PHASE 2 DOCUMENT FOR THE POST SYSTEM
SOFTWARE ENGINEERING
PROF. BLESSING
11-8-99

Team Members:
Josh Fedke – Project Leader
Kevin Kaddatz – MFC/GUI Expert
Brenden Morgenthaler – Database Expert
Ryan Champlin – Rational Rose Expert

TABLE OF CONTENTS

PHASE 2 ANALYSIS SECTION 3
OVERVIEW 4
DEVELOPMENT TEAM INFORMATION 4
SYSTEM REQUIREMENTS 4
System Functions - Cycle One 4
System Functions - Cycle Two 4
System Attributes 5
POST USE CASE SCENARIOS 5
Buy Items 5
Section: Main 5
Section: Pay by Cash 6
Section: Pay by Credit 6
Section: Pay by Check 7
Section: Pay by Store Credit 7
Refund Purchased Items 7
Section: Main 7
Section: Cash Refund 8
Section: Credit Refund 8
Login 9
Section: Main 9
Change Administrator Only Options 9
Section: Main 9
Initialize the POST System 9
Section: Main 9
Void Item 10
Section: Main 10
POST CONCEPTUAL MODEL 11
SEQUENCE/COLLABORATION DIAGRAMS 12
Login Sequence 12
Login Collaboration 12
Change Administrator Only Options Sequence 13
Change Administrator Only Options Collaboration 13
Buy Items Sequence 14
Buy Items Collaboration 15
Refund Purchased Items Sequence 16
Refund Purchased Items Collaboration 17
PHASE 2 DESIGN SECTION 18
Design Class Diagram 19
SYSTEM CONTRACTS 20
Contracts for the Buy Items Use Case 20
Contracts for the Login Use Case 23
Contracts for the Refund Items Use Case 23
PHASE 2 IMPLEMENTATION SECTION 26
INTRODUCTION 27
IMPLEMENTATION DETAILS AND COMPLICATIONS – PHASE 1 27
MFC/GUI Related Issues 27
Database Issues 27
Rational Rose Issues 27
Visibility Issues 27
OUTSTANDING REQUIREMENTS AND ISSUES FROM PHASE 1 28
TEST PLANS FOR PHASE 1 29
Cashier Login Test Plan 29
Buy Items Test Plan 29
Return Items Test Plan 29

Failure Details 30
IMPLEMENTATION DETAILS AND COMPLICATIONS – PHASE 2 31
MFC/GUI Related Issues 31
Rational Rose Issues 31
OUTSTANDING REQUIREMENTS AND ISSUES FROM PHASE 2 32
TEST PLANS FOR PHASE 2 33
Cashier Login Test Plan 33
Buy Items Test Plan 33
Return Items Test Plan 33
Failure Details 34
UPDATED DESIGN DIAGRAMS 35
Updated Class Diagram 35
Rational Rose Reverse-Engineered Model Overview – Phase 1 36
Rational Rose Reverse-Engineered Model Overview – Phase 2 37
THE GUI FOR THE POST SYSTEM 38
Cashier Login Form 38
Sale Form (Used for Buy Items Use Case Implementation) 39
Return Form (Used for Refund Items Use Case Implementation) 40
MODULE DESCRIPTIONS 41
APPENDIX A – POST CODE LISTING 43
Cashier.h 44
Cashier.cpp 45
CashierSet.h 47
CashierSet.cpp 49
Customer.h 51
Customer.cpp 52
CustomerSet.h 54
CustomerSet.cpp 56
LineItem.h 58
LineItem.cpp 59
LineItemSet.h 64
LineItemSet.cpp 66
Product.h 68
Product.cpp 69
ProductSet.h 73
ProductSet.cpp 75
Transaction.h 77
Transaction.cpp 79
TransactionSet.h 85
TransactionSet.cpp 87
PostTerminal.h 89
PostTerminal.cpp 92
LoginView.h 104
LoginView.cpp 106
SaleView.h 109
SaleView.cpp 112
ReturnView.h 121
ReturnView.cpp 123
PostConstants.h 130

PHASE 2 ANALYSIS SECTION

OVERVIEW

The purpose of this project is to create a general Point of Sale Terminal or POST system. This system can then be further customized to suit each customer's needs. This document contains the requirements and features listed for the general POST. Note: Phase 2 changes are shown in blue text.

DEVELOPMENT TEAM INFORMATION

The team working on this project is as follows.

- Kevin Kaddatz – MFC/GUI Expert
- Josh Fedke - Project Leader
- Ryan Champlin – Rational Rose Expert
- Brenden Morgenthaler – Database Expert

SYSTEM REQUIREMENTS

System Functions - Cycle One

Ref. #	Function	Category
R1.1	Record the underway (current) transaction (sale or return).	Evident
R1.2	Calculate the current transaction total, including tax and coupon calculations.	Evident
R1.3	Capture transaction item information from a bar code using a bar code scanner, or manual entry of a product code (UPC).	Evident
R1.4	Reduce/Increase inventory quantities when a transaction is finalized.	Hidden
R1.5	Log completed transactions.	Hidden
R1.6	Cashier must log in with a numerical employee number to use the system.	Evident
R1.7	Provide a persistent storage mechanism.	Hidden
R1.8	Display description and price of the current item.	Evident
R1.9	Handle all types of cash transactions.	Evident

NOTE: FUNCTIONALITY FROM PHASE ONE SHOWN IN RED WILL SPILL OVER INTO PHASE 2.

System Functions - Cycle Two

Ref. #	Function	Category
R2.1	Handle credit card transactions, capturing credit card information from a card reader or by manual entry, and authorizing the transaction with the store's (external) credit authorization service via a modem connection.	Evident
R2.2	Handle check payments, capturing driver's license by manual entry, and authorizing payment with the store's (external) check authorization service via a modem connection.	Evident
R2.3	Handle store credit transactions, authorizing the credit through the store's (internal) store credit record database.	Evident
R2.4	Log credit payments to the Accounts Receivable system, since the credit authorization service owes the store the payment amount.	Hidden
R2.5	Handle and system administrator options.	Evident
R2.6	Manager responsible for startup of the POST system.	Evident
R2.7	Cashier has ability to void an item after entry.	Evident

NOTE: The word "transaction" includes purchases and returns since both must be handled by the POST system.

System Attributes

Ref. #	Attribute	Details and Boundary Constraints
A1.1	Response time	(*boundary constraint*) When displaying an item within a transaction, the item must appear with information within 5 seconds.
A1.2	Interface metaphor	(*detail*) Forms-metaphor windows and dialog boxes (*detail*) Maximize for easy keyboard navigation rather than pointer navigation
A1.3	Operating system platforms	(*detail*) 32-bit Microsoft Windows Platforms (NT, Win 9x, 2000)

POST USE CASE SCENARIOS

Buy Items

Section: Main

Use Case: Buy Items
Actors: Customer (initiator), Cashier
Purpose: Capture a sale and its payment.
Overview: A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects a payment, which may be authorized. On completion, the Customer leaves with the items.
Type: Primary and Essential
Cross References:
Functions: R1.1, R1.2, R1.3, R1.7, R1.8, R1.9, R2.1, R2.2, R2.3, R2.4
Use Cases: The Cashier must have completed the *Log In* use case.
The Manager must have completed the *Initialize the POST System* use case.

Typical Course of Events	
Actor Action	**System Response**
1. This use case begins when a Customer arrives at the POST checkout with one or more items to purchase.	
2. The Cashier records each item. If there is more than one of the same items, the Cashier can enter the quantity as well.	**3.** Determine the item price and add the item to the running sales transaction. The description and price of the current item are presented.
4. On completion of item entry, the Cashier indicates to the POST system that item entry is complete.	**5.** Calculates and presents the sale total.
6. The Cashier tells the Customer the total.	
7. Customer chooses payment type: a. If cash payment, see section *Pay by Cash.* b. If credit payment, see section *Pay by Credit.* c. If check payment, see section *Pay by Check.* d. If store credit payment, see section *Pay by*	

Store Credit.	
	8. Logs the completed sale.
	9. Updates inventory levels.
	10. Generates a receipt.
11. The Cashier gives the receipt to the Customer.	

Alternative Courses

- Line 2: Invalid item identifier entered. Indicate error.
- Line 7: Customer could not pay. Cancel sales transaction.

Section: Pay by Cash

Typical Course of Events	
Actor Action	**System Response**
1. The Customer gives cash payment possibly greater than the sale total.	
2. The Cashier records the cash tendered.	**3.** Presents the balance due back to the Customer.
4. The Cashier deposits the cash received and extracts the balance owing. The Cashier gives the balance owing to the customer.	

Alternative Courses

- Line 1: Customer does not have sufficient cash. May cancel the sale or initiate another payment method.
- Line 4: Cash drawer does not contain sufficient cash to pay the balance. Cashier requests additional cash from the supervisor or ask Customer for different payment amount or method.

Section: Pay by Credit

Typical Course of Events	
Actor Action	**System Response**
1. The Customer communicates their credit information for the credit payment.	**2.** Generates a credit payment request and sends it to an external Credit Authorization Service (CAS).
3. CAS authorizes the payment.	**4.** Receives a credit approval from the CAS.
	5. Posts (records) the credit payment and approval information to the Accounts Receivable system. The CAS owes money to the Store; therefore Accounts Receivable must track it.
	6. Displays authorization success message.

Alternative Courses

- Line 3: Credit request denied by CAS. Suggest different payment method or cancel transaction.

Section: Pay by Check

Typical Course of Events	
Actor Action	System Response
1. The Customer writes a check and identifies self.	
2. Cashier records identification information and requests check payment authorization.	3. Generates a check payment request and sends it to an external Check Authorization Service.
4. Check Authorization Service authorizes the payment.	5. Receives a check approval reply from the Check Authorization Service.
	6. Indicates authorization success.

Alternative Courses

- Line 4: Check Authorization Service denies check request. Suggest different payment method or cancel the transaction.

Section: Pay by Store Credit

Typical Course of Events	
Actor Action	System Response
1. The Customer shows store credit to Cashier and the Cashier enters the store credit ID number.	2. Verifies the store credit by checking if the number exists in the database and it isn't expired.

Alternative Courses

- Line 2: The store credit is not valid. Cashier keeps the invalid document and alerts management to the situation.

Refund Purchased Items

Section: Main

Use Case: Refund Purchased Items
Actors: Customer (initiator), Cashier
Purpose: Capture a sale and its payment.
Overview: A Customer arrives at a checkout with items to return that were purchased previously. The Customer presents the receipt and the Cashier records transaction information. The Customer receives a refund based on the original tender used to purchase the items. On completion, the Customer leaves with a return receipt and returned tender.
Type: Primary and Real
Cross References:
Functions: R1.1, R1.2, R1.3, R1.4, R1.5, R1.7, R1.8, R1.9, R2.1, R2.2, R2.3, R2.4
Use Cases: Cashier must have completed the *Log In* use case.
The Manager must have completed the *Initialize the POST System* use case.

Typical Course of Events	
Actor Action	**System Response**
1. This use case begins when a Customer arrives at the POST checkout with an item to return.	
2. The Cashier records transaction information on the receipt to check for validity of the transaction.	**3.** Checks the transaction date against records. Indicates success in finding the transaction.
4. The Cashier records each item being returned. If there is more than one of an item, the Cashier can enter the quantity as well.	**5.** Determine the item price and add the item information to the running transaction. The description and price of the current item are

	presented.
6. On completion of item entry, the Cashier indicates to the POST that item entry is complete.	**7.** Calculates and presents the refund amount.
8. The Cashier determines the return tender based on the original receipt: a. If cash or check payment, see section *Cash Refund*. b. If credit payment, see section *Credit Refund.* Customer is first given the option of receiving store credit for the product. See section *Store Credit Refund*.	
	9. Logs the completed refund.
	10. Updates inventory levels.
	11. Generates a return receipt.
12. The Cashier gives the receipt to the customer as proof of the return transaction.	

Alternative Courses

- Line 3: Invalid transaction entered. Cashier informs the customer and the return is aborted.
- Line 4: Invalid item identifier entered (or item was not on the original receipt). Indicate error and allow for re-entry.

Section: Cash Refund

Typical Course of Events	
Actor Action	**System Response**
1. The Cashier enters the amount the Customer will be refunded in cash.	**2.** Calculates the refund balance.
3. The Cashier gives the Customer the cash from the register.	

Alternative Courses

- Line 1: There is not enough cash in the drawer to refund the Customer's money. Cashier will alert the Manager and suggest the store credit option to the Customer. System will also suggest this option.

Section: Credit Refund

Typical Course of Events	
Actor Action	**System Response**
1. The Customer gives the Cashier the original credit card. The Cashier communicates to the POST system the card information and amount to be refunded.	**2.** Displays successful match of the card number to the original transaction.
3. Credit Authorization Service (CAS) is made aware of the "credit" to the cardholder.	**4.** Receives approval to credit the card from the CAS.
	5. Records the refund and approval reply information to the Accounts Receivable system. This allows A/R to track the amount owed by the CAS.
	6. Displays authorization success message.

Alternative Courses

- Line 2: The credit card is not the same. Cashier asks for the correct card.
- Line3: The approval is not given. The account may be inactive. Cashier relays this information to the customer.

Login

Section: Main

Use Case: Login
Actors: Cashier (initiator)
Purpose: Allow the Cashier to logon to the POST terminal to complete transactions.
Overview: The Cashier arrives at the checkout and enters a password. The system validates the password and associates the password with a specific user.
Type: Primary and Essential
Cross References:
Functions: R1.6
Use Cases: This use case assumes the Manager has completed the *Initialize the POST System* use case.

Typical Course of Events	
Actor Action	**System Response**
1. Cashier enters a password at the terminal login screen.	**2.** Verifies the password and logs the Cashier into the system. Displays initial transaction screen.

Alternate Courses

- Line 2: Password entered was incorrect. System will not allow access and continue to prompt for a valid password.

Change Administrator Only Options

Section: Main

Use Case: Change Administrator Only Options
Actors: Manager (initiator)
Purpose: To allow for the addition and removal of items from the store database.
Overview: This use case begins when the Manager logs into the POST and indicates that they wish to change administrator options for the POST system. The POST system validates the Manager and adjusts the database accordingly.
Type: Secondary and Real
Cross References:
Functions: R2.5

Initialize the POST System

Section: Main

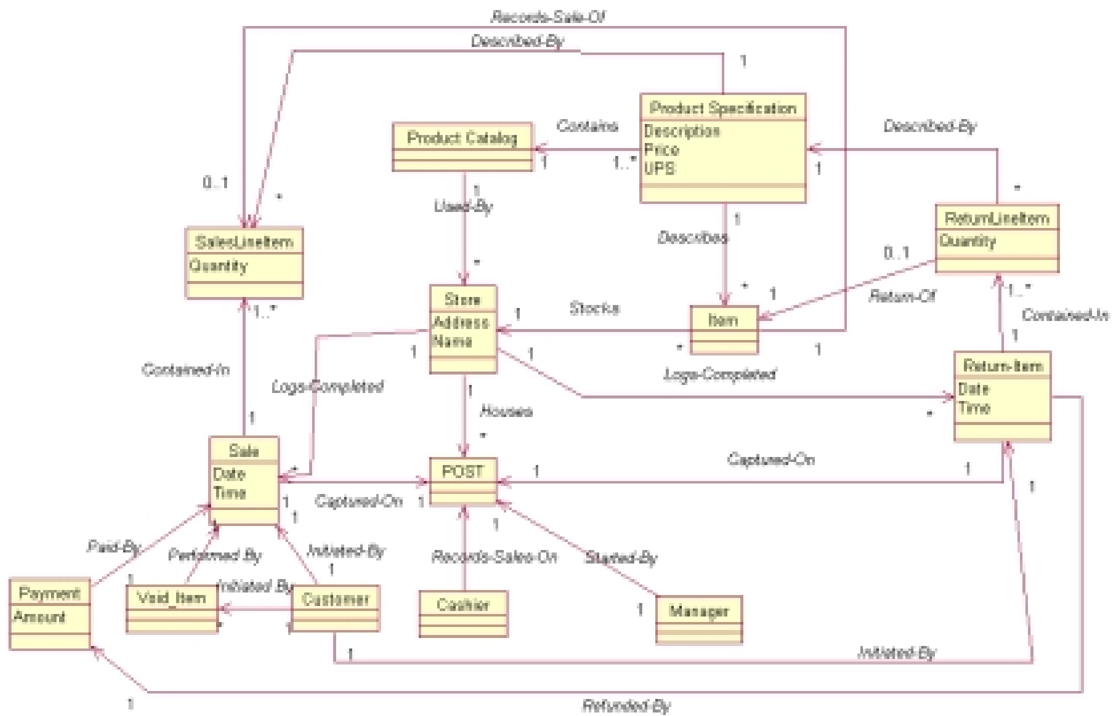
Use Case: Initialize the POST System
Actors: Manager (initiator)
Purpose: To allow only a Manager to initialize the POST system at the start of the business day.
Overview: This use case begins when the Manager turns on the power to the POST hardware and inserts the manager key. The system will boot and bring up the initial login screen seen by all users. The Manager will also place initial cash into the cash drawer.
Type: Primary and Essential
Cross References:
Functions: R2.6

Void Item

Section: Main

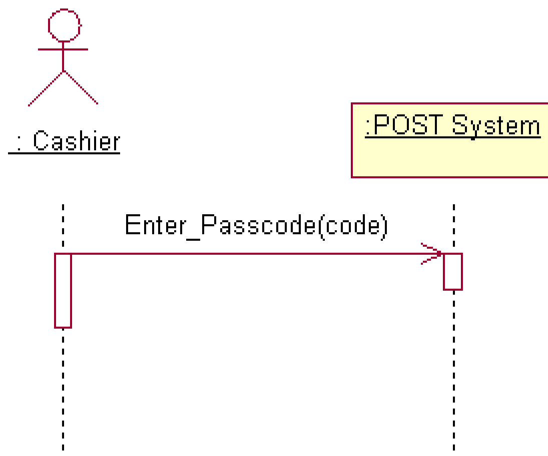
Use Case: Void an item previously entered by the Cashier
Actors: Cashier (initiator), Customer
Purpose: To allow only a Cashier to void an item entered into the POST system.
Overview: This use case begins when the Customer indicates they do not want to purchase an item the Cashier has entered into the POST system. The cashier selects the item with a mouse form the running receipt and presses the void button. The item is then removed from the receipt and deleted from the transaction
Type: Primary and Essential
Cross References:
Functions: R2.8

