Orbiz

Mobile Inventory Management System

Student Report

Prepared by Tony Lee (Version 0.1)
Auckland University
31 October 2003
Confidential

Table of Contents

Revision History

Date	Who	Comment
15 October 2003	Tony Lee	Initial draft for comments.
17 October 2003	Grant Archibald	Comments.
18 October 2003	Tony Lee	Restructure report
21 October 2003	Grant Archibald	Comments.
31 October 2003	Tony Lee	Final Report.

1. Executive Summary

This report documents the development lifecycle of a Mobile Inventory Management System project completed by a BTech student. The aim of this project is to give the student practical experience on real-life application development. The proposed Mobile Inventory Management System is based on two client applications running on Pocket PCs. They are the Inventory Management Tool and the Warehouse Management Tool.

- Inventory Management Tool - Allows users to retrieve up-to-date stock information while they are on-the-road. It returns stock level information, site information, and location information related to a particular product.

- Warehouse Management Tool – Allows users to update the database as stocks are moving between locations within the system. It also provides a convenient way for users to do stocktaking.

This report summarizes the options and strategies used during the development, and it gives comparisons between different approaches and technologies adopted in the project. A section summary is written for each of the two client applications, and a discussion section is given at the end of this report outlining any challenges encountered during the project.

This report is divided into the following sections.

- Project Overview
- Research / Considerations
- Inventory Management Tool
- Warehouse Management Tool
- Discussions / Challenges
- Conclusions

2. Document Structure

- **Executive Summary** – Gives an abstract about the project, and outlines each of the major sections in this document.
- **Project Overview** – Describes the objectives and the scope of the Mobile Inventory Management System project.
- **Research / Considerations** - Outlines relevant information regarding to the technologies used in the project.
- **Inventory Management Tool** - Details the development of the inventory management tool.
- **Warehouse Management Tool** – Details the development of the warehouse management tool.
- **Discussions / Challenges** – Gives a summary on the approaches used and any challenges encountered during the project.
- **Conclusions** – Summarizes the knowledge gained from the project.

3. Project Overview

3.1 Introduction

During the past decade, the I.T. industry has grown rapidly. An increasing usage of computing technologies has changed the business environment dramatically. In particular, the demand of mobile applications has growth rapidly. For example, personal mobile phones and Pocket PCs are becoming popular tools for business usage. Nowadays, it is relatively inexpensive to own a mobile device, these devices are becoming more user-friendly and easy to use. As a result of these potentials, many businesses invest money towards mobile technologies, and hence accelerate the growth in mobile application development.

3.1.1 Company Info

Orbiz International Limited is a leading New Zealand integrator providing services to large corporate clients in the areas of consulting, strategic development, architecture, analysis and implementation of leading edge mobile solutions. It builds and deploys business mobility solutions using Microsoft .NET technology and other related technologies. Orbiz works closely with other companies such as Microsoft, Vodafone and Telecom to provide mobility solutions to their customers.

3.1.2 Motivations

As part of the ongoing strategy, Orbiz plans to introduce a mobile application which provides integrated functionalities on Pocket PC based devices. This software consisted of many different modules, such as the job dispatching module, the GPS integration module, the inventory management module, and the knowledge management module. The BTech project proposed and sponsored by Orbiz is the development of a mobile inventory management system, which forms the basis of the inventory management module of the software application.

3.2 Project Objectives and Scope

This project aims to develop an application that runs on a Pocket PC, the application consists of two software components.

The first component is an “Inventory Management Tool”. It can be used by anyone in the system. The function of this tool is to allow users to retrieve stock level info and location info related to a product. The details for this application will be discussed in Section 5.

The second component is a “Warehouse Management Tool”. The function of this tool is to allow users to keep track of a product when it is moving from one location to another. It also provides a convenient way for users to do stocktaking. The details for this application will be discussed in Section 6.

The mobile inventory management application is expected to work with the local database that resides on the Pocket PC. The data inside the local database will be synchronized with the main database server via networking technologies.

