Tax_Table(
    TAX_ID INT AUTO_INCREMENT NOT NULL,
    TAX_Name varchar(20),
    TAX_Percent FLOAT,
    Primary Key (TAX_ID)
)type=InnoDB

```

3.1.3 Transaction Handling

```

DROP TABLE IF EXISTS Transaction_Log_Table;

```

```

CREATE TABLE Transaction_Log_Table(
    TRANS_REF_EMP_ID_Number INT,
    TRANS_REF_INV_SKU_Number varchar(10),
    TRANS_REF_CUST_Account_Number INT,
    TRANS_Sale_Price DECIMAL(10,2),

```



```
TRANS_ID INT NOT NULL,
TRANS_Quantity INT,
TRANS_Comment TEXT,
Primary Key(TRANS_REF_EMP_ID_Number,
            TRANS_REF_INV_SKU_Number,
            TRANS_REF_CUST_Account_Number,
            TRANS_Sale_Price, TRANS_ID,
            TRANS_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)
)type=InnoDB

DROP TABLE IF EXISTS Package_Table;

CREATE TABLE Package_Table(
    PKG_ID_Number INT AUTO_INCREMENT NOT NULL,
    PKG_REF_TRANS_ID INT,
    PKG_REF_CUST_Account_Number INT,
    PKG_REF_CRR_ID_Number INT,
    PKG_Tracking_Number varchar(50),
    PKG_REF_SHP_Shipping_Type varchar(30),
    Primary Key(PKG_ID_Number),
    INDEX trans_id (PKG_REF_TRANS_ID),
    INDEX cust_acct (PKG_REF_CUST_Account_Number),
    INDEX crr_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 )
) type=InnoDB

DROP TABLE IF EXISTS Carrier_Table;

CREATE TABLE Carrier_Table(
    CRR_ID INT AUTO_INCREMENT NOT NULL,
    CRR_Name varchar(50),
    CRR_Pickup_Location TEXT,
    CRR_Phone_Number varchar(20),
```

```

    Primary Key (CRR_ID)
)type=InnoDB

DROP TABLE IF EXISTS Shipping_Table;

CREATE TABLE Shipping_Table(
    SHP_Type varchar(30),
    SHP_REF_CRR_ID INT,
    SHP_Cost DECIMAL(7,2),
    Primary Key(SHP_Type, SHP_REF_CRR_ID, SHP_Cost),
    INDEX crr_id (SHP_REF_CRR_ID ),
    INDEX shp_type (SHP_Type),
    FOREIGN KEY (SHP_REF_CRR_ID) REFERENCES Carrier_Table( CRR_ID )
)type=InnoDB

```

3.1.4 Employee Management

```

DROP TABLE IF EXISTS Employee_Table;

CREATE TABLE Employee_Table(
    EMP_Social_Security_Number varchar(13) NOT NULL,
    EMP_ID_Number INT NOT NULL,
    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_REF_ACL_Type varchar(30),
    #EMP_REF_CUST_Account_Number INT,
    Primary Key (EMP_Social_Security_Number),
    UNIQUE INDEX(EMP_ID_Number),
    INDEX acl_type (EMP_REF_ACL_Type),
    #INDEX acct_number (EMP_REF_CUST_Account_Number),
    FOREIGN KEY (EMP_REF_ACL_Type) REFERENCES ACL_Table(ACL_Type),
    #FOREIGN KEY (EMP_REF_CUST_Account_Number) REFERENCES Customer_Table(CUST_Account_Number)
) type=InnoDB

DROP TABLE IF EXISTS Key_Table;

CREATE TABLE Key_Table(

```

```

KEY_ID INT AUTO_INCREMENT,
KEY_REF_EMP_ID_Number INT,
KEY_Keys TEXT,
PRIMARY KEY ( KEY_ID ),
INDEX ( KEY_REF_EMP_ID_Number ),
FOREIGN KEY (KEY_REF_EMP_ID_Number) REFERENCES Employee_Table( EMP_ID_Number )
)type=InnoDB

DROP TABLE IF EXISTS ACL_Table;

CREATE TABLE ACL_Table(
    ACL_Type varchar(30),
    ACL_Description TEXT,
    Primary Key (ACL_Type)
) 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

4.1 Testing Methods

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.

4.1.1 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.

4.1.2 Active Testing

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

Component Testing

Each component that utilizes the database is tested by creating "dummy" data which is loaded directly into the database backend. The each database interface screen is tested by logging into YardSale client and assuring that each dataset is loaded correctly. Inserts into the database are verified through a third-party database interface tool.

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. These tests verify that all components are able to store, copy, and construct data. The mains also run simple database interface tests to verify that all of the database tables can be returned correctly. If any of these tests fail the system will report the exact location of the failure by the use of function macros.

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.