POST CONCEPTUAL MODEL

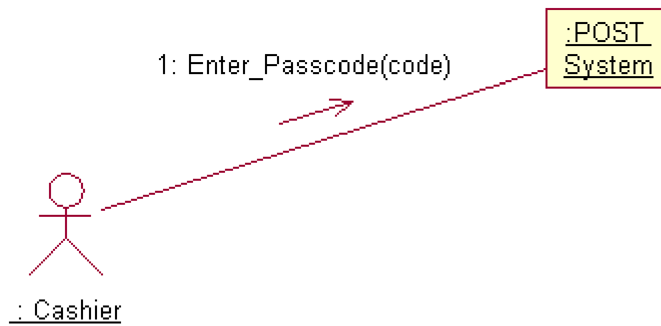


SEQUENCE/COLLABORATION DIAGRAMS

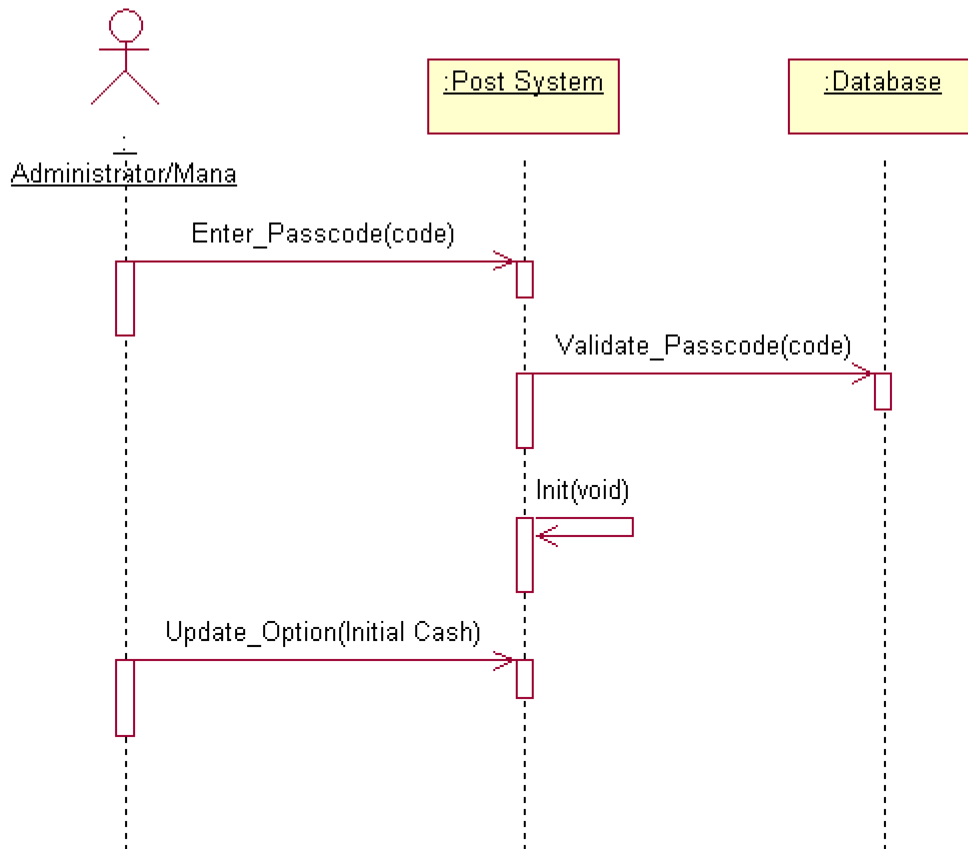
Login Sequence



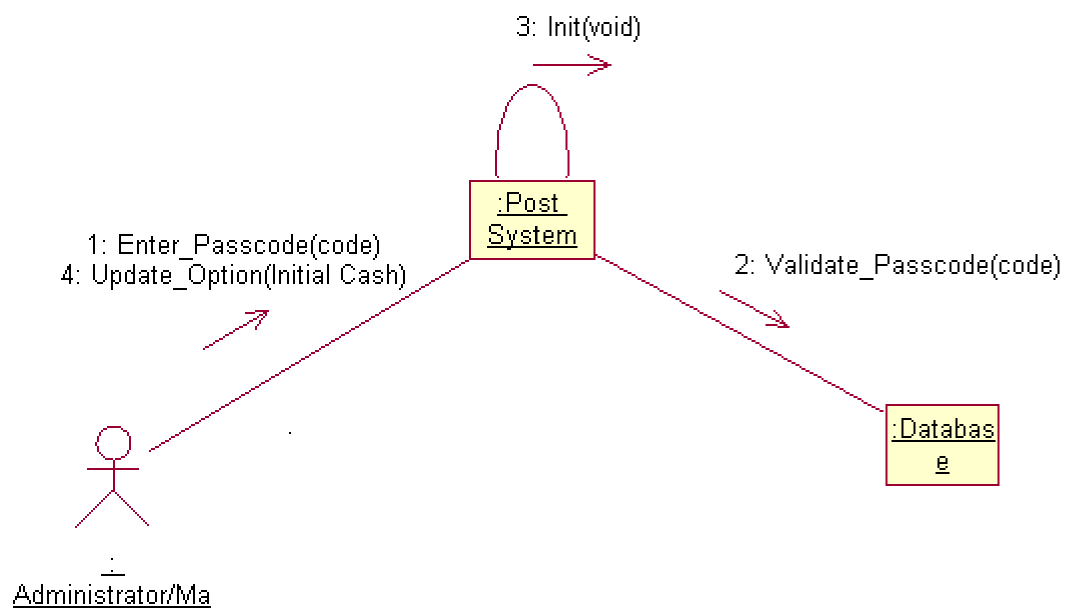
Login Collaboration



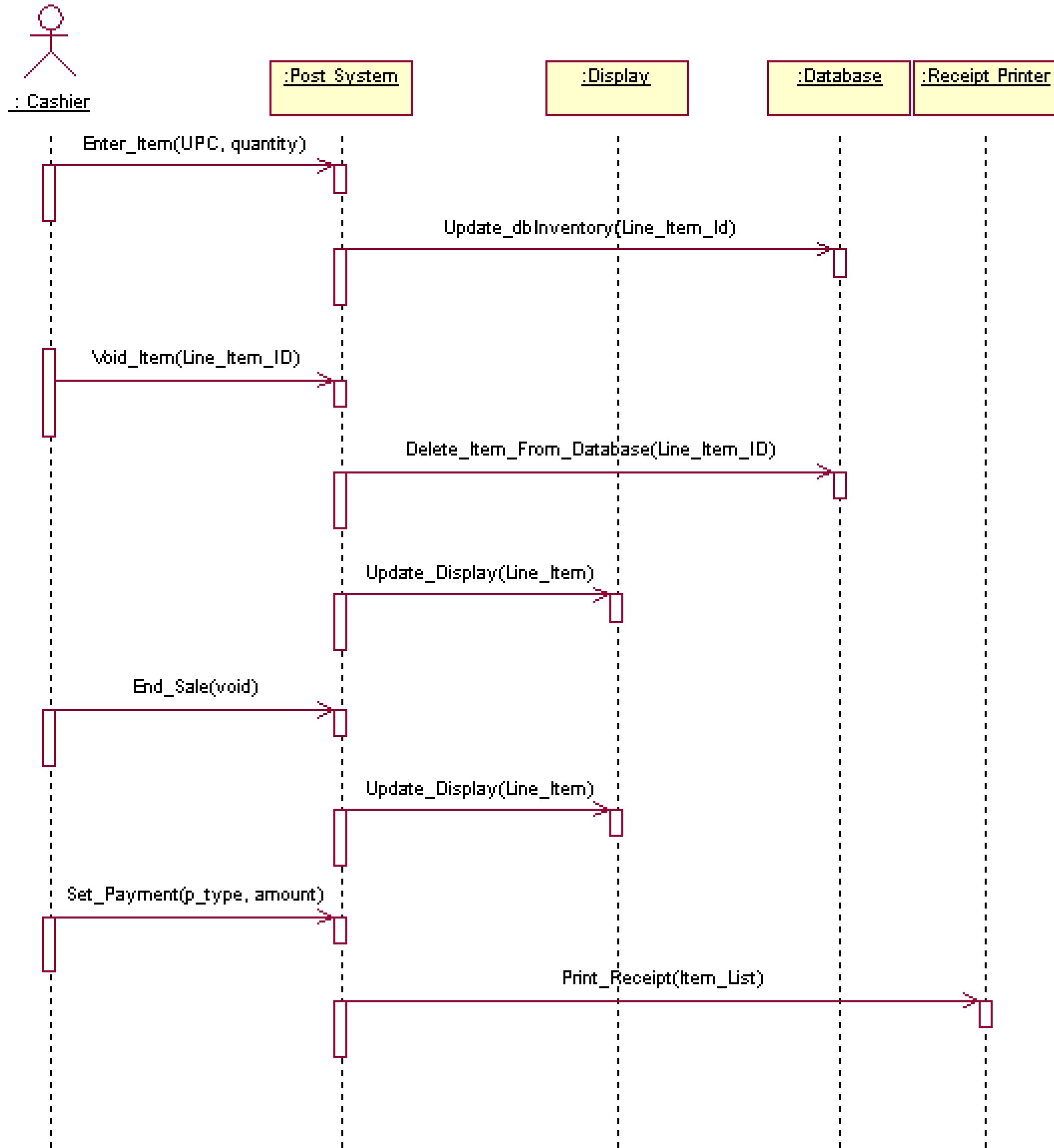
Change Administrator Only Options Sequence



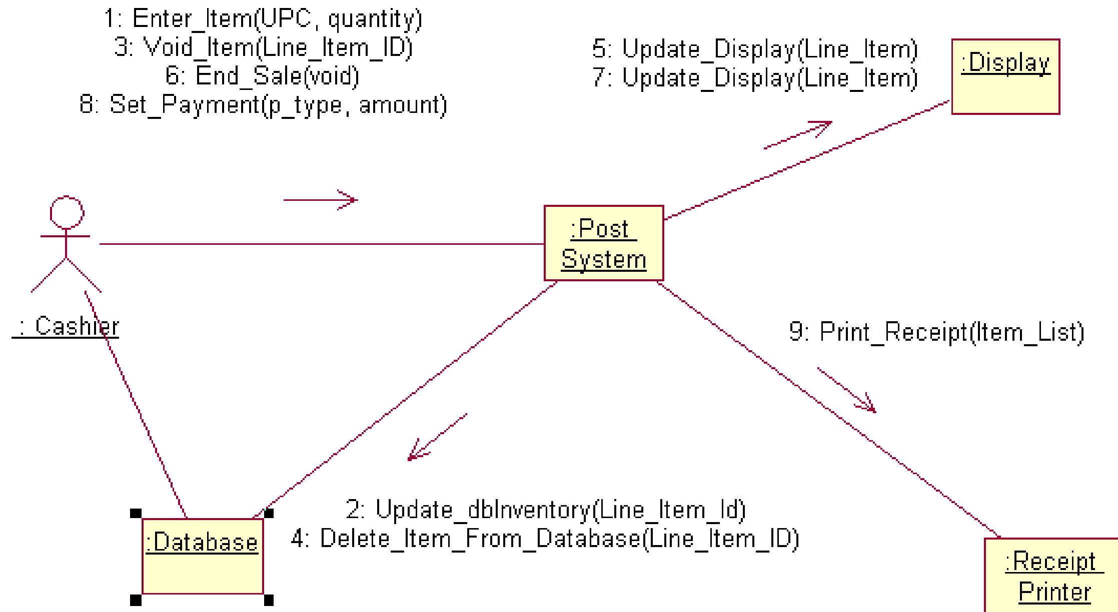
Change Administrator Only Options Collaboration



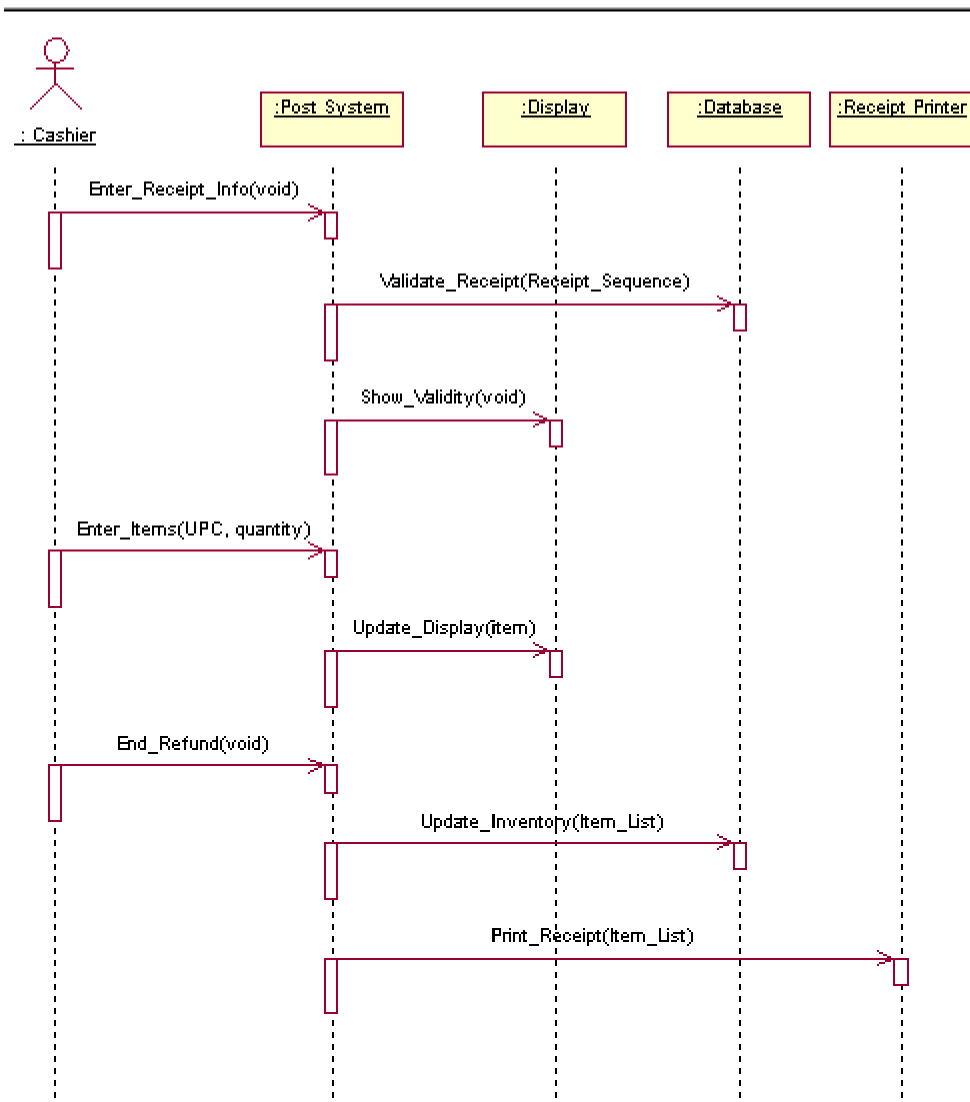
Buy Items Sequence



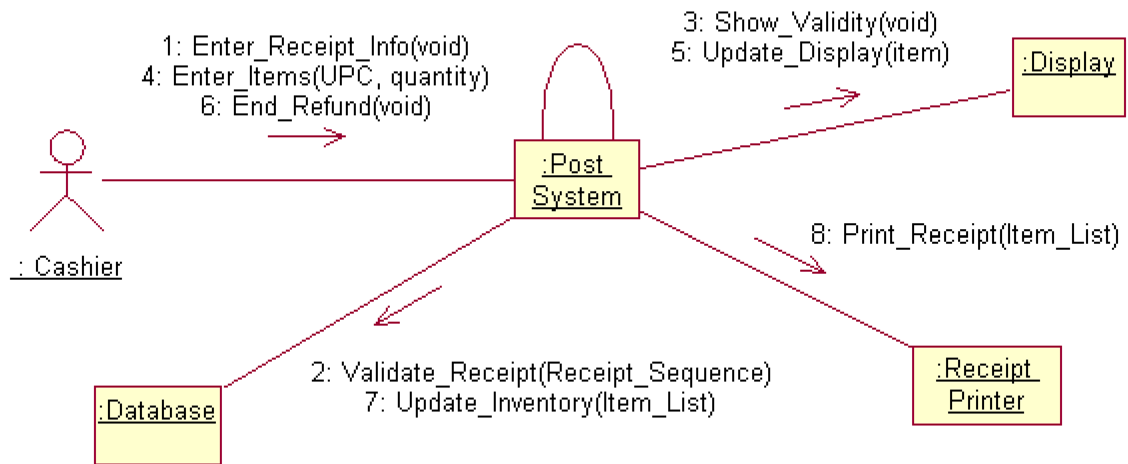
Buy Items Collaboration



Refund Purchased Items Sequence

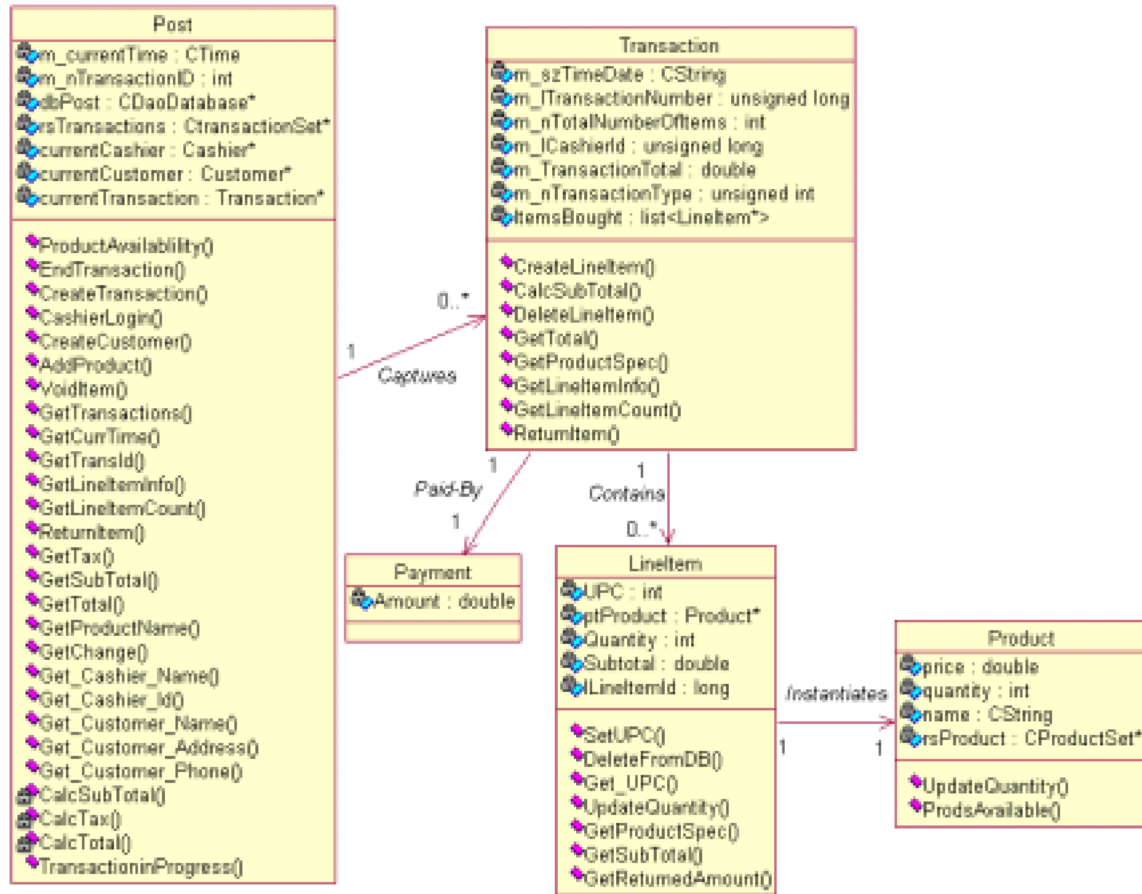


Refund Purchased Items Collaboration



PHASE 2 DESIGN SECTION

Design Class Diagram



SYSTEM CONTRACTS

Contracts for the Buy Items Use Case

Contract

Name: Enter_Item(upc : string, quantity : integer)
Responsibilities: Enter (record) sale of an item and add it to the transaction.
Type: System
Cross References: System Functions: R1.1, R1.3
Notes: Use DAO database access
Exceptions: If the UPC is not valid, indicate that it was an error
Output:
Pre-conditions: UPC is available and taken from the GUI
Post-conditions:

- If a new transaction, a *Transaction* was created (instance creation).
- If a new transaction, the new *Transaction* was associated with the POST (association formed).
- A *TransactionLineItem* was created (instance creation).
- The *TransactionLineItem* was associated with the *Transaction* (association formed).
- *TransactionLineItem.quantity* was set to *quantity* (attribute modification).
- The *TransactionLineItem* was associated with a *ProductSpecification*, based on UPC match (association formed).

Contract

Name: Update_Display(item)
Responsibilities: Update the transaction log with the current item entered.
Type: System
Cross References: System Functions: R1.9
Notes:
Exceptions:
Output: Update transaction log on GUI
Pre-conditions: A new *TransactionLineItem* was created
Post-conditions:

- The *TransactionLineItem* was associated with the transaction log.

Contract

Name: End_Sale(void)
Responsibilities: Record that it is the end of entry of sale items and display the Transaction total.
Type: System
Cross References: System Functions: R1.2
Notes:
Exceptions: If a Transaction is not underway, indicate that it was an error.
Output: Update the sub-total, sales total, tax, and coupon boxes.
Pre-conditions: *Transaction* is underway with valid *TransactionLineItem*(s).
Post-conditions:

- *Transaction.bIsComplete* was set to TRUE (attribute modification).

Contract

Name: Set_Payment(p_type : integer, amount : float)
Responsibilities: Set the payment type member variable and adjust the current balance based on *amount*.
Type: System
Cross References: System Functions: R1.2, R1.10
Notes:
Exceptions: If the payment type is not valid, indicate an error.
Output:
Pre-conditions: The cashier must have indicated the end of the current *Transaction*.
Post-conditions:

- *Transaction.PaymentType* is adjusted based on *p_type* (attribute modification).
- *Transaction.Balance* is adjusted based on *amount* (attribute modification)

Contract

Name: Update_dbInventory(Line_Item_Id : int)
Responsibilities: Update the inventory database with the items from the current *Transaction*.
Type: System
Cross References: System Functions: R1.4, R1.5, R1.7
Notes: Use DAO database access
Exceptions: If the database is corrupt or unavailable, indicate an error.
Output:
Pre-conditions: The *Transaction* must be completely finished (balance is zero).
Post-conditions:

- *TransationLineItem*(s) are associated with the transaction database (association formed).
- Inventory database is adjusted based on quantities from *TransactionLineItem*(s).