The following figure gives an abstract view of the project.

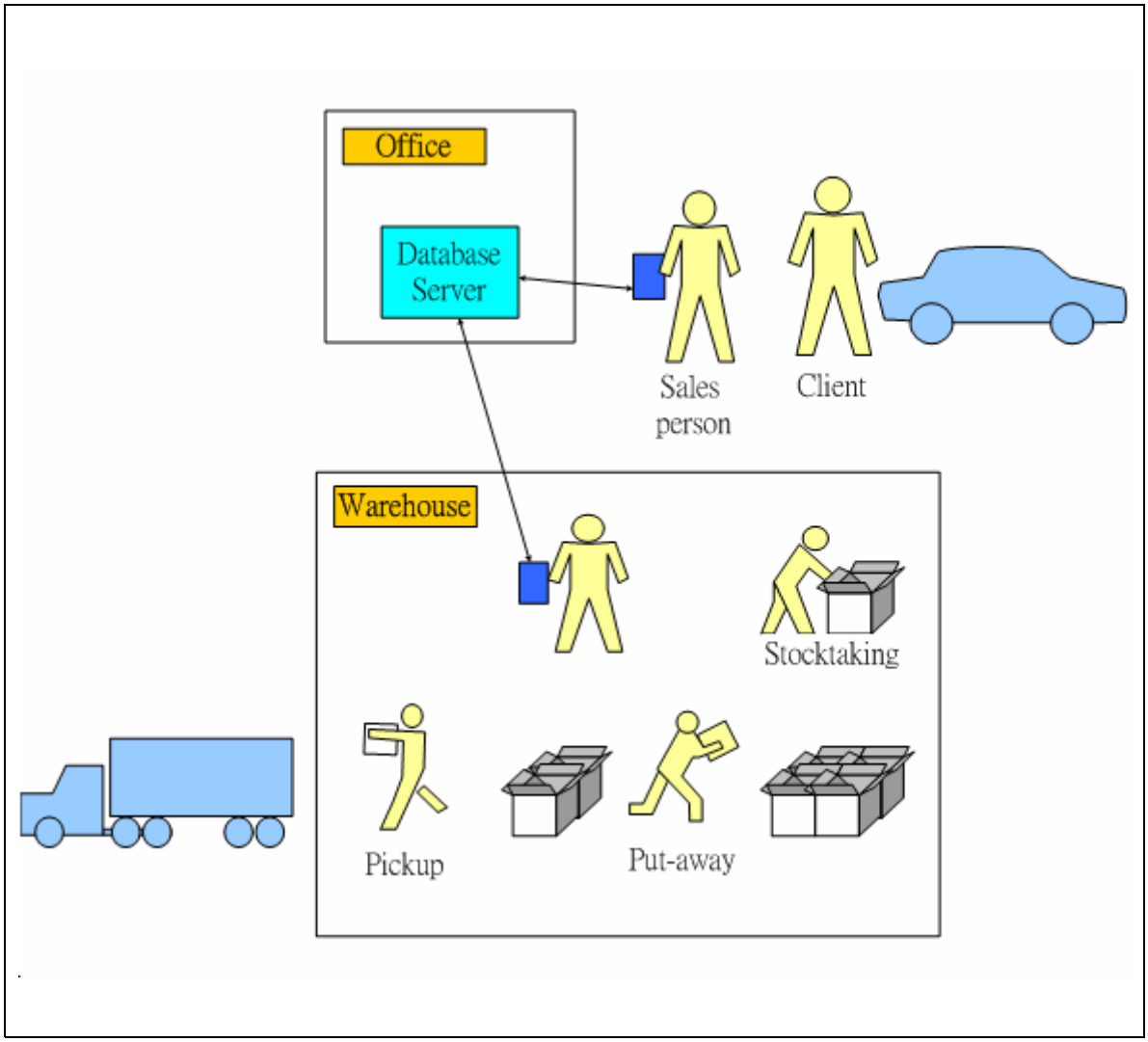


Figure 3.2 – Relationship between different actors in the system

3.3 System Architecture

The system architecture of the mobile inventory management application is very simple. The idea is to have the application runs on a Pocket PC. The application has no direct connection to the main server. Instead, it retrieves data and makes any necessary updates onto the local database. The application also provides the facility to init synchronization between the local database and the main server, which means the user can decides when to do the synchronization as long as a network connection exists between the device and the server.