Program Flow Tracking

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.

4.1.3 Functional Requirements Testing

- Point of Sale Related Data Testing

Testing of the functional requirements for YardSale is done via a hybrid system of active and passive methods.

- **Inventory Data**

Inventory data is represented by the class YardInvType. This class contains a testing main that verifies that all attributes of an inventory item (such as description, price, and quantity which are implemented as member variables) can be assigned, copied, and modified. If any of these checks fail the build system will report an error.

- * **Add a product to the inventory**

The YardDatabase class contains a testing main which connects to a "dummy" testing database. If the connection succeeds, the method tries to add a YardInvType to the database. The result of this attempt is reported by the build system.

- * **Edit an existing inventory item**

The YardDatabase class contains a testing main which connects to a "dummy" testing database. If the connection succeeds, the method tries to modify an inventory item that has been successfully added by the previous testing function. This is accomplished by (successfully) retrieving an item from the database, modifying that retrieved YardInvType and then sending that modified object to the database for modification. The result of this attempt is reported by the build system.

- * **Search for an inventory item by various criteria**

The YardDatabase class contains a testing main which connects to a "dummy" testing database. Once successfully connected the method will search for pre-loaded items by SKU number and description. If these results are incorrect the build system will report an error.

- * **Remove an inventory item**

The YardDatabase testing main will add a test item to the database and then try to "delete" that item. After the "delete" command is sent, the method will try to retrieve that deleted item. If these retrieve works then the method will fail and report this to the build system.

- * **Add an inventory item to a transaction**

The YardTransType is the class representation of a YardSale transaction. The testing main of the YardTransType creates a temporary transaction object and tries to add, remove, and modify items in the transaction. If any of these operations fail, the build system will report an error.

– Customer Data

The object representation of a customer is the YardCustType. This type contains a testing main which creates a YardCustType object and tests to see that the object can be copied, assigned, and modified. Each of these operations are verified independently by the method and any errors are reported to the build system.

- * **Add and remove customers**

The YardDatabase testing main tries to add a YardCustType to the the database and then removes that type from the database. If these operations succeed the method will pass.

- * **Search for customers by various criteria**

The YardDatabase class contains a testing main which connects to a "dummy" testing database. Once successfully connected the method will search for pre-loaded items by name and date. If these results are incorrect the build system will report an error.

- * **Edit customer information**

The YardDatabase class contains a testing main which connects to a "dummy" testing database. If the connection succeeds, the method tries modify a customer that has been successfully added by the previous testing function. This is accomplished by (successfully) retrieving an customer from the database, modifying that retrieved YardCustType and then sending that modified object to the database for modification. The result of this attempt is reported by the build system.

- * **Associate a customer with a transaction**

The YardTransType is the class representation of a YardSale transaction. The testing main of the YardTransType creates a temporary transaction object and tries to add, remove, and modify a customer for the transaction. If any of these operations fail, the build system will report an error.

– Employee Data

Employees are represented by the YardEmployeeType. The testing main of the YardEmployeeType class tests that a test object is creatable, assignable, modifiable, and copyable.

- * **Add and remove employees**

The YardDatabase class tries to add and remove temporary YardEmployeeType objects and reports the success or failure of these attempts.

- * **Associate an access level with each employee**

The YardEmployeeType testing main assigns an access level with a temporary object of type YardEmployeeType and then tests to verify that the access level is assignable, copyable, and modifiable.

- * **Associate an employee with a transaction**

The YardTransType testing main tests to verify that a YardEmployeeType can be added to the transaction object and reports the success or failure to the build system.

- * **Associate Login information with each employee**

The login information for an employee is stored in the database backend in a platform dependant way and is verified by the database control program (such as MySQL CC).

– Transaction Data

Transaction data is stored by the database and therefore is tested in the YardDatabase testing main.

- * **Add new transaction information**

The YardDatabase testing main verifies that a YardTransaction can be added to the database by creating a temporary object of type YardTransaction and adding that transaction to the database. The method then tries to retrieve the just-added transaction and will fail if the transaction cannot be returned.

- * **Search for transactions by various criteria**

The YardDatabase testing main loads various transaction data into the database and then tests to ensure that those items can be returned by searching by transaction ID, date, customer, and employee.

- * **Associate values with a transaction**

The testing main of the YardTransType verifies that information such as inventory items, a customer, and an employee can be added to the database.

- **Provide Detailed Reporting**

Reporting is verified by inspection after a report is generated by the database. A control set of data will be used to ensure that the reporting subroutine returns the correct data.

- **Allow for User Login with Differing Access Levels**

Each access level is tested by creating testing accounts at the Administrator, Manager, and Associate level. Each user will then be used to log into the database. If the user logs in successfully, the program will be manually tested to ensure that Associates cannot modify information such as other employee records and inventory items.