Contract

Name: Delete_Item_From_Database(Line_Item_Id : int)
Responsibilities: Update the inventory database with the items from the current *Transaction*.
Type: System
Cross References: System Functions: R1.4, R1.5, R1.7
Notes: Use DAO database access
Exceptions: If the database is corrupt or unavailable, indicate an error.
Output:
Pre-conditions: An item has been entered using Enter_Item(UPC, quantity).
Post-conditions:

- *TransationLineItem*(s) are associated with the transaction database (association formed).
- Inventory database is adjusted based on quantities from *TransactionLineItem*(s).

Contract

Name: Void_item(Line_Item_Id : int)
Responsibilities: Update the inventory database with the items from the current *Transaction*.
Type: System
Cross References: System Functions: R1.4, R2.7
Notes: Use DAO database access
Exceptions: If the database is corrupt or unavailable, indicate an error.
Output:

Pre-conditions: The item must have been entered using Enter_Item(UPC, quantity).
Post-conditions:

- Item is removed from the database (association formed).
- Item is removed from the receipt.

Contract

Name: Print_Receipt(Item_List : CList)
Responsibilities: Print a list of the current items in the *Transaction*, date, time, store location, employee ID, total, type of payment, and change returned.
Type: System
Cross References: System Functions: R1.8
Notes:
Exceptions: Printer is offline so indicate error.
Output: Paper receipt is printed on the receipt printer.
Pre-conditions: The *Transaction* must be completely finished (balance is zero).
Post-conditions:

- System returns to log in screen.
- *Transaction* object is destroyed (instance deletion).

Contracts for the Login Use Case

Contract

Name:	Enter_Passcode(code : string)
Responsibilities:	Check the passcode against the user database to assure user clearance.
Type:	System
Cross References:	System Functions: R1.6, R1.8
Notes:	Use DAO database access
Exceptions:	If the Cashier is not valid, indicate an error.
Output:	
Pre-conditions:	A Manager has initialized the POST system.
Post-conditions:	

- Item entry screen is presented.

Contracts for the Refund Items Use Case

Contract

Name:	Enter_Receipt_Info(void)
Responsibilities:	Get Receipt information
Type:	System
Cross References:	System Functions: R1.1, R1.3
Notes:	
Exceptions:	
Output:	
Pre-conditions:	Transaction number is available and taken from the GUI
Post-conditions:	

- The GUI contains the receipt transaction number
- *Transaction.szNumber* is updated (attribute modification)

Contract

Name:	Validate_Receipt(Receipt_Sequence : int)
Responsibilities:	Check the entered information vs. the database information
Type:	System
Cross References:	System Functions: R1.1, R1.3
Notes:	Use DAO database access
Exceptions:	
Output:	
Pre-conditions:	The transaction number is available from the GUI
Post-conditions:	

- *Transaction.bValid* is set

Contract

Name: Show_Validity(void)
Responsibilities: Output the validity of the transaction to the POST GUI.
Type: System
Cross References: System Functions: R1.1, R1.3
Notes:
Exceptions:
Output: Shows validity on the GUI
Pre-conditions:
Post-conditions:

- GUI is updated

Contract

Name: Enter_Item(upc : string, quantity : integer)
Responsibilities: Enter (record) refund of an item and add it to the Transaction.
Type: System
Cross References: System Functions: R1.1, R1.3
Notes: Use DAO database access
Exceptions: If the UPC is not valid, indicate that it was an error
Output:
Pre-conditions: UPC is available and taken from the GUI
Post-conditions:

- If a new transaction, a *Transaction* was created (instance creation).
- If a new transaction, the new *Transaction* was associated with the POST (association formed).
- A *TransactionLineItem* was created (instance creation).
- The *TransactionLineItem* was associated with the *Transaction* (association formed).
- *TransactionLineItem.quantity* was set to *quantity* (attribute modification).
- The *TransactionLineItem* was associated with a *ProductSpecification*, based on UPC match (association formed).

Contract

Name: Update_Display(item)
Responsibilities: Update the transaction log with the current item entered.
Type: System
Cross References: System Functions: R1.9
Notes:
Exceptions:
Output: Update transaction log on GUI
Pre-conditions: A new *TransactionLineItem* was created
Post-conditions:

- The *TransactionLineItem* was associated with the transaction log.

Contract

Name: End_Refund(void)
Responsibilities: Record that it is the end of entry of refund items and display the refund total.
Type: System
Cross References: System Functions: R1.2
Notes:

Exceptions: If a transaction is not underway, indicate that it was an error.
Output: Update the sub-total, sales total, tax, and coupon boxes.
Pre-conditions: *Transaction* is underway with valid *TransactionLineItem*(s).
Post-conditions:

- *Transaction.bIsComplete* was set to TRUE (attribute modification).

Contract

Name: Update_Inventory(Item_List : CList)
Responsibilities: Update the inventory database with the items from the current *Transaction*.
Type: System
Cross References: System Functions: R1.4, R1.5, R1.7
Notes: Use DAO database access
Exceptions: If the database is corrupt or unavailable, indicate an error.
Output:
Pre-conditions: The *Transaction* must be completely finished (balance is zero).
Post-conditions:

- *TransationLineItem*(s) are associated with the transaction database (association formed).
- Inventory database is adjusted based on quantities from *TransactionLineItem*(s).

Contract

Name: Print_Receipt(Item_List : CList)
Responsibilities: Print a list of the current items in the *Transaction*, date, time, store location, employee ID, total, type of payment, and change returned.
Type: System
Cross References: System Functions: R1.8
Notes:
Exceptions: Printer is offline so indicate error.
Output: Paper receipt is printed on the receipt printer.
Pre-conditions: The *Transaction* must be completely finished (balance is zero).
Post-conditions:

- System returns to log in screen.
- *Transaction* object is destroyed (instance deletion).

PHASE 2 IMPLEMENTATION SECTION

INTRODUCTION

This section contains information about the implementation of our design for a POST system given our system requirements. We implemented a GUI with three basic screens, the login screen, the buy items screen, and the return items screen. Our storage is done with a Microsoft© Access 97 database. We implemented classes to encapsulate cashier, customer, line item, payment, and product objects. The class descriptions, implementation problems, and code can be found in the rest of this document.

IMPLEMENTATION DETAILS AND COMPLICATIONS – PHASE 1

MFC/GUI Related Issues

Our first issue came with the use of classes developed outside of our main MFC project. We created most of the classes that encapsulate database functionality outside of the main project. These were then tested individually before they were merged with the MFC project. This is where the problems began. We spent almost two days trying to figure out linker errors (102 at one point) and strange compiler errors such as "Unexpected End of File Found" for all of the .cpp files that we created. Finally, we decided to create "fresh" classes by using the Class Wizard in our main MFC project. We gave them the same names as the classes created outside of the MFC project and used many copy and paste functions to place our code into them. Suddenly, every linker and compiler error disappeared. Why? The only possible explanation seems to be with the order that files are #include'd in .cpp files. Since Class Wizard takes care of this when it creates the framework for a new class, it probably fixed the problem for us.

Database Issues

Ryan spent a lot of time researching and probe-coding during the previous phases. This led to smooth implementation of the database portion. In addition, choosing DAO to access the database during the design phase led to less research time and fewer headaches.

Rational Rose Issues

We began our implementation phase by first forward engineering our Rose project from the Analysis and Design phases. The main issue with this step was readability of the code that was generated. We had a very difficult time trying to understand what Rose had created from our interaction and class diagrams. We decided at this point that we would not use any of Rose's forward-generated code. The plan was to code the design as we saw fit and then try to learn what Rose does with the code by reverse engineering our version of the design. By using this process, it has become evident to us what Rose was doing with our original design diagrams. We also learned about how to force Rose to generate more usable code from the start. It really is necessary to put exhaustive thought and detail into the design diagrams to help the forward engineering process. Therefore, in Phase 2, we plan on implementing the additional requirements in Rose directly and forward generating this into our Phase 2 code.

Visibility Issues

We encountered the age-old programming question of when to make a variable global. As a general practice, globals are not acceptable. They increase coupling and introduce easy corruption into the system. Based on how the POST system is structured, every time the System Start-up Use Case is performed, a POST object is created. This is the one and only time this should occur. In addition, every class based on the database implementation needs to be able to see the current POST object. For these two reasons, we decided to make the Post object global. This is similar to the reasoning the MFC uses to justify a global app. An MFC project contains an instance of the project's application class that is always called theApp. This object is global to the system because every class needs to see it and there is never more than one instance of theApp throughout the execution of an MFC program.

OUTSTANDING REQUIREMENTS AND ISSUES FROM PHASE 1

The table below shows our requirements as they were originally conceived in the Analysis and Design from Phase 1.

CYCLE ONE SYSTEM REQUIREMENTS		
Ref. #	**Function**	**Category**
R1.1	Record the underway (current) transaction (sale or return).	Evident
R1.2	Calculate the current transaction total, including tax and coupon calculations	Evident
R1.3	Capture transaction item information from a bar code using a bar code scanner, or manual entry of a product code (UPC).	Evident
R1.4	Reduce/Increase inventory quantities when a transaction is finalized.	Hidden
R1.5	Log completed transactions.	Hidden
R1.6	Cashier must log in with a numerical employee number to use the system.	Evident
R1.7	Provide a persistent storage mechanism.	Hidden
R1.8	Provide inter-process and inter-system communications.	Hidden
R1.9	Display description and price of the current item.	Evident
R1.10	Handle all types of cash transactions.	Evident

Meeting these requirements proved to be quite the challenge. The idea of simply handling the Buy Items and Login Use Cases for Phase 1 (as the author of the book does) may have been a better plan.

Our plan was to implement the idea of a "transaction". This transaction could be either a sale **or** a return. Once we began to implement the design, the amount of time spent learning all of the new coding and database techniques prevented us from completing the Refund Items Use Case. It stands at about 80% complete. We still have to handle removal of the items from the database and display of the items being returned from the original transaction. Had we not run into as many MFC issues, this would have certainly been a feasible Use Case to implement in Phase 1. However, we will have to restructure the requirements for Phase 2 and they will include completion of the Refund Items Use Case.

The next major requirement (or maybe just a feature) that we seem to have neglected in our design is error-checking. At this point, most of the database access functions will simply return database access errors if incorrect information is entered. This has also lead to instability of the POST after the error is encountered. Phase 2 will make error checking an essential feature with user-friendly messages and high database stability.

The final major issue that will be handled (actually before Phase 2 implementation) is documentation. Because of time constraints, several modules are not properly documented. This will be the effort for the upcoming weekend. We will document all modules completely and uniformly. We will also be cleaning up the code structure and naming conventions. Because four different developers worked on various sections, naming conventions are not well defined or preserved. This will be completed during the weekend effort.

All other requirements are fully implemented. The degree of correctness with which they were implemented is explored in the test plans we created and ran on the following pages.

TEST PLANS FOR PHASE 1

Cashier Login Test Plan

Test Operation	Pass	Fail
1. Enter Cashier ID in *Cashier ID* edit box.	X	
a. Enter incorrect Password.	X	
b. Verify Error dialog, prompting for re-entry of password appears.	X	
c. Enter correct Password.	X	
d. Press **Sale.**	X	
e. Verify that system has logged in the user by bringing up the new transaction window.	X	
f. See that correct cashier ID and Cashier Name is in the *ID* and *Name* fields.	X	
2. Return to the Login screen by restarting the POST. Enter a Password, leaving *Cashier ID* field blank.	X	
a. Verify that an error appears prompting for re-entry of the Cashier ID.	X	
b. Enter a correct Password.	X	
c. Press the **Sale** button.	X	
d. Verify that system has logged in the user by bringing up the new transaction window.	X	
e. See that correct cashier ID and Cashier Name is in the *ID* and *Name* fields.	X	

Buy Items Test Plan

NOTE: Assume the Cashier is logged in and has pressed the **Sale** button.

Test Operation	Pass	Fail
1. Enter a Customer ID.	X	
2. See that the *Name* and *Address* fields are updated.	X	
3. Make sure the cursor appears in the *UPC* entry box.	X	
4. Enter a UPC.	X	
a. If a UPC is invalid, make sure a re-entry dialog appears.		X
5. See that receipt information and subtotals are updated.	X	
a. If this is the first item, see that a date, time, and cashier are the first lines in the receipt, and verify accuracy of these fields.	X	
6. Enter two or more items, verifying that previously mentioned fields are updating correctly.	X	
7. Click **End Sale**, and verify that the receipt is updated with current customer's information. Make sure that *subtotal*, *tax*, and *total* are updated to correct amounts.		X
8. See that the *Amount Tendered* Edit box gets focus.	X	
9. Enter an amount greater than or equal to the purchase total.	X	
10. Verify that the pop-up dialog shows the correct amount owed to customer.	X	

Return Items Test Plan

NOTE: Assume the Cashier is logged in and has pressed the **Return** button.

Test Operation	Pass	Fail
1. Enter a transaction number	X	
a. If the transaction number is invalid, check for re-entry error dialog.		X
b. If valid, check that *date*, *cashier ID*, and *Customer ID* are updated and the Receipt List Box is updated with the correct line items from the previous transaction.		X
c. Select a Line Item on the left list box; use the right arrow button to move a line item to Return Items list box on the right.		X

d. See that the *Subtotal*, *Tax*, and *Total* boxes are updated correctly.		X
e. Repeat steps c and d		X
f. Select an item to be returned, from the right hand list box, and remove it using the left arrow button.		X
g. Click the **Finish** button and verify that a dialog box showing the amount to be refunded appears.		X
2. Verify that the POST system returns the to the Cashier Login Screen	X	

Failure Details

Line 4a. of the Buy Items Test Plan failed due to lack of error checking with the database implementation. Phase 2 will concentrate on the addition of database error checking.

Line 7 of the Buy Items Test Plan failed due to lack of proper database accessors to obtain the necessary data. These will be added early in Phase 2. Their implementations will reside in the PostTerminal module and have similar functionality to those already present.

Lines 1a. – 1g. of the Return Items Test Plan failed because of lack of implementation time. As was mentioned previously, we encountered quite a few MFC issues that took most of our time. This has resulted in the pushing back of the majority of Return Items functionality to Phase 2.

IMPLEMENTATION DETAILS AND COMPLICATIONS – PHASE 2

MFC/GUI Related Issues

We did not have a lot of MFC or GUI related issues in the second phase. The main problem we did have, though was what edit boxes/buttons should receive focus in certain stages in the three views. We also had to make sure that the loss of focus on these items did not create Transactions, Cashiers, or Line Items inadvertently. This was causing a lot of memory leaks and crashes. Eventually, we added a lot of flow/error checking to determine what the Cashier could do at what points in the program. This took care of all of the errors and crashes.

Rational Rose Issues

We were able to use Rose to generate the outline of the functions needed to fully implement the Return Items Use Case. We modeled the functions in Rose and then forward-engineered them into our ReturnView module. This is about all we used Rose for in the second pass. All other modules already had outlines and function stubs from phase 1.

We were also able to reverse-engineer the final design to create a new overview model (which is shown after the phase 1 model). Unfortunately, the phase 2 model does not appear to be as complete (or as correct) as the phase 1 model. We are not quite sure what caused this.

OUTSTANDING REQUIREMENTS AND ISSUES FROM PHASE 2

The table below shows our requirements as they were originally conceived in the Analysis and Design from Phase 2.

Ref. #	Function	Category
R2.1	Handle credit card transactions, capturing credit card information from a card reader or by manual entry, and authorizing the transaction with the store's (external) credit authorization service via a modem connection.	Evident
R2.2	Handle check payments, capturing driver's license by manual entry, and authorizing payment with the store's (external) check authorization service via a modem connection.	Evident
R2.3	Handle store credit transactions, authorizing the credit through the store's (internal) store credit record database.	Evident
R2.4	Log credit payments to the Accounts Receivable system, since the credit authorization service owes the store the payment amount.	Hidden
R2.5	Handle and system administrator options.	Evident
R2.6	Manager responsible for startup of the POST system.	Evident
R2.7	Cashier has ability to void an item after entry.	Evident

The requirement that carried over from phase 1 was to complete the Return Items Use Case implementation. This has been completed for phase 2 with one exception. After the Cashier selects an item to void from the original receipt, the item appears in a separate list showing the items that will be returned and their quantities. At this point, there is no method for removing items from that second list in case of an entry error. The implementation of this functionality is not very difficult but, due to time constraints, we were not able to fully complete it. This will have to be left for phase 3.

For the phase 2 requirements shown in the above table, requirements 2.4, 2.5, and 2.6 were not implemented. This was again due to time constraints. Requirements 2.1, 2.2, and 2.3 were implemented at the database and GUI levels. They do not have any actual validation that takes place outside of the POST terminal. Again, this would come into phase 3.

The new requirement added at phase 2 (shown above as R2.7) was fully implemented for the Buy Items Use Case. As was mentioned above, it is not functional for the Return Items Use Case.

TEST PLANS FOR PHASE 2

Cashier Login Test Plan

Test Operation	Pass	Fail
1. Enter Cashier ID in *Cashier ID* edit box.	X	
g. Enter incorrect Password.	X	
h. Verify Error dialog, prompting for re-entry of password appears.	X	
i. Enter correct Password.	X	
j. Press **Sale.**	X	
k. Verify that system has logged in the user by bringing up the new transaction window.	X	
l. See that correct cashier ID and Cashier Name is in the *ID* and *Name* fields.	X	
2. Return to the Login screen by restarting the POST. Enter a Password, leaving *Cashier ID* field blank.	X	
f. Verify that an error appears prompting for re-entry of the Cashier ID.	X	
g. Enter a correct Password.	X	
h. Press the **Sale** button.	X	
i. Verify that system has logged in the user by bringing up the new transaction window.	X	
j. See that correct cashier ID and Cashier Name is in the *ID* and *Name* fields.	X	

Buy Items Test Plan

NOTE: Assume the Cashier is logged in and has pressed the **Sale** button.

Test Operation	Pass	Fail
1. Enter a Customer ID.	X	
2. See that the *Name* and *Address* fields are updated.	X	
3. Make sure the cursor appears in the *UPC* entry box.	X	
4. Enter a UPC.	X	
b. If a UPC is invalid, make sure a re-entry dialog appears.	X	
5. See that receipt information and subtotals are updated.	X	
b. If this is the first item, see that a date, time, and cashier are the first lines in the receipt, and verify accuracy of these fields.	X	
6. Enter two or more items, verifying that previously mentioned fields are updating correctly.	X	
7. Click **End Sale**, and verify that the receipt is updated with current customer's information. Make sure that *subtotal*, *tax*, and *total* are updated to correct amounts.		X
a. Repeat steps 1 through 7 using all payment types. Verify correct dialog boxes appear based on payment type chosen.	X	
8. See that the *Amount Tendered* Edit box gets focus.	X	
9. Enter an amount greater than or equal to the purchase total.	X	
10.Verify that the pop-up dialog shows the correct amount owed to customer.	X	

Return Items Test Plan

NOTE: Assume the Cashier is logged in and has pressed the **Return** button.