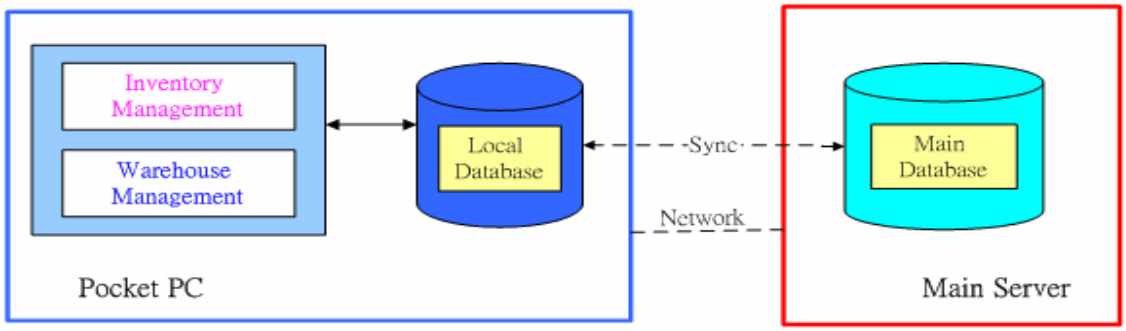


Figure 3.3 – System architecture of the Mobile Inventory Management System.

3.4 Existing Data Structure

An existing SQL Server database will be used as the basis of this project. This database is a simple database that contains 14 tables. The existing database has the ability to capture customer details, customer order details, product information and system user information. This database will be enhanced to capture data that are necessary for the mobile inventory management system.

3.5 Project Template

The screen design shown on the right is the initial template used for the project.

There are three buttons in the template.

The “Inventory” button is used to initialize the Inventory Management Tool. The “Warehouse” button is used to initialize the Warehouse Management Tool. These two buttons have no action associate with them at the start of the project.

The “Synchronize” button will open the “Synchronization Form”, which starts synchronization between the local database and the main server. The code to do with synchronization is already implemented with the template. The synchronization process will be discussed in Section 4.5.

4. Research / Considerations

4.1 Pocket PC

A Pocket PC is lightweight device similar to a Palm Pilot. It is powered by a Microsoft Pocket PC operation system, which is a derivative of the Microsoft Windows CE operating system. It is readily available on the market and it continues to grow in its popularity. Figure 4.1 shows an example of a Pocket PC, it is one of those popular Pocket PCs available on the market.

Although the processing power of a Pocket PC is still far behind comparing to a laptop or a desktop computer, but it is growing rapidly. The primary advantage of Pocket PC is the ease to carry, so that people can use it anywhere at anytime. This advantage makes it one of the best devices used in business mobility solutions.

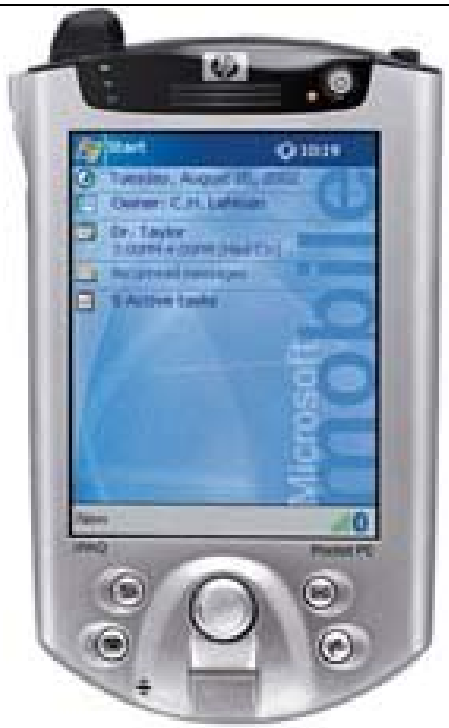


Figure 4.1– A typical Pocket PC.