Test Operation	Pass	Fail
1. Enter a transaction number	X	
h. If the transaction number is invalid, check for re-entry error dialog.	X	
i. If valid, check that *date*, *cashier ID*, and *Customer ID* are updated and the Receipt List Box is updated with the correct line items from the previous transaction.	X	
j. Select a Line Item on the left list box; use the right arrow button to move a line	X	

item to Return Items list box on the right.		
k. Repeat step c.	X	
l. Select an item to be returned, from the right hand list box, and remove it using the left arrow button.		X
m. Click the **Finish** button and verify that a dialog box showing the amount to be refunded appears.	X	
2. Verify that the POST system returns the to the Cashier Login Screen	X	

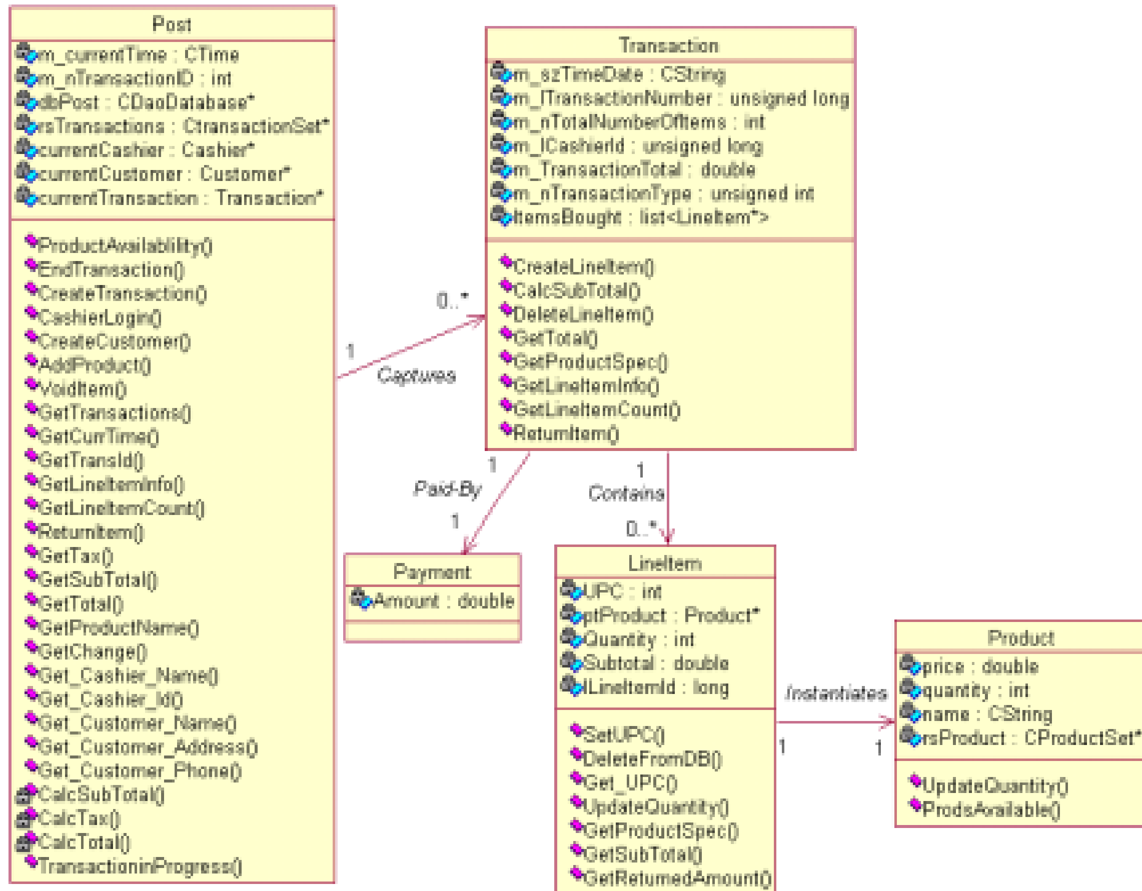
Failure Details

Line 7 of the Buy Items Test Plan failed due to lack of proper database accessors to obtain the necessary data. These will be added early in Phase 2. Their implementations will reside in the PostTerminal module and have similar functionality to those already present.

Line 1l of the Return Items Test Plan failed due to lack of implementation time. This will be left for Phase 3.

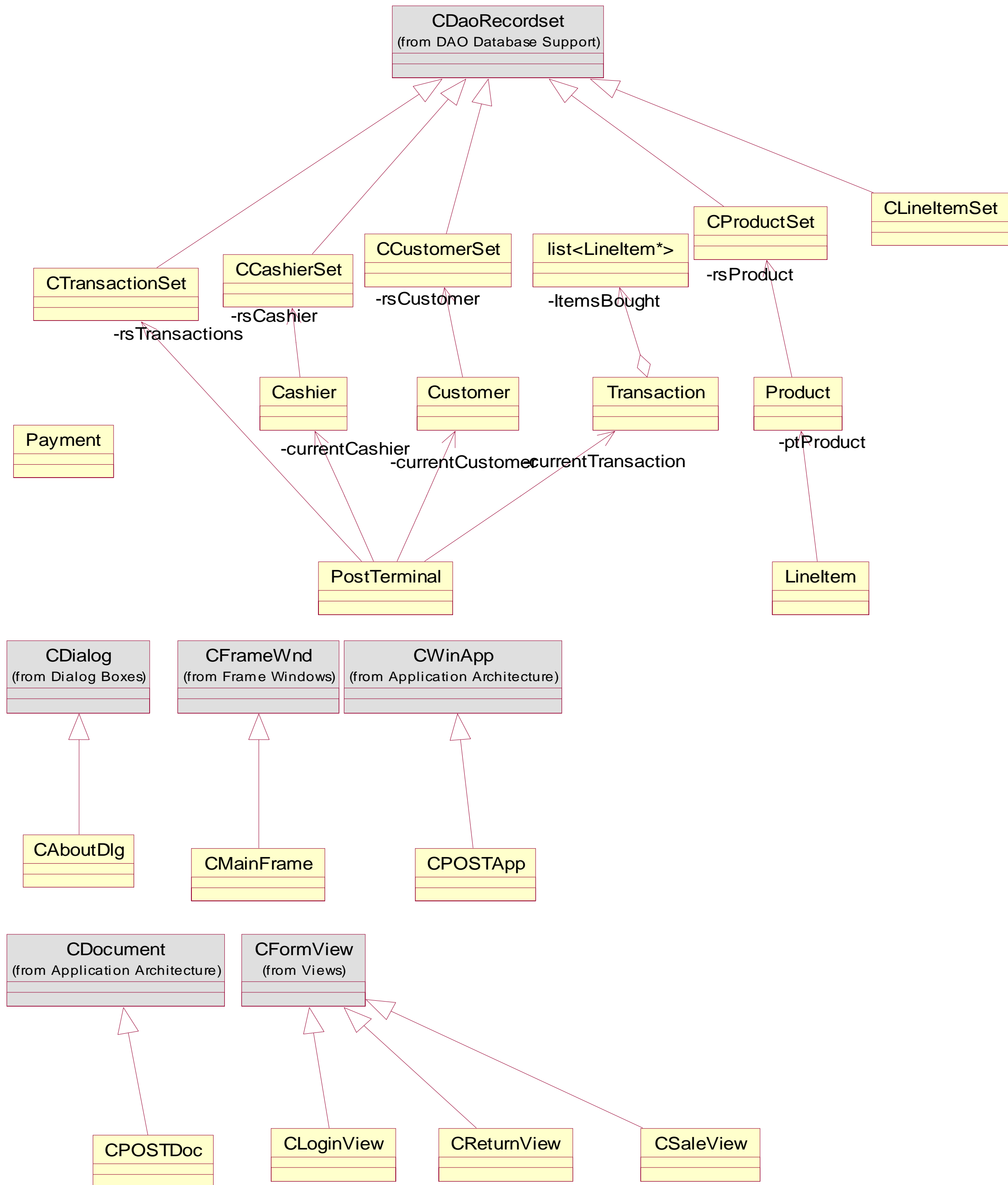
UPDATED DESIGN DIAGRAMS

Updated Class Diagram



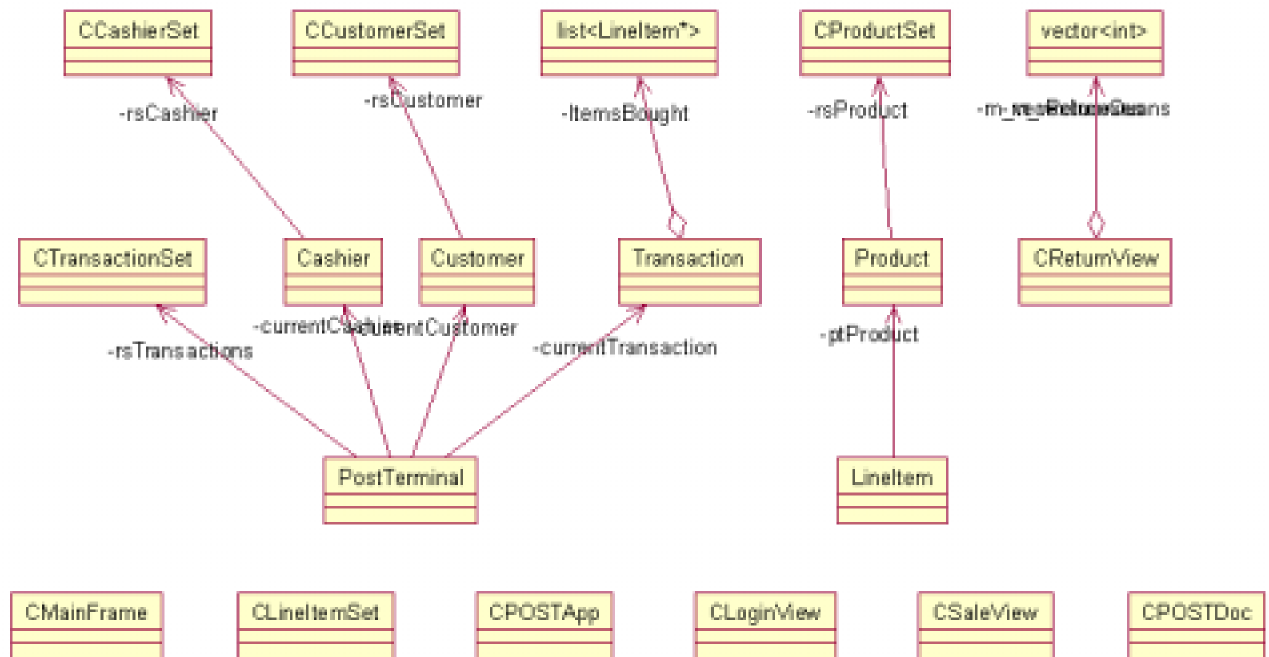
Rational Rose Reverse-Engineered Model Overview – Phase 1

POST Model Update Overview
This diagram was automatically created by Rational Rose 98i Model Update Tool.
Wednesday, October 20, 1999 8:11:00 AM



Rational Rose Reverse-Engineered Model Overview – Phase 2

POST Model Update Overview
This diagram was automatically created by Rational Rose 98i Model Update Tool.
Friday, November 12, 1999 4:16:15 PM



THE GUI FOR THE POST SYSTEM

Cashier Login Form

Cashier Login - POST

File

POST Cashier Login

Cashier ID:

Password:

Sale Return Logout

Sale Form (Used for Buy Items Use Case Implementation)

Return Form (Used for Refund Items Use Case Implementation)

Return Items - POST

File

Sale Information

Transaction ID: 134

Enter Trans ID

Date: Fri, Nov 12, 1999

Cashier ID: 2

Customer ID: 3

Receipt

Original

Point of Sale Terminal
Cashier ID: 2
Trans. ID: 2
Date: Fri, Nov 12, 1999

Qty.	Item	Price
1	Football	20.
0	Football	20.
1	Football	20.
1	Football	20.

0

>>

<<

Items Returned

1	Football	20.

Return

MODULE DESCRIPTIONS

Module Name: CashierSet
Module Description: This module contains a record set connection to the Cashier table in the POST97 database. All database functions are acquired from the CDaoRecordSet.

Module Name: Cashier
Module Description: This module represents a Cashier actor. This class contains a CashierSet object, an ID, a first and last name, and a password. All of these can be returned from a class object.

Module Name: CustomerSet
Module Description: This module contains a record set connection to the Customer table in the POST97 database. All database functions are acquired from the CDaoRecordSet.

Module Name: Customer
Module Description: This module represents a Customer actor. The class contains a CustomerSet object, the customer's ID, their address, phone, first name, and last name. All of these can be returned from a class object.

Module Name: LineItem
Module Description: This module contains information about one line on the receipt. A line item holds information about one product, the quantity, and the price. The class holds a UPC, a quantity, a subtotal, and a pointer to an instance of Product. This class can also get a subtotal for the current line item.

Module Name: Payment
Module Description: This module contains information about the Customer's payment. This contains a type and the amount.

Module Name: ProductSet
Module Description: This module contains a record set connection to the Product table in the POST97 database. All database functions are acquired from the CDaoRecordSet.

Module Name: Product
Module Description: This module represents a Product. The class contains a ProductSet object, a price, quantity, and name. All of these can be returned from a class object.

Module Name: TransactionSet
Module Description: This module contains a record set connection to the Transaction table in the POST97 database. All database functions are acquired from the CDaoRecordSet.

Module Name: Transaction
Module Description: This module represents a transaction. This can be either a sale or return. It contains a time for the transaction, a transaction number, the number of items in the transaction, the Cashier's ID, the amount of the sale without the tax, the type of transaction, and a list of the items purchased represented by LineItem objects.

Module Name: PostTerminal
Module Description: This is the largest and most important module in the POST system. It represents a POS Terminal and all its "global" functionality. The class interacts mostly with the database through a series of accessor functions.

APPENDIX A – POST CODE LISTING

Cashier.h

```
// Cashier.h: interface for the Cashier class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CASHIER_H__A6D73D00_85A8_11D3_9D03_402F51C10000__INCLUDED_
)
#define AFX_CASHIER_H__A6D73D00_85A8_11D3_9D03_402F51C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CashierSet.h" //Cashier Recordset

class Cashier
{
public:

            //constructors
            Cashier(int CashierId,CDaoDatabase* dbPost);

            //destructor
            ~Cashier(void);

            //accessors
            //Might want to change to CStrings??

            CString Get_Name (void);
            CString Get_Password (void);
            int Get_Id(void);

      private:

            CCashierSet* rsCashier;
            int id;
            CString lname;
            CString fname;
            CString password;

};

#endif //
!defined(AFX_CASHIER_H__A6D73D00_85A8_11D3_9D03_402F51C10000__INCLUDED_
)
```

Cashier.cpp

```
// Cashier.cpp: implementation of the Cashier class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "post.h"
#include "Cashier.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif



Cashier::~Cashier(void)
{
	rsCashier->Close();
	delete rsCashier;
}


Cashier::Cashier(int CashierId,CDaoDatabase* dbPost)
{

	//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	//Change to do Cashier info I copied from product
	//class which is very similar
	//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

	//need database connection
	rsCashier = new CCashierSet(dbPost);

	char* strCashierId = new char[64];

	itoa(CashierId, strCashierId, 10);

	CString selection = "SELECT * FROM CashierTbl WHERE CashierId = 
";
	CString sCashierId = strCashierId;
	selection += sCashierId;

	rsCashier->Open(dbOpenDynaset, (LPCTSTR)selection, dbConsistent);

	password = rsCashier->m_Password;
	lname = rsCashier->m_LName;
	fname = rsCashier->m_FName;
	id = rsCashier->m_CashierId;

	delete strCashierId;
}
```

```
int Cashier::Get_Id (void)
{
	return id;
}



CString Cashier::Get_Password (void)
{
	return password;
}


CString Cashier::Get_Name (void)
{
	//Should contantinae fname and lname to give full name
	CString space = " ";

	return fname + space + lname;
}
```

CashierSet.h

```
#if
!defined(AFX_CASHIERSET_H__6DCAB68B_84FE_11D3_A209_0000C0433AF8__INCLUD
ED_)
#define
AFX_CASHIERSET_H__6DCAB68B_84FE_11D3_A209_0000C0433AF8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CashierSet.h : header file
//

//////////////////////////////////////////////////////////////////////
//////
// CCashierSet DAO recordset

class CCashierSet : public CDaoRecordset
{
public:
      CCashierSet(CDaoDatabase* pDatabase = NULL);
      DECLARE_DYNAMIC(CCashierSet)

// Field/Param Data
      //{{AFX_FIELD(CCashierSet, CDaoRecordset)
      long  m_CashierId;
      CString     m_FName;
      CString     m_LName;
      CString     m_Password;
      //}}AFX_FIELD

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CCashierSet)
      public:
      virtual CString GetDefaultDBName();       // Default database
name
      virtual CString GetDefaultSQL();          // Default SQL for
Recordset
      virtual void DoFieldExchange(CDaoFieldExchange* pFX);  // RFX
support
      //}}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_CASHIERSET_H__6DCAB68B_84FE_11D3_A209_0000C0433AF8__INCLUD
ED_)
```

CashierSet.cpp

```
// CashierSet.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "CashierSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////////////////////////////////////////////
//////
// CCashierSet

IMPLEMENT_DYNAMIC(CCashierSet, CDaoRecordset)

CCashierSet::CCashierSet(CDaoDatabase* pdb)
	: CDaoRecordset(pdb)
{
	//{{AFX_FIELD_INIT(CCashierSet)
	m_CashierId = 0;
	m_FName = _T("");
	m_LName = _T("");
	m_Password = _T("");
	m_nFields = 4;
	//}}AFX_FIELD_INIT
	m_nDefaultType = dbOpenDynaset;
}


CString CCashierSet::GetDefaultDBName()
{
	return _T("S:\\NPost\\Post97.mdb");
}

CString CCashierSet::GetDefaultSQL()
{
	return _T("[CashierTbl]");
}

void CCashierSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
	//{{AFX_FIELD_MAP(CCashierSet)
	pFX->SetFieldType(CDaoFieldExchange::outputColumn);
	DFX_Long(pFX, _T("[CashierId]"), m_CashierId);
	DFX_Text(pFX, _T("[FName]"), m_FName);
	DFX_Text(pFX, _T("[LName]"), m_LName);
	DFX_Text(pFX, _T("[Password]"), m_Password);
	//}}AFX_FIELD_MAP
}
```

```
///////////////////////////////////////////////////////////////////////
//////
// CCashierSet diagnostics

#ifdef _DEBUG
void CCashierSet::AssertValid() const
{
	CDaoRecordset::AssertValid();
}

void CCashierSet::Dump(CDumpContext& dc) const
{
	CDaoRecordset::Dump(dc);
}
#endif //_DEBUG
```

Customer.h

```
// Customer.h: interface for the Customer class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CUSTOMER_H__A6D73D01_85A8_11D3_9D03_402F51C10000__INCLUDED
_)
#define AFX_CUSTOMER_H__A6D73D01_85A8_11D3_9D03_402F51C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "CustomerSet.h"      //Customer Recordset

class Customer
{
public:

            //constructors
            Customer(int CustomerId,CDaoDatabase* dbPost);

            //destructor
            ~Customer(void);

            //accessors
            //Might want to change to CStrings??
            CString Get_Name (void);
            CString Get_Address (void);
            CString Get_Phone (void);
            int Get_Id(void);

      private:

            CCustomerSet* rsCustomer;
            CString address;
            CString phone;
            CString lname;
            CString fname;
            int id;

};

#endif //
!defined(AFX_CUSTOMER_H__A6D73D01_85A8_11D3_9D03_402F51C10000__INCLUDED
_)
```

Customer.cpp

```
// Customer.cpp: implementation of the Customer class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "post.h"
#include "Customer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif


Customer::~Customer(void)
{
	rsCustomer->Close();
	delete rsCustomer;
}


Customer::Customer(int CustomerId,CDaoDatabase* dbPost)
{

	//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	//Change to do customer info I copied from product
	//class which is very similar
	//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

	//need database connection
	rsCustomer = new CCustomerSet(dbPost);

	char* strCustomerId = new char[64];

	itoa(CustomerId, strCustomerId, 10);

	CString selection = "SELECT * FROM CustomerTbl WHERE CustomerId =
";
	CString sCustomerId = strCustomerId;
	selection += sCustomerId;

	rsCustomer->Open(dbOpenDynaset, (LPCTSTR)selection,
dbConsistent);

	id = rsCustomer->m_CustomerId;
	fname = rsCustomer->m_FName;
	lname = rsCustomer->m_LName;
	address = rsCustomer->m_Address;
	phone =  rsCustomer->m_Phone;

	delete strCustomerId;
}
```

```
CString Customer::Get_Address (void)
{
	return address;
}



CString Customer::Get_Phone (void)
{
	return phone;
}


CString Customer::Get_Name (void)
{
	//Should contantinae fname and lname to give full name
	CString space = " ";
	return fname + space + lname;

}

int Customer::Get_Id(void)
{
	return id;
}
```

CustomerSet.h

```
#if
!defined(AFX_CUSTOMERSET_H__8E313934_8580_11D3_99D5_0000C0EB65E5__INCLU
DED_)
#define
AFX_CUSTOMERSET_H__8E313934_8580_11D3_99D5_0000C0EB65E5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CustomerSet.h : header file
//

///////////////////////////////////////////////////////////////////////
//////
// CCustomerSet DAO recordset

class CCustomerSet : public CDaoRecordset
{
public:
	CCustomerSet(CDaoDatabase* pDatabase = NULL);
	DECLARE_DYNAMIC(CCustomerSet)

// Field/Param Data
	//{{AFX_FIELD(CCustomerSet, CDaoRecordset)
	long	m_CustomerId;
	CString	m_FName;
	CString	m_LName;
	CString	m_Address;
	CString	m_Phone;
	//}}AFX_FIELD

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCustomerSet)
	public:
	virtual CString GetDefaultDBName();		// Default database
name
	virtual CString GetDefaultSQL();		// Default SQL for
Recordset
	virtual void DoFieldExchange(CDaoFieldExchange* pFX);  // RFX
support
	//}}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_CUSTOMERSET_H__8E313934_8580_11D3_99D5_0000C0EB65E5__INCLU
DED_)
```

CustomerSet.cpp

```
// CustomerSet.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "CustomerSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
//////
// CCustomerSet

IMPLEMENT_DYNAMIC(CCustomerSet, CDaoRecordset)

CCustomerSet::CCustomerSet(CDaoDatabase* pdb)
	: CDaoRecordset(pdb)
{
	//{{AFX_FIELD_INIT(CCustomerSet)
	m_CustomerId = 0;
	m_FName = _T("");
	m_LName = _T("");
	m_Address = _T("");
	m_Phone = _T("");
	m_nFields = 5;
	//}}AFX_FIELD_INIT
	m_nDefaultType = dbOpenDynaset;
}


CString CCustomerSet::GetDefaultDBName()
{
	return _T("H:\\ChampliR\\Software Design\\Post V.
2.1\\Post97.mdb");
}

CString CCustomerSet::GetDefaultSQL()
{
	return _T("[CustomerTbl]");
}

void CCustomerSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
	//{{AFX_FIELD_MAP(CCustomerSet)
	pFX->SetFieldType(CDaoFieldExchange::outputColumn);
	DFX_Long(pFX, _T("[CustomerId]"), m_CustomerId);
	DFX_Text(pFX, _T("[FName]"), m_FName);
	DFX_Text(pFX, _T("[LName]"), m_LName);
	DFX_Text(pFX, _T("[Address]"), m_Address);
	DFX_Text(pFX, _T("[Phone]"), m_Phone);
```

```
	//}}AFX_FIELD_MAP
}

////////////////////////////////////////////////////////////////////////
//////
// CCustomerSet diagnostics

#ifdef _DEBUG
void CCustomerSet::AssertValid() const
{
	CDaoRecordset::AssertValid();
}

void CCustomerSet::Dump(CDumpContext& dc) const
{
	CDaoRecordset::Dump(dc);
}
#endif //_DEBUG
```

LineItem.h

```
// LineItem.h: interface for the LineItem class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_LINEITEM_H__A6D73D02_85A8_11D3_9D03_402F51C10000__INCLUDED
_)
#define AFX_LINEITEM_H__A6D73D02_85A8_11D3_9D03_402F51C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Product.h"

class LineItem
{
public:
      void DeleteFromDB(CDaoDatabase* dbPost);
      int Get_UPC();
      void SetUPC(int nUPC);
      void SetQuantity(int nQuan);
      int GetQuantity();
      void UpdateQuantity(CDaoDatabase* dbPost,int prods_bought);

      //Takes quantity of products returned
      double GetReturnedAmount(int quantity_returned);

            //constructors
            LineItem();
            LineItem(int new_UPC, int Quant,int TransId,CDaoDatabase*
dbPost,int Type,int LineItemId);

            //destructor
            ~LineItem(void);

            CString GetProductSpec();

            double GetSubTotal(void);

      private:

            int UPC;                      //price of this product
            int Quantity;
            double Subtotal;
            long lLineItemId;
            Product* ptProduct;           //pointer to product


};

#endif //
!defined(AFX_LINEITEM_H__A6D73D02_85A8_11D3_9D03_402F51C10000__INCLUDED
_)
```

LineItem.cpp

```
// LineItem.cpp: implementation of the LineItem class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "post.h"
#include "LineItem.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

LineItem::LineItem()
{

}
//***************************************************
//
// LineItem - Constructor
//
// Returns - null
//
// Written by: Josh Fedke       10 Oct 99
//
//
//***************************************************
LineItem::LineItem(int new_UPC,int Quant,int TransId, CDaoDatabase*
dbPost,int Type,int LineItemId)
{
	//Set the properties of the line item
	UPC = new_UPC;
	Quantity = Quant;
	lLineItemId = LineItemId;

	//Create the new product object
	ptProduct = new Product(UPC,dbPost,Quantity,Type);

	//Calculate subtotal by multiplying the number of products
	//bought by the products price
	Subtotal = Quantity * ptProduct->Get_Price();


	if(Type == sale)
	{
		//Recrodset pointer
		CLineItemSet* rsLineItems;

		//Create the new recordset object
		rsLineItems = new CLineItemSet(dbPost);

		char* strUPC = new char[64];
		char* strQuant = new char[64];
```

```
		char* strTransId = new char[64];

		itoa(new_UPC, strUPC, 10);
		itoa(Quant,strQuant,10);
		ltoa(TransId,strTransId,10);


		CString selection = "INSERT INTO 
LineItemTbl(UPC,Quantity,TransactionId) VALUES (";
		CString sUPC = strUPC;
		CString sQuant = strQuant;
		CString sTransId = strTransId;
		CString sComma = ", ";
		CString sParens = ")";

		selection += sUPC;
		selection += sComma;
		selection += sQuant;
		selection += sComma;
		selection += sTransId;
		selection += sParens;

		//Execute the SQL statement with the database connection to 
insert a new
		//record into the line item table for this line item
		dbPost->Execute((LPCTSTR)selection);

		//Open a recordset to get the lineitems Id back
		rsLineItems->Open(dbOpenDynaset, "SELECT * FROM 
LineItemTbl", dbConsistent);

		//Go to the newest line item just added
		rsLineItems->MoveLast();

		lLineItemId = rsLineItems->m_LineItemId;

		//Close the recordset
		rsLineItems->Close();

		//Clean up
		delete rsLineItems;
		delete strQuant;
		delete strTransId;
		delete strUPC;
	}

}

LineItem::~LineItem()
{
	delete ptProduct;
}


double LineItem::GetSubTotal(void)
{
	return Subtotal;
```

```
}

CString LineItem::GetProductSpec()
{
	CString ProductSpecification;
	CString Price;
	CString Comma = ",";
	CString Quant;

	char* strPrice = new char[64];
	char* strQuant = new char[64];

	itoa(Quantity,strQuant,10);

	gcvt(ptProduct->Get_Price(),5,strPrice);
	Price = strPrice;
	Quant = strQuant;

	//Form a string that contains the quantity bought, the name of
the
	//product and the price of the product all separated by commas.
	ProductSpecification = strQuant + Comma + ptProduct->Get_Name() +
Comma + Price;

	delete strPrice;
	delete strQuant;
	return ProductSpecification;
	//Convert the price to a string
}

void LineItem::SetQuantity(int nQuan)
{
	Quantity = nQuan;

	//Update the subtotal since quantity changed
	//Calculate subtotal by multiplying the number of products
	//bought by the products price
	Subtotal = Quantity * ptProduct->Get_Price();
}

void LineItem::SetUPC(int nUPC)
{
	UPC = nUPC;
}

int LineItem::Get_UPC()
{
	return UPC;
}

void LineItem::DeleteFromDB(CDaoDatabase* dbPost)
{
	CString sSQL;
	CString strLineItemId;

	sSQL = "DELETE * FROM LineItemTbl WHERE LineItemId = ";
```

```
	char* sLineItemId = new char[64];

	ltoa(lLineItemId, sLineItemId, 10);

	strLineItemId = sLineItemId;

	sSQL += strLineItemId;

	//Execute the above sql statement to delete this line item from
the
	//database.
	dbPost->Execute(sSQL);

	delete sLineItemId;

}

int LineItem::GetQuantity()
{
	return Quantity;
}

void LineItem::UpdateQuantity(CDaoDatabase* dbPost,int prods_bought)
{
	CString sSQL1,sSQL2, sSQL;

	CString strLineItemId;
	CString strQuantity;

	//Update the quantity of the line item object in the database
	sSQL1="UPDATE LineItemTbl SET Quantity = ";

	sSQL2=" WHERE LineItemId = ";


	char* sLineItemId = new char[64];
	char* sQuantity = new char[64];

	ltoa(lLineItemId, sLineItemId, 10);
	itoa(Quantity, sQuantity,10);

	strLineItemId = sLineItemId;
	strQuantity = sQuantity;

	sSQL = sSQL1 + strQuantity + sSQL2 + strLineItemId;

	//Execute the above sql statement to delete this line item from
the
	//database.
	dbPost->Execute(sSQL);

	//Update the number of products available
	ptProduct->UpdateQuantity(prods_bought,dbPost);

}

double LineItem::GetReturnedAmount(int quantity_returned)
```

```
{
	double amount;

	amount = quantity_returned * ptProduct->Get_Price();

	return amount;
}
```

LineItemSet.h

```
#if
!defined(AFX_LINEITEMSET_H__62748A66_8665_11D3_9D03_A04851C10000__INCLU
DED_)
#define
AFX_LINEITEMSET_H__62748A66_8665_11D3_9D03_A04851C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LineItemSet.h : header file
//

////////////////////////////////////////////////////////////////////////
//////
// CLineItemSet DAO recordset

class CLineItemSet : public CDaoRecordset
{
public:
	CLineItemSet(CDaoDatabase* pDatabase = NULL);
	DECLARE_DYNAMIC(CLineItemSet)

// Field/Param Data
	//{{AFX_FIELD(CLineItemSet, CDaoRecordset)
	long	m_LineItemId;
	long	m_UPC;
	long	m_Quantity;
	long	m_Discount;
	long	m_TransactionId;
	//}}AFX_FIELD

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CLineItemSet)
	public:
	virtual CString GetDefaultDBName();		// Default database
name
	virtual CString GetDefaultSQL();		// Default SQL for
Recordset
	virtual void DoFieldExchange(CDaoFieldExchange* pFX);  // RFX
support
	//}}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_LINEITEMSET_H__62748A66_8665_11D3_9D03_A04851C10000__INCLU
DED_)
```

LineItemSet.cpp

```
// LineItemSet.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "LineItemSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////////////////////////////////////////////
//////
// CLineItemSet

IMPLEMENT_DYNAMIC(CLineItemSet, CDaoRecordset)

CLineItemSet::CLineItemSet(CDaoDatabase* pdb)
	: CDaoRecordset(pdb)
{
	//{{AFX_FIELD_INIT(CLineItemSet)
	m_LineItemId = 0;
	m_UPC = 0;
	m_Quantity = 0;
	m_Discount = 0;
	m_TransactionId = 0;
	m_nFields = 5;
	//}}AFX_FIELD_INIT
	m_nDefaultType = dbOpenDynaset;
}


CString CLineItemSet::GetDefaultDBName()
{
	return _T("C:\\Program Files\\Microsoft Visual
Studio\\MyProjects\\Software Engineering\\POST\\Post97.mdb");
}

CString CLineItemSet::GetDefaultSQL()
{
	return _T("[LineItemTbl]");
}

void CLineItemSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
	//{{AFX_FIELD_MAP(CLineItemSet)
	pFX->SetFieldType(CDaoFieldExchange::outputColumn);
	DFX_Long(pFX, _T("[LineItemId]"), m_LineItemId);
	DFX_Long(pFX, _T("[UPC]"), m_UPC);
	DFX_Long(pFX, _T("[Quantity]"), m_Quantity);
	DFX_Long(pFX, _T("[Discount]"), m_Discount);
	DFX_Long(pFX, _T("[TransactionId]"), m_TransactionId);
```

```
	//}}AFX_FIELD_MAP
}

///////////////////////////////////////////////////////////////////////
//////
// CLineItemSet diagnostics

#ifdef _DEBUG
void CLineItemSet::AssertValid() const
{
	CDaoRecordset::AssertValid();
}

void CLineItemSet::Dump(CDumpContext& dc) const
{
	CDaoRecordset::Dump(dc);
}
#endif //_DEBUG
```

Product.h

```
// Product.h: interface for the Product class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_PRODUCT_H__A6D73D05_85A8_11D3_9D03_402F51C10000__INCLUDED_
)
#define AFX_PRODUCT_H__A6D73D05_85A8_11D3_9D03_402F51C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "ProductSet.h"
#include <string.h>

class Product
{
public:

            //constructors
            Product(int new_UPC,CDaoDatabase* dbPost,int Quantity,int
Type);

            //destructor
            ~Product(void);

            //accessors
            double Get_Price (void);
            int Get_Quantity (void);
            CString Get_Name (void);

            //Updates the number of products available in the database
            void UpdateQuantity(int Quantity, CDaoDatabase* dbPost);
            int ProdsAvailable(CDaoDatabase* dbPost,int UPC);

      private:

            CProductSet* rsProduct;       //record set of items
            double price;           //price of this product
            int quantity;           //quantity in stock
            CString name;                 //name of the product
            int UPC;

};

#endif //
!defined(AFX_PRODUCT_H__A6D73D05_85A8_11D3_9D03_402F51C10000__INCLUDED_
)
```

Product.cpp

```
// Product.cpp: implementation of the Product class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "post.h"
#include "Product.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif


Product::~Product(void)
{
	delete rsProduct;
}

//************************************************
//
// Product - Constructor
//
// Returns - null
//
// Written by: Ryan Champlin       10 Oct 99
//
//  Creates a new product object and initializes all
//    private members with the data that is stored in
//  the database about this product.
//************************************************
Product::Product(int new_UPC,CDaoDatabase* dbPost,int Quantity,int
Type)
{


	//need database connection
	rsProduct = new CProductSet(dbPost);

	char* strUPC = new char[64];

	itoa(new_UPC, strUPC, 10);

	CString selection = "SELECT * FROM ProductTbl WHERE UPC = ";
	CString sUPC = strUPC;
	selection += sUPC;

	//Open the recordset getting the getting information on the
product with
	//the passed in UPC
	rsProduct->Open(dbOpenDynaset, (LPCTSTR)selection, dbConsistent);

	if(Type == sale)
```

```
	{
		//Update the quantity of the product available by
subracting off the quantity
		//that was bought by the customer
		rsProduct->Edit();
		rsProduct->m_Available -= Quantity;
		rsProduct->Update();
	}


	//Store the information from the database private data members
	price = rsProduct->m_Price;
	quantity = rsProduct->m_Available;
	name = rsProduct->m_Name;
	UPC = rsProduct->m_UPC;

	//Close the recordset
	rsProduct->Close();

	delete strUPC;
}


//**************************************************
//
// Get_Price
//
// Returns - double - price of the product
//
// Written by: Josh Fedke     10 Oct 99
//
//**************************************************
double Product::Get_Price (void)
{
	return price;
}


//**************************************************
//
// Get_Quantity
//
// Returns - int - quantity of the product
//
// Written by: Josh Fedke      10 Oct 99
//
//**************************************************
int Product::Get_Quantity (void)
{
	return quantity;
}

//**************************************************
//
// Get_Name
//
// Returns - int - quantity of the product
```

```
//
// Written by: Josh Fedke       10 Oct 99
//
//**********************************************
CString Product::Get_Name (void)
{
      return name;
}

//Updates the number of products available in the database
void Product::UpdateQuantity(int Quantity, CDaoDatabase* dbPost)
{
      CString sSQL1,sSQL2, sSQL;

      CString strUPC;
      CString strQuantity;

      int products_available = ProdsAvailable(dbPost,UPC);

      Quantity += products_available;

      //Update the availability of the product object in the database
      sSQL1="UPDATE ProductTbl SET Available = ";

      sSQL2=" WHERE UPC = ";


      char* sUPC = new char[64];
      char* sQuantity = new char[64];

      itoa(UPC, sUPC, 10);
      itoa(Quantity, sQuantity,10);

      strUPC = sUPC;
      strQuantity = sQuantity;

      sSQL = sSQL1 + strQuantity + sSQL2 + strUPC;

      //Execute the above sql statement to delete this line item from
the
      //database.
      dbPost->Execute(sSQL);
}


int Product::ProdsAvailable(CDaoDatabase* dbPost,int UPC)
{
      //Recordset containing info on product of given UPC
      CProductSet* rsProduct = new CProductSet(dbPost);

      char* strTemp = new char[64];
      int   products_available=0;

      itoa(UPC, strTemp, 10);

      //SQL Statement to retrieve product information where the
      //UPC is the given UPC
```

```
	CString sSQL = "SELECT * FROM ProductTbl WHERE UPC = ";
	CString sTemp = strTemp;
	sSQL += sTemp;

	//Open the recordset
	rsProduct->Open(dbOpenDynaset, (LPCTSTR)sSQL, dbConsistent);

	//Get the number of products in stock
	products_available = rsProduct->m_Available;

	//CLose the recordset
	rsProduct->Close();

	//Clean up
	delete strTemp;
	delete rsProduct;

	return products_available;
}
```

ProductSet.h

```
#if
!defined(AFX_PRODUCTSET_H__8E313936_8580_11D3_99D5_0000C0EB65E5__INCLUD
ED_)
#define
AFX_PRODUCTSET_H__8E313936_8580_11D3_99D5_0000C0EB65E5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ProductSet.h : header file
//

///////////////////////////////////////////////////////////////////////
//////
// CProductSet DAO recordset

class CProductSet : public CDaoRecordset
{
public:
	CProductSet(CDaoDatabase* pDatabase = NULL);
	DECLARE_DYNAMIC(CProductSet)

// Field/Param Data
	//{{AFX_FIELD(CProductSet, CDaoRecordset)
	long	m_UPC;
	CString	m_Name;
	double	m_Price;
	long	m_Available;
	//}}AFX_FIELD

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CProductSet)
	public:
	virtual CString GetDefaultDBName();		// Default database
name
	virtual CString GetDefaultSQL();		// Default SQL for
Recordset
	virtual void DoFieldExchange(CDaoFieldExchange* pFX);  // RFX
support
	//}}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_PRODUCTSET_H__8E313936_8580_11D3_99D5_0000C0EB65E5__INCLUD
ED_)
```

ProductSet.cpp

```
// ProductSet.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "ProductSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
//////
// CProductSet

IMPLEMENT_DYNAMIC(CProductSet, CDaoRecordset)

CProductSet::CProductSet(CDaoDatabase* pdb)
	: CDaoRecordset(pdb)
{
	//{{AFX_FIELD_INIT(CProductSet)
	m_UPC = 0;
	m_Name = _T("");
	m_Price = 0.0;
	m_Available = 0;
	m_nFields = 4;
	//}}AFX_FIELD_INIT
	m_nDefaultType = dbOpenDynaset;
}


CString CProductSet::GetDefaultDBName()
{
	return _T("H:\\ChampliR\\Software Design\\Post V.
2.1\\Post97.mdb");
}

CString CProductSet::GetDefaultSQL()
{
	return _T("[ProductTbl]");
}

void CProductSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
	//{{AFX_FIELD_MAP(CProductSet)
	pFX->SetFieldType(CDaoFieldExchange::outputColumn);
	DFX_Long(pFX, _T("[UPC]"), m_UPC);
	DFX_Text(pFX, _T("[Name]"), m_Name);
	DFX_Double(pFX, _T("[Price]"), m_Price);
	DFX_Long(pFX, _T("[Available]"), m_Available);
	//}}AFX_FIELD_MAP
}
```

```
////////////////////////////////////////////////////////////////////////
//////
// CProductSet diagnostics

#ifdef _DEBUG
void CProductSet::AssertValid() const
{
	CDaoRecordset::AssertValid();
}

void CProductSet::Dump(CDumpContext& dc) const
{
	CDaoRecordset::Dump(dc);
}
#endif //_DEBUG
```

Transaction.h

```
// Transaction.h: interface for the Transaction class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_TRANSACTION_H__A6D73D03_85A8_11D3_9D03_402F51C10000__INCLU
DED_)
#define
AFX_TRANSACTION_H__A6D73D03_85A8_11D3_9D03_402F51C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <list>
#include "LineItem.h"
#include "TransactionSet.h"
#include "PostConstants.h"

using namespace std;

class Transaction
{
public:
	void CreateLineItem(int UPC,int Quantity, CDaoDatabase*
dbPost,int Type, int LineItemId);
	void CalcSubTotal();
	double GetTotal();
	CString GetProductSpec();
	void DeleteLineItem(int index,CDaoDatabase* dbPost);

	CString GetLineItemInfo(int index);
	int GetLineItemCount();
	double ReturnItem(int index,int quantity,CDaoDatabase* dbPost);

	Transaction();
	Transaction(int CashierId,int TransactionId, unsigned int Type);
	Transaction(int TransactionId,CDaoDatabase* dbpost);

	~Transaction();

private:
	CString m_szTimeData;			//Date transaction was
done
	unsigned long m_lTransactionNumber; //Transaction Id
	int m_nTotalNumberOfItems;		//Do we need this
	unsigned long m_lCashierId;
	double m_TransactionTotal;		//Total amount of sale
without tax
	unsigned int m_nTransactionType;	//Type of transaction
	list<LineItem*> ItemsBought;		//List of line items

};
```

```
#endif //
!defined(AFX_TRANSACTION_H__A6D73D03_85A8_11D3_9D03_402F51C10000__INCLU
DED_)
```