Other features of a Pocket PC are listed below.

- Highly extensible with other external devices, such as wireless network adaptors and external keyboard.
- Easy to use user interface.
- Many built-in applications.
- Can synchronize with a desktop computer, download emails, files etc onto the Pocket PC.
- Some Pocket PCs have enhanced features, such as barcode reading.
- Instead of having a mouse pointer on the screen, the user taps on the screen using a stylus.
- Some Pocket PCs are integrated with Bluetooth, biometric fingerprint reader, Infra Red sensor and microphone etc.

4.1.1 Limitations to a Pocket PC

As far as this project concerns, there are a number of critical limitations to a Pocket PC. These limitations made the development of this project differ from traditional application programming. A list of limiting factors is described below:

Limitations	Descriptions
Limited processing power	A normal Pocket PC on the market has a 400MHz processor. This is relatively slow compared to a normal desktop computer. For this reason, it is very critical to choose the most efficient algorithm to implement a mobile application.
Limited memory and storage	A normal Pocket PC has 32-64Mb built-in memory. This is not a large storage and it limits the size of the data file stored on the device. Furthermore, physical memory is shared among the files in the Pocket PC and the memory consumed when an application is run. That means the more files you have on the device, the less memory you have to run an application. However, external memory cards are very common nowadays, they can be used to store large amount of data, which helps to free the physical memory of the Pocket PC. An example of external memory is a SD card.
Limited screen size and screen resolution	The screen resolution differs between different brands of Pocket PC, a common resolution supported by most Pocket PC is 240 x 320 pixels. That means it is hard to display large amount of text and graphics on the screen. Some Pocket PCs used in the industry doesn't display colour, which impacts the design of the user interface.
Input method	Without external add-ons, a Pocket PC has no physical keyboard. That means it is critical to provide an easy to use interface for speedy data entry. Other input methods such as handwriting recognition and voice recorder are usually available on the device.
Power consumption	Compared to a desktop computer, the battery life of a Pocket PC is relative short. The amount of battery consumption depends on the usage of the processor. In most cases, the battery needs to be recharge after a day of usage. Or it will last for a few days if the Pocket PC is idle most of the time. It is very important to keep the battery charged up, it's because all the programs and files are stored in the physical RAM, and they will be lost if the battery is flat out. Therefore, external memory card are need to keep important data persistent.

4.1.2 Various types of Pocket PC

There are various types of Pocket PC available in the market. Figure 4.1.2 shows some Pocket PCs used in the industry.

Figure 4.1.2 - Different types of Pocket PC

4.2 .NET

There are a few programming technologies used widely for mobile application development. The most popular technologies are the Microsoft .NET technology and Java 2 Micro Edition (J2ME). While it is arguable to say which technology is better, the choice of technology really depends on the device running the application and the type of application to be developed.

The Microsoft .NET technology is chosen for this project, and Visual C# will be used as the primary programming language. This is because a Pocket PC is powered by a Windows CE based operating system, which is highly integrated with the .NET Compact Framework. The .NET Compact Framework is a subset of the full .NET framework, it is specially designed to operate on small devices that are limited in resources. The .NET Compact Framework requires small amount of memory to operate, it is very good for a device like a Pocket PC. Although many libraries and controls are not supported by the .NET Compact Framework, but many commonly used libraries are available to the developers, so that they can write up their custom controls to provide the specific functions.