Transaction.cpp

```
// Transaction.cpp: implementation of the Transaction class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "post.h"
#include "Transaction.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif



Transaction::Transaction()
{
	//does nothing
}

Transaction::~Transaction()
{
	//Create iterators to access list
	list<LineItem*>::iterator access_iter, end_iter;

	access_iter = ItemsBought.begin();
	end_iter = ItemsBought.end();

	//Delete all the line item objects
	for(;access_iter!=end_iter; access_iter++)
	{
		delete (*access_iter);
	}

}

Transaction::Transaction(int CashierId,int TransactionId, unsigned int
Type)
{

	m_TransactionTotal = 0;				//Initial
balance is 0
	m_nTotalNumberOfItems = 0;				//Initial
item count is 0
	m_lTransactionNumber = TransactionId;		//Set the
transaction number
	m_nTransactionType = Type;				//Set the
transaction type
	m_lCashierId = CashierId;
	CTime tempTime = CTime::GetCurrentTime(); //Record the time and
date of the transaction

	//Output format is "Tuesday, March 19, 1999"
```

```
	m_szTimeData = tempTime.Format( "%A, %B %d, %Y" );

}

//Use this constructor when doing a return item.
Transaction::Transaction(int TransactionId,CDaoDatabase* dbpost)
{
	//Didn't check type we use this constructor for doing returns.

	CTransactionSet* rsTransactions = new CTransactionSet(dbpost);
	CLineItemSet* rsLineItems = new CLineItemSet(dbpost);

	char* strTemp = new char[64];

	itoa(TransactionId, strTemp, 10);

	CString sSQL = "SELECT * FROM TransactionTbl WHERE TransactionId
= ";
	CString sTemp = strTemp;
	sSQL += sTemp;

	//Open recordset to get the transaction information from the
database
	rsTransactions->Open(dbOpenDynaset, (LPCTSTR)sSQL, dbConsistent);

	m_TransactionTotal = 0;					//Initial
balance is 0
	m_nTotalNumberOfItems = 0;				//Initial
item count is 0
	m_lTransactionNumber = TransactionId;		//Set the
transaction number

	//Shouldn't do this should be passing in type but didn't do it
just to save
	//time.
	m_nTransactionType = refund;			//Set the
transaction type

	m_lCashierId = rsTransactions->m_CashierId;

	//Retrieve the date
	m_szTimeData = rsTransactions->m_Date;

	rsTransactions->Close();

	delete rsTransactions;

	////////////////////////////////////////////////////////////////
/////////////////
	//Dynamically create the list of line items from the database
	////////////////////////////////////////////////////////////////
/////////////////

	sSQL = "SELECT * FROM LineItemTbl WHERE TransactionId = ";
	sTemp = strTemp;
	sSQL += sTemp;
```

```
	//Open recordset to get the transaction information from the
database
	rsLineItems->Open(dbOpenDynaset, (LPCTSTR)sSQL, dbConsistent);

	int UPC;
	int Quantity;
	int LineItemId;

	//Get the information out of the database for each line item and
then
	//create a new line item object and add it to the list by calling
	//CreateLineItem
	while(!(rsLineItems->IsEOF()))
	{
		UPC = rsLineItems->m_UPC;
		Quantity = rsLineItems->m_Quantity;
		LineItemId = rsLineItems->m_LineItemId;
		CreateLineItem(UPC,Quantity,dbpost,refund,LineItemId);
		rsLineItems->MoveNext();
	}

	delete rsLineItems;
	delete strTemp;
}

void Transaction::CreateLineItem(int UPC,int Quantity,CDaoDatabase*
dbPost,int Type, int LineItemId)
{
	//Create a line item object
	LineItem* purchasedProduct = new
LineItem(UPC,Quantity,m_lTransactionNumber,dbPost,Type,LineItemId);

	//Add the line item pointer to the list of line items
	ItemsBought.push_back(purchasedProduct);

}

void Transaction::CalcSubTotal()
{
	//Erase the m_TransactionTotal
	m_TransactionTotal = 0;

	//Create iterators to access list
	list<LineItem*>::iterator access_iter, end_iter;

	access_iter = ItemsBought.begin();
	end_iter = ItemsBought.end();

	for(;access_iter!=end_iter; access_iter++)
	{
		m_TransactionTotal+= (*access_iter)->GetSubTotal();
	}

}

double Transaction::GetTotal()
{
```

```
	return m_TransactionTotal;
}



CString Transaction::GetProductSpec()
{
	CString ProdSpec;

	//Get the product specification
	ProdSpec = ((ItemsBought.back())->GetProductSpec());
	return ProdSpec;
}

void Transaction::DeleteLineItem(int index,CDaoDatabase* dbPost)
{
	//Subtract six because we have header information in the text box
	index -=6;

	//Keeps track of where you are in the list, acts as an index into
the list.
	int counter=0;

	//Create iterators to access list
	list<LineItem*>::iterator access_iter, end_iter;

	access_iter = ItemsBought.begin();
	end_iter = ItemsBought.end();

	//Goes through the list to find the line item the user
	//wishes to delete.
	while((counter != index)&&(access_iter!=end_iter))
	{
		access_iter++;
		counter++;
	}

	//See if we went all the way through the list.  If we did then
the index was wrong
	//because it was too big.
	if(access_iter!=end_iter)
	{
		//Delete its information from the database
		(*access_iter)->DeleteFromDB(dbPost);

		//Delete the line item object referenced in the list
		delete (*access_iter);

		//Get rid of the line item pointer in the list
		ItemsBought.erase(access_iter);
	}

	else
		AfxMessageBox("Index For Removing Line Item is Too
Large!!!(LineItem.cpp)");

}
```

```
//Returns the product specification for a line item given an index into
the list of
//line items.
CString Transaction::GetLineItemInfo(int index)
{
	CString product_spec;

	//Keeps track of where you are in the list, acts as an index into
the list.
	int counter=0;

	//Create iterators to access list
	list<LineItem*>::iterator access_iter, end_iter;

	access_iter = ItemsBought.begin();
	end_iter = ItemsBought.end();

	//Goes through the list to find the line item the user
	//wishes to delete.
	while((counter != index)&&(access_iter!=end_iter))
	{
		access_iter++;
		counter++;
	}

	//See if we went all the way through the list.  If we did then
the index was wrong
	//because it was too big.
	if(access_iter!=end_iter)
	{
		//Return the string containing the information from the
line item
		product_spec = (*access_iter)->GetProductSpec();
	}

	return product_spec;
}

int Transaction::GetLineItemCount()
{

	return (unsigned int)ItemsBought.size();

}

//Quantity is the quantity returned
double Transaction::ReturnItem(int index,int quantity,CDaoDatabase*
dbPost)
{
	double return_amount = 0;

	//Subtract six because we have header information in the text box
	index -=6;
```

```
	//Keeps track of where you are in the list, acts as an index into
the list.
	int counter=0;

	//Create iterators to access list
	list<LineItem*>::iterator access_iter, end_iter;

	access_iter = ItemsBought.begin();
	end_iter = ItemsBought.end();

	//Goes through the list to find the line item the user
	//wishes to delete.
	while((counter != index)&&(access_iter!=end_iter))
	{
		access_iter++;
		counter++;
	}

	//See if we went all the way through the list.  If we did then
the index was wrong
	//because it was too big.
	if(access_iter!=end_iter)
	{
		//Retrieve the quantity of products bought
		int prods_bought = (*access_iter)->GetQuantity();

		//subtract quantity from the number of the current products
purchased
		(*access_iter)->SetQuantity(prods_bought-quantity);

		return_amount = (*access_iter)-
>GetReturnedAmount(quantity);

		(*access_iter)->UpdateQuantity(dbPost,quantity);
	}
	return return_amount;
}
```

TransactionSet.h

```
#if
!defined(AFX_TRANSACTIONSET_H__8E313935_8580_11D3_99D5_0000C0EB65E5__IN
CLUDED_)
#define
AFX_TRANSACTIONSET_H__8E313935_8580_11D3_99D5_0000C0EB65E5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// TransactionSet.h : header file
//

///////////////////////////////////////////////////////////////////////
//////
// CTransactionSet DAO recordset

class CTransactionSet : public CDaoRecordset
{
public:
	CTransactionSet(CDaoDatabase* pDatabase = NULL);
	DECLARE_DYNAMIC(CTransactionSet)

// Field/Param Data
	//{{AFX_FIELD(CTransactionSet, CDaoRecordset)
	long	m_TransactionId;
	CString	m_Date;
	long	m_CustomerId;
	long	m_CashierId;
	long	m_paymentType;
	//}}AFX_FIELD

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTransactionSet)
	public:
	virtual CString GetDefaultDBName();		// Default database
name
	virtual CString GetDefaultSQL();		// Default SQL for
Recordset
	virtual void DoFieldExchange(CDaoFieldExchange* pFX);	// RFX
support
	//}}AFX_VIRTUAL

// Implementation
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_TRANSACTIONSET_H__8E313935_8580_11D3_99D5_0000C0EB65E5__IN
CLUDED_)
```

TransactionSet.cpp

```
// TransactionSet.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "TransactionSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////
//////
// CTransactionSet

IMPLEMENT_DYNAMIC(CTransactionSet, CDaoRecordset)

CTransactionSet::CTransactionSet(CDaoDatabase* pdb)
	: CDaoRecordset(pdb)
{
	//{{AFX_FIELD_INIT(CTransactionSet)
	m_TransactionId = 0;
	m_paymentType = 0;
	//m_Date = (DATE)0;
	m_Date = _T("");
	m_CustomerId = 0;
	m_CashierId = 0;
	m_nFields = 5;
	//}}AFX_FIELD_INIT
	m_nDefaultType = dbOpenDynaset;
}


CString CTransactionSet::GetDefaultDBName()
{
	return _T((LPCTSTR)sDBLocation);
}

CString CTransactionSet::GetDefaultSQL()
{
	return _T("[TransactionTbl]");
}

void CTransactionSet::DoFieldExchange(CDaoFieldExchange* pFX)
{
	//{{AFX_FIELD_MAP(CTransactionSet)
	pFX->SetFieldType(CDaoFieldExchange::outputColumn);
	DFX_Long(pFX, _T("[TransactionId]"), m_TransactionId);
	//DFX_DateTime(pFX, _T("[Date]"), m_Date);
	DFX_Text(pFX, _T("[Date]"), m_Date);
	DFX_Long(pFX, _T("[CustomerId]"), m_CustomerId);
	DFX_Long(pFX, _T("[CashierId]"), m_CashierId);
```

```
	DFX_Long(pFX, _T("[paymentType]"), m_paymentType);
	//}}AFX_FIELD_MAP
}

////////////////////////////////////////////////////////////////////////
//////
// CTransactionSet diagnostics

#ifdef _DEBUG
void CTransactionSet::AssertValid() const
{
	CDaoRecordset::AssertValid();
}

void CTransactionSet::Dump(CDumpContext& dc) const
{
	CDaoRecordset::Dump(dc);
}
#endif //_DEBUG
```

PostTerminal.h

```
// PostTerminal.h: interface for the PostTerminal class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_POSTTERMINAL_H__A6D73D04_85A8_11D3_9D03_402F51C10000__INCL
UDED_)
#define
AFX_POSTTERMINAL_H__A6D73D04_85A8_11D3_9D03_402F51C10000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "TransactionSet.h"
#include "Cashier.h"
#include "Customer.h"
#include "Transaction.h"
#include "ProductSet.h"

#include "LineItem.h"
#include "LineItemSet.h"
#include <list>
using namespace std;

class PostTerminal
{
public:
	bool TransactionInProgress();

	CString GetTransactions(int nTransID);
	CString GetCurrTime(void);

	int GetTransID(void);			//Get transactionId of
current tranaction
	int ProductAvailability(int UPC);	//Returns quantity of product
in stock

	void EndTransaction();			//Cleans up the objects
such as customer
	void EndTransaction(unsigned int credit);

	//Creates a new tranaction where Type specifies a sale or return.
	//TranactionId must be specified for a return of items
	void CreateTransaction(unsigned int Type,int TransactionId);

	bool CashierLogin(int CashierId,CString password);//Creates a
cashier object

	void CreateCustomer(int CustomerId);			//Create
a customer object

	CString AddProduct(int UPC, int nQuantity);		//Adds a
lineitem to the sale
```

```
	/////////////////////////////
	//Used for doing return items

	CString GetLineItemInfo(int index);				//String
with line item info
	int GetLineItemCount();
//Number of current line items

	//Return an item given index of selected item and amount to
return
	double ReturnItem(int index, int quantity);
	/////////////////////////////


	//list<LineItem*> GetByTransID(int nTransID);

	void VoidItem(int index);
//Remove an item from the sale

	//Retrieve values for certain text fields
	CString GetTax();
//Return amount of tax on sale
	CString GetSubTotal();
//Get subtotal of sale
	CString GetTotal();
//Get (subtotal+tax)
	CString GetProductName(int UPC);				//Prod
name given its UPC
	double GetChange(double AmountPaid);			//Return
customers change

	//Cashier information
	CString Get_Cashier_Name();
//Current Cashier Name
	int Get_Cashier_Id();
//Current Cashier Id

	//Customer information
	CString Get_Customer_Name();				//Current
Customer Name
	CString Get_Customer_Address();
//Current Customer Address
	CString Get_Customer_Phone();				//Current
Customer Phone #

	//Constructor and destructor
	PostTerminal();
	~PostTerminal();

private:
	CTime m_currentTime;					//Current
Time
	int m_nTransactionID;
	//Transaction Id of current tranaction
```

```
      CDaoDatabase* dbPost;                                  //Database
connection
      CTransactionSet* rsTransactions;                 //Recordset of
transactions

      //This will be pointer to current cashier using system
      Cashier* currentCashier;

      //Should also have a customer object
      Customer* currentCustomer;

      //Current Transaction Object
      Transaction* currentTransaction;

      //Calc Functions
      double CalcSubTotal();                                  //Calculate
the sub total
      double CalcTax();                                       //Calculate
the tax on the subtotal
      double CalcTotal();
      //Calculate total cost

};

#endif //
!defined(AFX_POSTTERMINAL_H__A6D73D04_85A8_11D3_9D03_402F51C10000__INCL
UDED_)
```

PostTerminal.cpp

```
// PostTerminal.cpp: implementation of the PostTerminal class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "post.h"
#include "PostTerminal.h"
#include "PostConstants.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Creates a new database connection and creates a
//    recordset of all tranactions ever done from the
//    database
//************************************************

PostTerminal::PostTerminal()
{

      dbPost = new CDaoDatabase;

      //Open the database connection to the database
      //sDBLocation which is a global string specifies which database
to open
      dbPost->Open((LPCSTR)sDBLocation,false,false,"");

      //Create a new recordset of the transactions
      rsTransactions = new CTransactionSet(dbPost);

      //Load the recordset of all transactions in the database
      rsTransactions->Open(dbOpenDynaset,"SELECT * FROM
TransactionTbl",dbConsistent);

      currentTransaction = NULL;
      currentCustomer = NULL;
}

//************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Closes the recordset of transactions and the
//    database connection
//************************************************
PostTerminal::~PostTerminal()
{
      //Close the recordset
      rsTransactions->Close();
```

```
	//Close the database connection
	dbPost->Close();

	delete rsTransactions;
	delete dbPost;
}


//**************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Creates a new database connection and creates a
//    recordset of all tranactions ever done from the
//    database
//**************************************************
void PostTerminal::CreateTransaction(unsigned int Type,int
TransactionId)
{
	//If we're doing a sale then
	if(Type == sale)
	{

	//Make a new entry into the transaction table in the DB
	rsTransactions->AddNew();

	//Set the customerid of the tranaction
	rsTransactions->m_CustomerId = currentCustomer->Get_Id();

	//Record the time and date of the transaction
	m_currentTime = CTime::GetCurrentTime();

	//Output format is "Tuesday, March 19, 1999"
	rsTransactions->m_Date = m_currentTime.Format( "%a, %b %d, %Y" );

	//Get the cashier id of the current cashier
	rsTransactions->m_CashierId = currentCashier->Get_Id();

	//Submit new record to the database
	rsTransactions->Update();

	//Position recordset pointer to the just added tranaction
	rsTransactions->MoveLast();

	//Since the recordset index should be pointing at the newly added
	//record we should be able to just retrieve the Transaction Id
from it.
	m_nTransactionID = rsTransactions->m_TransactionId;

	//Create a new transaction object
	currentTransaction = new Transaction(currentCashier->Get_Id(),
m_nTransactionID, Type);

	}

	else
		//Processing a refund
```

```
            //Need to pass it the transaction id
            currentTransaction = new Transaction(TransactionId,
dbPost);

}

//***************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Get rid of the tranaction, customer and cashier
//    objects since we start with new for each transaction
//***************************************************

void PostTerminal::EndTransaction()
{
      //Get rid of the transaction and customer object
      delete currentTransaction;
      delete currentCustomer;

      //Do this because we relogin after each sale?
      delete currentCashier;

      currentTransaction = NULL;
      currentCustomer = NULL;
      currentCashier = NULL;
}


void PostTerminal::EndTransaction(unsigned int credit)
{
      CString strCredit, sSQL, strTransId;

      //Get rid of the transaction and customer object
      delete currentTransaction;
      delete currentCustomer;

      //Do this because we relogin after each sale?
      delete currentCashier;

      currentTransaction = NULL;
      currentCustomer = NULL;
      currentCashier = NULL;

      char* sCredit = new char[64];
      char* sTransId = new char[64];

      itoa(m_nTransactionID,sTransId,10);
      itoa(credit, sCredit, 10);

      strCredit = sCredit;
      strTransId = sTransId;

      sSQL = "UPDATE TransactionTbl SET paymentType = " + strCredit + "
WHERE TransactionId = " + strTransId;
      dbPost->Execute(sSQL);

      delete sCredit;
```

```
	delete sTransId;
}




//**************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Verifies that the username and password associated
//    with it are valid and returns to true if so.
//**************************************************

bool PostTerminal::CashierLogin(int CashierId,CString password)
{
	CString pass_from_db;               //Password of user from the
database
	bool passed;                              //Specifies if password
was right

	CCashierSet* rsCashier;             //Recordset of cashiers
	rsCashier = new CCashierSet(dbPost);

	char* strId = new char[64];

	itoa(CashierId, strId, 10);

	//Select all cashier information from the Cashier Table where the
	//Cashier Id is the specified ID CashierId
	CString selection = "SELECT * FROM CashierTbl WHERE CashierId =
";
	CString sId = strId;
	selection += sId;


	rsCashier->Open(dbOpenDynaset, (LPCTSTR)selection, dbConsistent);

	//Get the password from the database
	pass_from_db = rsCashier->m_Password;

	//If the password the user entered is the same as the password
	//in the database
	if(pass_from_db == password)
	{
		passed = true;
		//Valid password so create cashier object
		currentCashier = new Cashier(CashierId,dbPost);
	}
	else
	{
		passed = false;
		currentCashier = NULL;
	}

	delete strId;
	delete rsCashier;
```

```
	return passed;
}

//*************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Create a new customer object to represent the
//    current customer for a given sale
//*************************************************
void PostTerminal::CreateCustomer(int CustomerId)
{

	if(currentCustomer != NULL)
	{
		delete currentCustomer;
	}

	//Make a create customer function
	currentCustomer = new Customer(CustomerId,dbPost);
}

//*************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  The user just entered a UPC and quantity so add
//  a new lineitem to the reciept.  This adds the
//  line item and returns a string containing the
//    UPC, product name and quantity bought
//*************************************************
CString PostTerminal::AddProduct(int UPC,int nQuantity)
{
	CString ProductSpec;

	//call add line item of the current transaction
	//we're doing a sale and line item id doesn't matter
	currentTransaction->CreateLineItem(UPC,nQuantity,dbPost,sale,0);

	//Get the string containing the information about
	//the product purchased
	ProductSpec = currentTransaction->GetProductSpec();

	return ProductSpec;
}



//Customer Information accessors


CString PostTerminal::Get_Customer_Name()
{
	return currentCustomer->Get_Name();
}


CString PostTerminal::Get_Customer_Address()
{
```

```
	return currentCustomer->Get_Address();
}

CString PostTerminal::Get_Customer_Phone()
{
	return currentCustomer->Get_Phone();
}


//Cashier information acccessors


CString PostTerminal::Get_Cashier_Name()
{
	return currentCashier->Get_Name();
}

int PostTerminal::Get_Cashier_Id()
{
	return currentCashier->Get_Id();
}



//Cost information accessors

//**************************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Returns the total sale value as a CString
//**************************************************
CString PostTerminal::GetTotal()
{
	CString Total;
	double total = CalcTotal();
	char* strTotal = new char[64];
	sprintf(strTotal, "%7.2f", total);

	Total = strTotal;

	delete strTotal;

	return Total;

}

//**************************************************
// Written by: Ryan Champlin       23 Oct 99
//
//    Returns the tax of the sale as a CString
//**************************************************
CString PostTerminal::GetTax()
{
	CString Tax;
	double tax = CalcTax();
	char* strTax = new char[64];
```

```
      sprintf(strTax, "%7.2f", tax);

      Tax = strTax;

      delete strTax;

      return Tax;

}


//***************************************************
// Written by: Ryan Champlin       21 Oct 99
//
//  Return the subtotal of the sale as a CString
//***************************************************

CString PostTerminal::GetSubTotal()
{
      CString SubTotal;

      double sub = CalcSubTotal();
      char* strSubTotal = new char[64];
      sprintf(strSubTotal, "%7.2f", sub);

      SubTotal = strSubTotal;

      delete strSubTotal;

      return SubTotal;
}


//***************************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Calculates the sub total
//***************************************************
double PostTerminal::CalcSubTotal()
{
      double SubTotal;

      //Calculate the subtotal for the current transaction
      currentTransaction->CalcSubTotal();

      //Get the sub total from the current transaction
      SubTotal = currentTransaction->GetTotal();

      return SubTotal;
}

//***************************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Calculate the tax of the sale
//***************************************************
double PostTerminal::CalcTax()
```

```
{
	double Tax;

	//Calculate tax using 5% times the sub total
	//Shouldn't be hardcoded
	Tax = 0.05 * CalcSubTotal();

	return Tax;
}

//*********************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Calculate the total sale
//*********************************************
double PostTerminal::CalcTotal()
{
	double Total;

	Total = CalcSubTotal() + CalcTax();

	return Total;
}


//*********************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Returns the change the customer should get given
//    the amount he paid
//*********************************************
double PostTerminal::GetChange(double AmountPaid)
{
	double TotalChange;
	double Temp;

	Temp = CalcTotal();

	//Calculate the change
	TotalChange = AmountPaid - Temp;

	return TotalChange;
}

//*********************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Returns a products name given a UPC
//*********************************************
CString PostTerminal::GetProductName(int UPC)
{
	CString ProductName;          //Recordset of Product in Database

	CProductSet* rsProduct = new CProductSet(dbPost);

	char* strUPC = new char[64];
```

```
	itoa(UPC, strUPC, 10);

	//SQL statement that selects all products from the Product Table
	//where the UPC is the given UPC
	CString selection = "SELECT * FROM ProductTbl WHERE UPC = ";
	CString sUPC = strUPC;
	selection += sUPC;

	//Open the recordset
	rsProduct->Open(dbOpenDynaset, (LPCTSTR)selection, dbConsistent);

	//Retrieve the product name
	ProductName = rsProduct->m_Name;

	//Close the recordset
	rsProduct->Close();

	//Clean up
	delete rsProduct;
	delete strUPC;
	return ProductName;

}

int PostTerminal::GetTransID()
{
	return m_nTransactionID;
}

CString PostTerminal::GetCurrTime()
{
	return m_currentTime.Format( "%a, %b %d, %Y" );
}

CString PostTerminal::GetTransactions(int nTransID)
{
	CString strDate;
	CString strTemp;
	int nCashierID;
	int nCustID;
	char buffer[20];
	int nTemp;

	rsTransactions->MoveFirst();

	while(!rsTransactions->IsEOF())
	{
		nTemp = rsTransactions->m_TransactionId;
		if(nTemp == nTransID)
		{
			nCashierID=rsTransactions->m_CashierId;
			nCustID=rsTransactions->m_CustomerId;
			strDate=rsTransactions->m_Date;
			rsTransactions->MoveLast();
		}
		rsTransactions->MoveNext();
	}
```

```
	strTemp = strDate + "@";
	itoa(nCashierID, buffer, 10);
	strTemp = strTemp + buffer + "@";
	itoa(nCustID, buffer, 10);
	strTemp = strTemp + buffer;

	return strTemp;
}

/*
list<LineItem*> PostTerminal::GetByTransID(int nTransID)
{
	list<LineItem*> lstLineItems;
	CLineItemSet* rsLineItem = new CLineItemSet(dbPost);
	char* strTemp = new char[64];
	LineItem* tmpLineItem = new LineItem();


	itoa(nTransID, strTemp, 10);

	CString selection = "SELECT * FROM LineItemTbl WHERE
TransactionId = ";
	CString sTemp = strTemp;
	selection += sTemp;

	//Open the recordset getting the getting information on the
product with
	//the passed in UPC
	rsLineItem->Open(dbOpenDynaset, (LPCTSTR)selection,
dbConsistent);

	rsLineItem->MoveFirst();
	while(!rsLineItem->IsEOF())
	{
		tmpLineItem->SetQuantity(rsLineItem->m_Quantity);
		tmpLineItem->SetUPC(rsLineItem->m_UPC);

		lstLineItems.push_back(tmpLineItem);
	}

	return lstLineItems;
}
*/


//**************************************************
// Written by: Ryan Champlin       23 Oct 99
//
//  Removes an item from the current sale
//**************************************************
void PostTerminal::VoidItem(int index)
{
	//Delete the current line item selected in the list box
	//Index is the index into the list box of the selected
	//line item
	currentTransaction->DeleteLineItem(index,dbPost);
```

```
	currentTransaction->CalcSubTotal();
}


//**********************************************
// Written by: Ryan Champlin       29 Oct 99
//
//  Returns the number of products that are in stock
//    given a product UPC
//**********************************************
int PostTerminal::ProductAvailability(int UPC)
{
	//Recordset containing info on product of given UPC
	CProductSet* rsProduct = new CProductSet(dbPost);

	char* strTemp = new char[64];
	int   products_available=0;

	itoa(UPC, strTemp, 10);

	//SQL Statement to retrieve product information where the
	//UPC is the given UPC
	CString sSQL = "SELECT * FROM ProductTbl WHERE UPC = ";
	CString sTemp = strTemp;
	sSQL += sTemp;

	//Open the recordset
	rsProduct->Open(dbOpenDynaset, (LPCTSTR)sSQL, dbConsistent);

	//Get the number of products in stock
	products_available = rsProduct->m_Available;

	//CLose the recordset
	rsProduct->Close();

	//Clean up
	delete strTemp;
	delete rsProduct;

	return products_available;
}

//**********************************************
// Written by: Ryan Champlin       2 Nov 99
//
//  Returns a string containing the UPC, product name
//    and amount purchase given an index of the product
//  selected in the list box
//**********************************************
CString PostTerminal::GetLineItemInfo(int index)
{
	return currentTransaction->GetLineItemInfo(index);
}


//**********************************************
// Written by: Ryan Champlin       2 Nov 99
```

```
//
//  Return the number of lineitems
//***********************************************
int PostTerminal::GetLineItemCount()
{
      return currentTransaction->GetLineItemCount();
}

//***********************************************
// Written by: Ryan Champlin       2 Nov 99
//
//  Customer returns a given product given an index as to which
//    line item was selected in the list box and how
//  many are being returned.
//***********************************************
double PostTerminal::ReturnItem(int index, int quantity)
{
      double return_amount;

      return_amount = currentTransaction-
>ReturnItem(index,quantity,dbPost);

      return return_amount;
}

bool PostTerminal::TransactionInProgress()
{
      if(currentTransaction == NULL)
            return false;
      return true;
}
```