From the choice of programming language, Microsoft Visual Studio .NET 2003 (VS.NET) is the primary tool used in this project. VS.NET is a RAD tool, which offers a rich environment for application programming. It has a built-in Pocket PC emulator, which facilitates testing and debugging of application without having physical connection to a real Pocket PC.

4.3 Networking Technologies

In order to synchronize the local database with the main database, the Pocket PC must be able to connect to the main server via networking. There are a number of ways a Pocket PC can connect to the main server. A few examples are list below.

- USB connection – the simplest way to connect to the server machine is to use a USB cradle. This is usually the fastest and most economical way to synchronize the databases. However, the user needs to connect the Pocket PC to the server machine physically, that means it is not possible to have real-time synchronization when the user is distance apart from the main server.
- Bluetooth – Bluetooth is a worldwide radio standard developed to allow devices to communicate wirelessly over short distances. This technology is usually used if the Pocket PC and the main server is less than 10m apart.
- WLAN 802.11 – Wireless LAN can be used to connect the Pocket PC with the main server, wireless network cards are needed in order to achieve this. This technology usually supports up to 100m in distance, but the distance and the transmission speed is usually limited by the surrounding noise level.
- GSM / CDMA – These services are provides by telecommunication companies, they are used if the mobile device is very far away from the main server.

4.4 Database Servers

There are several well known database servers available in the market, for examples, Oracle, MySQL server and Microsoft SQL Server 2000. Each of these databases has its own advantages and disadvantages.

The choice of SQL Server 2000 seems to be suitable for this project. It is because this project is to be built on top of the existing data structure, which is built using SQL Server.

Features of using SQL Server as the backend system are listed below:

- SQL Server is highly compatibility with other Microsoft products. It can be easily integrated with a .NET application and the operating system of a Pocket PC.
- Through the use of SQL Server CE, SQL Server can replicate a database from the main server onto a Windows CE device.
- A Pocket PC has support for SQL CE, and it has a SQL CE Query Analyzer that allows users to query the database. After the SQL CE database synchronizes with the main server, a file named with an ".sdf" extension is stored on the Pocket PC. This file contains all the data in the original database, and it can be copied to an external memory card when needed.
- SQL Server supports three types of publication, they are the "Snapshot publication", "Transactional publication" and "Merge publication". In "Snapshot publication", the server periodically updates subscriber data with an updated snapshot. In "Transactional publication", changes to the database are sent to the subscriber as they happen. In "Merge publication", data can be updated at the main server or any subscriber, these updates are merged when the user decides to synchronize the databases.
- It supports a lot of database functions such as views, constraints, stored procedures, triggers, user defined functions etc.
- It also supports batch execution of SQL commands, which is very useful when making up a dummy database.

Other advantages of SQL Server 2000 include:

1. It is an enterprise relational database management and analysis system.
2. It is widely used in many businesses.
3. It handles massive amount of textual and non-textual data.
4. It handles massive amount of transactions.

4.5 Synchronization

Figure 4.5 shows the replication process between the SQL CE database and the main SQL Server.

The SQL CE database engine manages the local database on the Pocket PC. It tracks all the database records that are inserted, updated or deleted.

When synchronization occurs, the SQL CE Client Agent communications with the SQL CE Server Agent via IIS, which in turn merge the SQL CE database with the main server. Any updates to the databases will be synchronized on both sides.

In order to initialize synchronization, the client application needs to configure settings such as the IP address of the main server, the name of the publication and other authentication details etc, so that the Pocket PC can connect to IIS and complete the replication process.

The actual code used to implement the "Synchronization Form" is not discussed here, because it is a code module supplied by Orbiz along with the template.