LoginView.h

```
#if
!defined(AFX_LOGINVIEW_H__6DCAB686_84FE_11D3_A209_0000C0433AF8__INCLUDE
D_)
#define
AFX_LOGINVIEW_H__6DCAB686_84FE_11D3_A209_0000C0433AF8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LoginView.h : header file
//

///////////////////////////////////////////////////////////////////////
//////
// CLoginView form view

#ifndef __AFXEXT_H__
#include <afxext.h>

#endif

#include "stdafx.h"

class CLoginView : public CFormView
{
// Moved to public so MainFrm and other Views can access it
public:
	CLoginView();           // protected constructor used by dynamic
creation
	DECLARE_DYNCREATE(CLoginView)


// Form Data
public:
	//{{AFX_DATA(CLoginView)
	enum { IDD = IDD_LOGIN_FORM };
	CEdit m_ctrlCashierID;
	CString     m_strCashierID;
	CString     m_strPassword;
	//}}AFX_DATA

// Attributes
public:




// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CLoginView)
	public:
```

```
	virtual void OnInitialUpdate();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	virtual void OnActivateView(BOOL bActivate, CView* pActivateView,
CView* pDeactiveView);
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CLoginView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
	//{{AFX_MSG(CLoginView)
	afx_msg void OnSale();
	afx_msg void OnReturn();
	afx_msg void OnLogout();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_LOGINVIEW_H__6DCAB686_84FE_11D3_A209_0000C0433AF8__INCLUDE
D_)
```

LoginView.cpp

```
// LoginView.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "LoginView.h"
#include "POSTDoc.h"
// View Includes
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// Gloabl Post Terminal
extern PostTerminal* g_pPostTerm;

///////////////////////////////////////////////////////////////////////
//////
// CLoginView

IMPLEMENT_DYNCREATE(CLoginView, CFormView)

CLoginView::CLoginView()
	: CFormView(CLoginView::IDD)
{
	//{{AFX_DATA_INIT(CLoginView)
	m_strCashierID = _T("");
	m_strPassword = _T("");
	//}}AFX_DATA_INIT

	g_pPostTerm = new PostTerminal();
}

CLoginView::~CLoginView()
{
	delete g_pPostTerm;
}

void CLoginView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CLoginView)
	DDX_Control(pDX, IDC_LOGIN_BOX, m_ctrlCashierID);
	DDX_Text(pDX, IDC_LOGIN_BOX, m_strCashierID);
	DDX_Text(pDX, IDC_PASSWORD_BOX, m_strPassword);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CLoginView, CFormView)
	//{{AFX_MSG_MAP(CLoginView)
```

```
	ON_BN_CLICKED(IDC_SALE, OnSale)
	ON_BN_CLICKED(IDC_RETURN, OnReturn)
	ON_BN_CLICKED(WM_DESTROY, OnDestroy)
	ON_BN_CLICKED(IDC_LOGOUT, OnLogout)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////////
//////
// CLoginView diagnostics

#ifdef _DEBUG
void CLoginView::AssertValid() const
{
	CFormView::AssertValid();
}

void CLoginView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}
#endif //_DEBUG

///////////////////////////////////////////////////////////////////////
//////
// CLoginView message handlers

void CLoginView::OnActivateView(BOOL bActivate, CView* pActivateView,
CView* pDeactiveView)
{
	//Get a pointer to the document
	CPOSTDoc* pDoc = (CPOSTDoc*)GetDocument();

	CFormView::OnActivateView(bActivate, pActivateView,
pDeactiveView);
	m_ctrlCashierID.SetFocus();
	m_strCashierID = "";
	m_strPassword = "";
	UpdateData(false);
	pDoc->SetTitle("Cashier Login");
}

void CLoginView::OnInitialUpdate()
{
	CFormView::OnInitialUpdate();

	// TODO: Add your specialized code here and/or call the base
class

}

void CLoginView::OnSale()
{
	UpdateData(true);
	if (m_strCashierID != "" || m_strPassword != "")
	{
```

```
		if(g_pPostTerm->CashierLogin(atoi(m_strCashierID),
m_strPassword))
		{
			((CMainFrame*)GetParentFrame())-
>SelectView(SALE_VIEW);
		}
		else
		{
			AfxMessageBox("Please enter valid Cashier ID &&
Password");
		}
	}
	else
	{
		AfxMessageBox("Please enter a Cashier ID && Password");
	}

}

void CLoginView::OnReturn()
{
	UpdateData(true);
	if (m_strCashierID != "" || m_strPassword != "")
	{
		if(g_pPostTerm->CashierLogin(atoi(m_strCashierID),
m_strPassword))
		{
			((CMainFrame*)GetParentFrame())-
>SelectView(RETURN_VIEW);
		}
		else
		{
			AfxMessageBox("Please enter valid Cashier ID &&
Password");
		}
	}
	else
	{
		AfxMessageBox("Please enter a Cashier ID && Password");
	}

}

void CLoginView::OnLogout()
{
	PostQuitMessage(0);
}
```

SaleView.h

```
#if
!defined(AFX_SALEVIEW_H__6DCAB688_84FE_11D3_A209_0000C0433AF8__INCLUDED
_)
#define AFX_SALEVIEW_H__6DCAB688_84FE_11D3_A209_0000C0433AF8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SaleView.h : header file
//

//////////////////////////////////////////////////////////////////////
//////
// CSaleView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CSaleView : public CFormView
{
public:
	CSaleView();           // protected constructor used by dynamic
creation
	DECLARE_DYNCREATE(CSaleView)

// Form Data
public:
	//{{AFX_DATA(CSaleView)
	enum { IDD = IDD_SALE_FORM };
	CButton	m_ctrlEnterCustId;
	CComboBox	m_ctrlPaymentType;
	CButton	m_ctrlVoidItem;
	CEdit	m_ctrlUPC;
	CListBox	m_ctrlReceiptList;
	CEdit	m_ctrlAmtTendered;
	CButton	m_ctrlEndSale;
	CButton	m_ctrlEnterItem;
	CEdit	m_ctrlCustID;
	CString	m_strCustName;
	CString	m_strCustAddress;
	CString	m_strCustCity;
	CString	m_strCustState;
	CString	m_strCustZIP;
	long	m_lCashierID;
	CString	m_strCashierName;
	CString	m_strQuantity;
	CString	m_strUPC;
	CString	m_SubTotal;
	CString	m_Tax;
	CString	m_Total;
	CString	m_strCustID;
	CString	m_strItemName;
	CString	m_strAmountTendered;
```

```
	CString	m_strPaymentType;
	//}}AFX_DATA

// Attributes
public:


// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSaleView)
	public:
	virtual void OnInitialUpdate();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	virtual void OnActivateView(BOOL bActivate, CView* pActivateView,
CView* pDeactiveView);
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CSaleView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
	//{{AFX_MSG(CSaleView)
	afx_msg void OnEndSale();
	afx_msg void OnEnterItem();
	afx_msg void OnKillfocusItemUPC();
	afx_msg void OnSetfocusItemUPC();
	afx_msg void OnVoidItem();
	afx_msg void OnCustId();
	afx_msg void OnSelchangePaymentCombo();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()

private:
	bool m_bClearFlag;
	bool m_bHeader;

};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
```

```
#endif //
!defined(AFX_SALEVIEW_H__6DCAB688_84FE_11D3_A209_0000C0433AF8__INCLUDED
_)
```

SaleView.cpp

```
// SaleView.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "SaleView.h"
#include "POSTDoc.h"

#include "MainFrm.h"
//#include "Transaction.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// Gloabl Post Terminal
extern PostTerminal* g_pPostTerm;

///////////////////////////////////////////////////////////////////////
//////
// CSaleView

IMPLEMENT_DYNCREATE(CSaleView, CFormView)

CSaleView::CSaleView()
	: CFormView(CSaleView::IDD)
{
	//{{AFX_DATA_INIT(CSaleView)
	m_strCustName = _T("");
	m_strCustAddress = _T("");
	m_strCustCity = _T("");
	m_strCustState = _T("");
	m_strCustZIP = _T("");
	m_strQuantity = _T("");
	m_strUPC = _T("");
	m_SubTotal = _T("");
	m_Tax = _T("");
	m_Total = _T("");
	m_strCustID = _T("");
	m_strItemName = _T("");
	m_strAmountTendered = _T("");
	m_strPaymentType = _T("Cash");
	//}}AFX_DATA_INIT

	m_bClearFlag = true;
}

CSaleView::~CSaleView()
{

}
```

```
void CSaleView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSaleView)
	DDX_Control(pDX, IDC_CUST_ID, m_ctrlEnterCustId);
	DDX_Control(pDX, IDC_PAYMENT_COMBO, m_ctrlPaymentType);
	DDX_Control(pDX, IDC_BUTTON1, m_ctrlVoidItem);
	DDX_Control(pDX, IDC_ItemUPC, m_ctrlUPC);
	DDX_Control(pDX, IDC_RECEIPTLIST, m_ctrlReceiptList);
	DDX_Control(pDX, IDC_AmountTendered, m_ctrlAmtTendered);
	DDX_Control(pDX, IDC_END_SALE, m_ctrlEndSale);
	DDX_Control(pDX, IDC_ENTER_ITEM, m_ctrlEnterItem);
	DDX_Control(pDX, IDC_CustID, m_ctrlCustID);
	DDX_Text(pDX, IDC_CustName, m_strCustName);
	DDX_Text(pDX, IDC_CustAddress, m_strCustAddress);
	DDX_Text(pDX, IDC_CustCity, m_strCustCity);
	DDX_Text(pDX, IDC_CustState, m_strCustState);
	DDX_Text(pDX, IDC_CustZip, m_strCustZIP);
	DDX_Text(pDX, IDC_CashierID, m_lCashierID);
	DDX_Text(pDX, IDC_CashierName, m_strCashierName);
	DDX_Text(pDX, IDC_ItemQuantity, m_strQuantity);
	DDX_Text(pDX, IDC_ItemUPC, m_strUPC);
	DDX_Text(pDX, IDC_SubTotal, m_SubTotal);
	DDX_Text(pDX, IDC_Tax, m_Tax);
	DDX_Text(pDX, IDC_Total, m_Total);
	DDX_Text(pDX, IDC_CustID, m_strCustID);
	DDX_Text(pDX, IDC_ItemName, m_strItemName);
	DDX_Text(pDX, IDC_AmountTendered, m_strAmountTendered);
	DDX_CBString(pDX, IDC_PAYMENT_COMBO, m_strPaymentType);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSaleView, CFormView)
	//{{AFX_MSG_MAP(CSaleView)
	ON_BN_CLICKED(IDC_END_SALE, OnEndSale)
	ON_BN_CLICKED(IDC_ENTER_ITEM, OnEnterItem)
	ON_EN_KILLFOCUS(IDC_ItemUPC, OnKillfocusItemUPC)
	ON_EN_SETFOCUS(IDC_ItemUPC, OnSetfocusItemUPC)
	ON_BN_CLICKED(IDC_BUTTON1, OnVoidItem)
	ON_BN_CLICKED(IDC_CUST_ID, OnCustId)
	ON_CBN_SELCHANGE(IDC_PAYMENT_COMBO, OnSelchangePaymentCombo)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////
//////
// CSaleView diagnostics

#ifdef _DEBUG
void CSaleView::AssertValid() const
{
	CFormView::AssertValid();
}

void CSaleView::Dump(CDumpContext& dc) const
{
```

```
	CFormView::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////
//////
// CSaleView message handlers

void CSaleView::OnActivateView(BOOL bActivate, CView* pActivateView,
CView* pDeactiveView)
{
	//Get a pointer to the document
	CPOSTDoc* pDoc = (CPOSTDoc*)GetDocument();

	if (m_bClearFlag == true)
	{
		CFormView::OnActivateView(bActivate, pActivateView,
pDeactiveView);
		CFormView::OnInitialUpdate();
		m_ctrlCustID.EnableWindow(true);
		m_ctrlEnterCustId.EnableWindow(true);
		m_ctrlReceiptList.ResetContent();
		m_strCustID = "";
		m_strCustName = "";
		m_strCustAddress = "";
		m_strCustCity = "";
		m_strCustState = "";
		m_strCustZIP = "";
		m_strCashierName = "";
		m_strQuantity = "";
		m_strUPC = "";
		m_SubTotal = "";
		m_Tax = "";
		m_Total = "";
		m_strItemName = "";
		m_strAmountTendered = "";
		m_strPaymentType = "Cash";

		//Only get the cashier id and name when activating this
view
		if(bActivate)
		{
			m_lCashierID = g_pPostTerm->Get_Cashier_Id();
			m_strCashierName = g_pPostTerm->Get_Cashier_Name();
		}

		UpdateData(false);
		m_ctrlCustID.SetFocus();
		m_ctrlEnterItem.EnableWindow(false);
		m_ctrlEndSale.EnableWindow(false);
		m_ctrlAmtTendered.EnableWindow(false);
		m_ctrlVoidItem.EnableWindow(false);
		m_ctrlPaymentType.EnableWindow(false);
		m_bHeader = false;
	}
	pDoc->SetTitle("Buy Items");
}
```

```
void CSaleView::OnInitialUpdate()
{

	// TODO: Add your specialized code here and/or call the base
class

}

void CSaleView::OnEndSale()
{
	double Change;
	char* temp;
	double tendered, owed;

	UpdateData(true);

	// Modified Code to use with Ryan's New Payment methods
	// Assuming we are not allowing checks for more then the amount
purchased and
	// that we are not doing verification on checks.
	if(m_strPaymentType == "Cash")
	{
		// Check to see if amount tendered is great enough
		owed = strtod(m_Total, &temp);
		tendered = strtod(m_strAmountTendered, &temp);

		if((owed <= tendered) || (owed <= 0))
		{

			Change = g_pPostTerm->GetChange(tendered);

			char* temp = new char[50];
			sprintf(temp, "%7.2f", Change);
			CString strTemp = temp;
			strTemp.TrimLeft();
			AfxMessageBox("The Customer's Change is: $" +
strTemp);
			delete temp;

			//Ends the transaction
			g_pPostTerm->EndTransaction(cash);
			m_strCustID = "";
			UpdateData(false);
			m_bClearFlag = true;
			((CMainFrame*)GetParentFrame())-
>SelectView(LOGIN_VIEW);
		}
		else
		{
			m_bClearFlag = false;
			AfxMessageBox("Not enough cash tendered!");
		}
	}
	else if(m_strPaymentType == "Check")
	{
		bool bAccepted = true;	// Need to implement
```

```
		if(!bAccepted)
		{
			m_bClearFlag = false;
			AfxMessageBox("The Check was not Approved!");
		}
		else
		{
			m_bClearFlag = false;
			AfxMessageBox("Check Accepted!");

			//Ends the transaction
			g_pPostTerm->EndTransaction(check);
//******************  Pass a variable with type of transaction
(Credit, Check, Cash)
			m_strCustID = "";
			UpdateData(false);
			m_bClearFlag = true;
			((CMainFrame*)GetParentFrame())-
>SelectView(LOGIN_VIEW);
		}
	}

	else
	{
		bool bAccepted = true;    // Need to implement
		if(!bAccepted)
		{
			m_bClearFlag = false;
			AfxMessageBox("The Credit Card was declined!");
		}
		else
		{
			m_bClearFlag = false;
			AfxMessageBox("Credit Card accepted, remember to
get\ncustomers signature.");

			//Ends the transaction
			g_pPostTerm->EndTransaction(credit);
//******************  Pass a variable with type of transaction
(Credit, Check, Cash)
			m_strCustID = "";
			UpdateData(false);
			m_bClearFlag = true;
			((CMainFrame*)GetParentFrame())-
>SelectView(LOGIN_VIEW);
		}
	}

	UpdateData(false);
}

void CSaleView::OnEnterItem()
{
	//Need to update to make sure the newly typed quantity is
available
	UpdateData(true);
```

```
	int	nQuan = g_pPostTerm->ProductAvailability(atoi(m_strUPC));

	if(atoi(m_strQuantity) > nQuan)
	{
		char* error = new char[50];
		sprintf(error, "There is only %d of that in stock, please
change quantity.", nQuan);
		AfxMessageBox(error);
	}
	else
	{
		char* left = new char[50];
		char* center = new char[50];
		char* right = new char[50];
		char* temp = new char[50];

		if (!m_bHeader)
		{
			sprintf(temp, "Point of Sale Terminal");
			m_ctrlReceiptList.InsertString(-1, temp);
			sprintf(temp, "Cashier ID: %d", m_lCashierID);
			m_ctrlReceiptList.InsertString(-1, temp);
			sprintf(temp, "Trans. ID: %d", g_pPostTerm-
>GetTransID());
			m_ctrlReceiptList.InsertString(-1, temp);
			sprintf(temp, "Date: %s", g_pPostTerm-
>GetCurrTime());
			m_ctrlReceiptList.InsertString(-1, temp);
			m_ctrlReceiptList.InsertString(-1, "");
			sprintf(temp, "%8s%16s%13s", "Qty.", "Item",
"Price");
			m_ctrlReceiptList.InsertString(-1, temp);

			m_bHeader = true;
		}

		UpdateData(true);
		CString spec = g_pPostTerm-
>AddProduct(atoi(m_strUPC),atoi(m_strQuantity));

		//Add spec to the list box
		//Add it to the end of the list box by sending it -1
		strcpy(left, spec.Left(spec.Find(",")));
		spec = spec.Right((spec.GetLength() - spec.Find(",") - 1));
		strcpy(center, spec.Left(spec.Find(",", 0)));
		spec = spec.Right((spec.GetLength() - spec.Find(",") - 1));
		strcpy(right, spec);

		sprintf(temp, "%8s%16s%10s", left, center, right);
		m_ctrlReceiptList.InsertString(-1,temp);
		//Set the Subtotal
		m_SubTotal = g_pPostTerm->GetSubTotal();
		m_SubTotal.TrimLeft();
		//Set the tax
		m_Tax = g_pPostTerm->GetTax();
		m_Tax.TrimLeft();
		//Set the total
```

```
            m_Total = g_pPostTerm->GetTotal();
            m_Total.TrimLeft();
            UpdateData(false);
            m_ctrlAmtTendered.EnableWindow(true);
            m_ctrlPaymentType.EnableWindow(true);
            delete left;
            delete center;
            delete right;
            delete temp;
      }
}

void CSaleView::OnKillfocusItemUPC()
{
      // TODO: Add your control notification handler code here
      UpdateData(true);
      m_strItemName = g_pPostTerm->GetProductName(atoi(m_strUPC));
      if(m_strItemName == "")
      {
            m_bClearFlag = false;
            AfxMessageBox("That is not a valid UPC.  Please re-
enter.");
            m_ctrlUPC.SetFocus();
      }
      else
      {
            m_strQuantity = "1";
            UpdateData(false);
            m_ctrlEnterItem.EnableWindow(true);
            m_ctrlVoidItem.EnableWindow(true);
      }
}

void CSaleView::OnSetfocusItemUPC()
{
      // TODO: Add your control notification handler code here
      m_strQuantity = "";
      m_strItemName = "";
      m_strUPC = "";
      UpdateData(false);
}


void CSaleView::OnVoidItem()
{
      int index;

      //Gets the index of the currently selected item
      index = m_ctrlReceiptList.GetCurSel();
      if (index <= 5)
      {
            m_bClearFlag = false;
            AfxMessageBox("Please select a valid item to void!");
      }
      else
      {
            //Deletes line item from Post System
```

```
		g_pPostTerm->VoidItem(index);

		//Delete the string from the list box
		m_ctrlReceiptList.DeleteString(index);
	}
	m_SubTotal = g_pPostTerm->GetSubTotal();
	m_SubTotal.TrimLeft();
	m_Tax = g_pPostTerm->GetTax();
	m_Tax.TrimLeft();
	m_Total = g_pPostTerm->GetTotal();
	m_Total.TrimLeft();
	UpdateData(false);
}

void CSaleView::OnCustId()
{
	UpdateData(true);
	if(m_strCustID != "")
	{
		CString tempAddr;
		m_bClearFlag = false;
		// Create customer
		g_pPostTerm->CreateCustomer(atoi(m_strCustID));

		m_strCustName = g_pPostTerm->Get_Customer_Name();
		tempAddr = g_pPostTerm->Get_Customer_Address();
		int nAddr = tempAddr.Find(",");
		int nCity = tempAddr.Find(",", nAddr + 1);
		int nState = tempAddr.Find(",", nCity + 1);
		m_strCustAddress = tempAddr.Mid(0, nAddr);
		m_strCustCity = tempAddr.Mid(nAddr + 1, (nCity - 1) -
nAddr);
		m_strCustState = tempAddr.Mid(nCity + 1, (nState - 1) -
nCity);
		m_strCustZIP = tempAddr.Mid(nState + 1, 5);
		UpdateData(false);

		//Create a new transaction
		//Creating a sale so just past zero for the trans id
		g_pPostTerm->CreateTransaction(sale,0);

		m_ctrlCustID.EnableWindow(false);
		m_ctrlEndSale.EnableWindow(true);
		m_ctrlEnterCustId.EnableWindow(false);

		m_ctrlUPC.SetFocus();
	}
}


void CSaleView::OnSelchangePaymentCombo()
{
	// TODO: Add your control notification handler code here
	UpdateData(true);

	if((m_strPaymentType == "Credit Card") || (m_strPaymentType ==
"Check"))
```

```
	{
		m_ctrlAmtTendered.EnableWindow(false);
	}
	else
	{
		m_ctrlAmtTendered.EnableWindow(true);
	}
}
```

ReturnView.h

```
#if
!defined(AFX_RETURNVIEW_H__6DCAB687_84FE_11D3_A209_0000C0433AF8__INCLUD
ED_)
#define
AFX_RETURNVIEW_H__6DCAB687_84FE_11D3_A209_0000C0433AF8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ReturnView.h : header file
//

///////////////////////////////////////////////////////////////////////
//////
// CReturnView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif
#include <vector>

using namespace std;

class CReturnView : public CFormView
{
public:
	CReturnView();           // protected constructor used by dynamic
creation
	DECLARE_DYNCREATE(CReturnView)

// Form Data
public:
	//{{AFX_DATA(CReturnView)
	enum { IDD = IDD_RETURN_FORM };
	CListBox	m_Receipt2;
	CListBox	m_ctrlReceipt;
	CEdit	m_ctrlTransactionNumber;
	CString	m_strTransactionNum;
	CString	m_strTransactionDate;
	CString	m_strCashierID;
	CString	m_strCustomerID;
	int		m_nQuantityReturning;
	//}}AFX_DATA

// Attributes
public:

// Operations
public:


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CReturnView)
```

```
	public:
	virtual void OnInitialUpdate();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	virtual void OnActivateView(BOOL bActivate, CView* pActivateView,
CView* pDeactiveView);
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CReturnView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
	//{{AFX_MSG(CReturnView)
	afx_msg void OnDone();
	afx_msg void OnTransid();
	afx_msg void OnSetfocusTransactionNumber();
	afx_msg void OnAdd();
	afx_msg void OnRemove();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	bool m_bHeader;
	bool m_bClearFlag;
	vector<int> m_vecIndexes;
	vector<int> m_vecReturnQuans;
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_RETURNVIEW_H__6DCAB687_84FE_11D3_A209_0000C0433AF8__INCLUD
ED_)
```

ReturnView.cpp

```
// ReturnView.cpp : implementation file
//

#include "stdafx.h"
#include "post.h"
#include "ReturnView.h"
#include "PostConstants.h"
#include "POSTDoc.h"

// View Includes
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// Gloabl Post Terminal
extern PostTerminal* g_pPostTerm;

///////////////////////////////////////////////////////////////////////
//////
// CReturnView

IMPLEMENT_DYNCREATE(CReturnView, CFormView)

CReturnView::CReturnView()
	: CFormView(CReturnView::IDD)
{
	//{{AFX_DATA_INIT(CReturnView)
	m_strTransactionNum = _T("");
	m_strTransactionDate = _T("");
	m_strCashierID = _T("");
	m_strCustomerID = _T("");
	m_nQuantityReturning = 0;
	//}}AFX_DATA_INIT
	m_bClearFlag = true;
}

CReturnView::~CReturnView()
{
}

void CReturnView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CReturnView)
	DDX_Control(pDX, IDC_Receipt2, m_Receipt2);
	DDX_Control(pDX, IDC_Receipt, m_ctrlReceipt);
	DDX_Control(pDX, IDC_TransactionNumber, m_ctrlTransactionNumber);
	DDX_Text(pDX, IDC_TransactionNumber, m_strTransactionNum);
	DDX_Text(pDX, IDC_ReceiptDate, m_strTransactionDate);
	DDX_Text(pDX, IDC_CashierID, m_strCashierID);
```

```
	DDX_Text(pDX, IDC_CustomerID, m_strCustomerID);
	DDX_Text(pDX, IDC_EDIT1, m_nQuantityReturning);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CReturnView, CFormView)
	//{{AFX_MSG_MAP(CReturnView)
	ON_BN_CLICKED(IDC_DONE, OnDone)
	ON_BN_CLICKED(IDC_TRANSID, OnTransid)
	ON_EN_SETFOCUS(IDC_TransactionNumber,
OnSetfocusTransactionNumber)
	ON_BN_CLICKED(IDC_ADD, OnAdd)
	ON_BN_CLICKED(IDC_REMOVE, OnRemove)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////////////////////////////////////////////////////////////////
//////
// CReturnView diagnostics

#ifdef _DEBUG
void CReturnView::AssertValid() const
{
	CFormView::AssertValid();
}

void CReturnView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}
#endif //_DEBUG

///////////////////////////////////////////////////////////////////////
//////
// CReturnView message handlers

void CReturnView::OnActivateView(BOOL bActivate, CView* pActivateView,
CView* pDeactiveView)
{
	//Get a pointer to the document
	CPOSTDoc* pDoc = (CPOSTDoc*)GetDocument();

	// TODO: Add your strDataialized code here and/or call the base
class
	if (m_bClearFlag == true)
	{
		m_ctrlTransactionNumber.EnableWindow(true);
		CFormView::OnActivateView(bActivate, pActivateView,
pDeactiveView);
		m_ctrlTransactionNumber.SetFocus();
		m_ctrlReceipt.ResetContent();
		m_Receipt2.ResetContent();
		m_strCashierID = "";
		m_strCustomerID = "";
		m_strTransactionNum = "";
		m_strTransactionDate = "";
```

```
            m_nQuantityReturning = 0;
            UpdateData(false);
            m_bHeader = false;
      }
      pDoc->SetTitle("Return Items");
}

void CReturnView::OnInitialUpdate()
{
      CFormView::OnInitialUpdate();

      // TODO: Add your strDataialized code here and/or call the base
class

}

void CReturnView::OnDone()
{
      double return_amount = 0;

      for(int i=0; i < m_vecIndexes.size(); i++)
      {
            return_amount += g_pPostTerm->ReturnItem(m_vecIndexes[i],
m_vecReturnQuans[i]);
      }
      //add on tax
      return_amount *= 1.05;
      m_bClearFlag = true;

      if(return_amount != 0)
      {
            char* temp = new char[64];
            sprintf(temp, "%7.2f", return_amount);
            CString strTemp = temp;
            strTemp.TrimLeft();
            AfxMessageBox("The Amount Owed to the Customer is: $" +
strTemp);
            delete temp;
      }
      if(g_pPostTerm->TransactionInProgress() == true)
      {
            m_vecIndexes.clear();
            m_vecReturnQuans.clear();
            g_pPostTerm->EndTransaction();
      }
      ((CMainFrame*)GetParentFrame())->SelectView(LOGIN_VIEW);
}

void CReturnView::OnTransid()
{
      CString strData;

      UpdateData(true);

      // Populate Transaction Information
      // Query DB and get info
```

```
      strData = g_pPostTerm-
>GetTransactions(atoi(m_strTransactionNum));
      // Parse out Date of Transaction
      m_strTransactionDate = strData.Left(strData.Find("@"));
      strData = strData.Right((strData.GetLength() - strData.Find("@")
- 1));
      // Parse out CashierID
      m_strCashierID = strData.Left(strData.Find("@"));
      strData = strData.Right((strData.GetLength() - strData.Find("@")
- 1));
      // Parse out CustomerID
      m_strCustomerID = strData;
      if(atoi(m_strCustomerID) <= 0)
      {
            m_bClearFlag = false;
            AfxMessageBox("That Transaction ID is invalid.");
            m_strTransactionNum = "";
            m_strCustomerID = "";
            m_strCashierID = "";
            m_ctrlTransactionNumber.SetFocus();
            UpdateData(false);
      }
      else
      {
            m_ctrlTransactionNumber.EnableWindow(false);
            m_bClearFlag = false;
            BeginWaitCursor();
            char* temp = new char[50];
            UpdateData(false);

            // Create customer
            g_pPostTerm->CreateCustomer(atoi(m_strCustomerID));

            UpdateData(false);

            //Create a new transaction
            g_pPostTerm-
>CreateTransaction(refund,atoi(m_strTransactionNum));

            // Populate the receipt
            if (!m_bHeader)
            {
                  sprintf(temp, "Point of Sale Terminal");
                  m_ctrlReceipt.InsertString(-1, temp);
                  sprintf(temp, "Cashier ID: %s", m_strCashierID);
                  m_ctrlReceipt.InsertString(-1, temp);
                  sprintf(temp, "Trans. ID: %s", m_strCashierID);
                  m_ctrlReceipt.InsertString(-1, temp);
                  sprintf(temp, "Date: %s", m_strTransactionDate);
                  m_ctrlReceipt.InsertString(-1, temp);
                  m_ctrlReceipt.InsertString(-1, "");
                  sprintf(temp, "%8s%16s%13s", "Qty.", "Item",
"Price");
                  m_ctrlReceipt.InsertString(-1, temp);

                  m_bHeader = true;
            }
```

```
            CString spec;
            char* left = new char[50];
            char* center = new char[50];
            char* right = new char[50];
            int x = g_pPostTerm->GetLineItemCount();
            for(int i=0; i < x; i++)
            {
                  spec = g_pPostTerm->GetLineItemInfo(i);
                  //Add spec to the list box
                  //Add it to the end of the list box by sending it -1
                  strcpy(left, spec.Left(spec.Find(",")));
                  spec = spec.Right((spec.GetLength() - spec.Find(",")
- 1));
                  strcpy(center, spec.Left(spec.Find(",", 0)));
                  spec = spec.Right((spec.GetLength() - spec.Find(",")
- 1));
                  strcpy(right, spec);

                  sprintf(temp, "%8s%16s%10s", left, center, right);
                  m_ctrlReceipt.InsertString(-1,temp);
            }

            UpdateData(false);
            delete temp;
            EndWaitCursor();
      }
}

void CReturnView::OnSetfocusTransactionNumber()
{
      // TODO: Add your control notification handler code here
      if (m_bClearFlag == true)
      {
            m_bHeader = false;
            m_ctrlReceipt.ResetContent();
            m_ctrlTransactionNumber.SetFocus();
            m_strCashierID = "";
            m_strCustomerID = "";
            m_strTransactionNum = "";
            m_strTransactionDate = "";
            UpdateData(false);
      }
}

void CReturnView::OnAdd()
{
      // TODO: Add your control notification handler code here
      int index;
      int nOriginalQuan, nNewQuan;
      CString strOriginalQuan, strTemp, strNewItem;
      CString strSelectedItem;

      UpdateData(true);
      //Gets the index of the currently selected item
      index = m_ctrlReceipt.GetCurSel();
```

```
	if (index <= 5)
	{
		m_bClearFlag = false;
		AfxMessageBox("Please select a valid item to void!");
	}
	else
	{
		m_ctrlReceipt.GetText(index, strSelectedItem);
		//strOriginalQuan =
strSelectedItem.Left(strSelectedItem.Find(" "));
		strOriginalQuan =
strSelectedItem.SpanExcluding("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
		nOriginalQuan = atoi(strOriginalQuan);
		if((nNewQuan = nOriginalQuan - m_nQuantityReturning) < 0)
		{
			m_bClearFlag = false;
			AfxMessageBox("Cannot return more than originally
purchased!");
		}
		else if(m_nQuantityReturning == 0)
		{
			m_bClearFlag = false;
			AfxMessageBox("Cannot return zero items!");
		}
		else
		{
			m_vecIndexes.push_back(index);
			m_vecReturnQuans.push_back(m_nQuantityReturning);
			// Create new string and modify old string
			char* Temp1 = new char[10];
			char* Temp2 = new char[10];
			itoa(nOriginalQuan, Temp1, 10);
			itoa(m_nQuantityReturning, Temp2, 10);

			// Create Return String
			strTemp = strSelectedItem;
			strNewItem = strOriginalQuan;
			strNewItem.Replace(Temp1, Temp2);
			strTemp.Replace(strOriginalQuan,strNewItem);

			m_Receipt2.InsertString(-1,strTemp);

			// Modifiy Original String
			strTemp = strSelectedItem;
			itoa(nNewQuan, Temp2, 10);
			strNewItem = strOriginalQuan;
			strNewItem.Replace(Temp1, Temp2);
			strTemp.Replace(strOriginalQuan,strNewItem);


			// Insert Strings into Receipts
			m_ctrlReceipt.DeleteString(index);
			m_ctrlReceipt.InsertString(index, strTemp);

		}
	}
	m_nQuantityReturning = 0;
```

```
	UpdateData(false);
}

void CReturnView::OnRemove()
{
	//Maybe implement this
}
```

PostConstants.h

```
#ifndef POST_CONSTANTS_H
#define POST_CONSTANTS_H

const unsigned int sale = 101U;
const unsigned int refund = 102U;
const unsigned int cash = 103U;
const unsigned int credit = 104U;
const unsigned int check = 105U;

// View Identifiers
const UINT LOGIN_VIEW = 1U;
const UINT SALE_VIEW = 2U;
const UINT RETURN_VIEW = 3U;

//Set for at school
const CString sDBLocation = "h:\\FedkeJ\\Software
Engineering\\post\\Post97.mdb";

//Folder at home (Ryan)
//CString sDBLocation = "D:\\school\\Software Engineering
Design\\post\\Post97.mdb";

//Folder at home (Brenden)
//const CString sDBLocation = "S:\\Post\\Post97.mdb";

//Folder at home (Kevin)
//const CString sDBLocation = "c:\\Program Files\\Microsoft Visual
Studio\\MyProjects\\Software Engineering\\Post\\Post97.mdb";


#endif
```