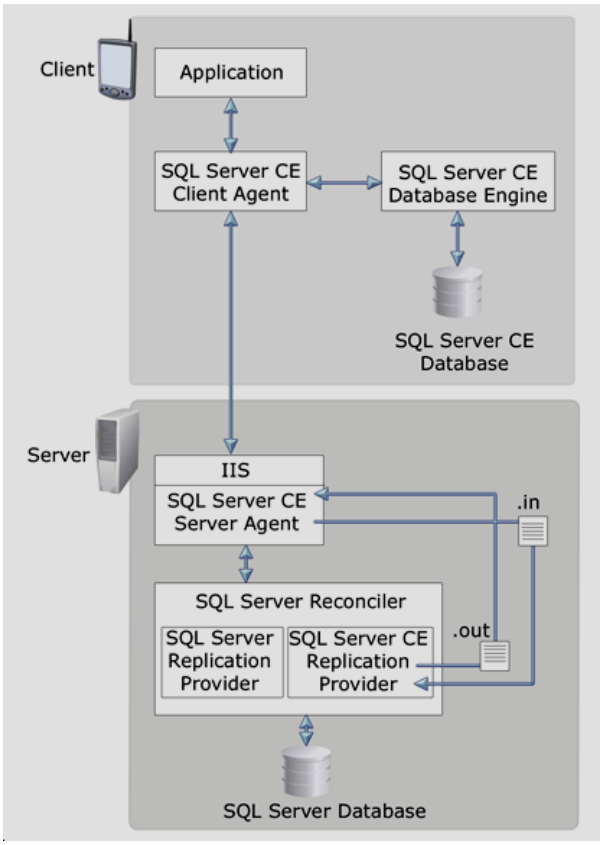


Figure 4.5 - A diagram taken from the MSDN library. Shows the replication process between SQL Server and its subscribers.

5. Inventory Management Tool

5.1 Overview

This section addresses the development of the inventory management tool. The development lifecycle of the inventory management application is divided into five different phases as shown below.

- Requirement Analysis
- Design
- Implementation
- Testing
- Deployment

Each of these phases will be discussed in details, a section summary will be provided at the end of this section summarizing the pros and cons of the strategies applied.

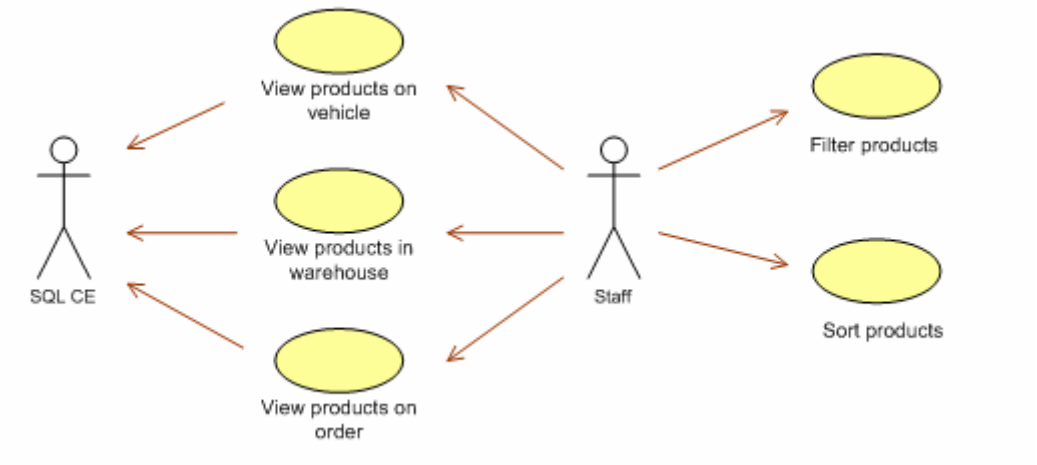
5.2 Requirement Analysis

This section specifies the objectives of the inventory management tool. It lists all the functional and non-functional requirements of the application, and any assumptions made during the development.

5.2.1 Objectives and Scope

The development of the inventory management tool forms the first part of this project. The purpose of this application is to allow users to retrieve stock level information via a Pocket PC. That means users can use this tool to find out the amount of stock available in the system, and how many of them are on orders from suppliers. This tool also allows users to find out where a particular product is located in the system.

5.2.2 Use Case Diagram



5.2.3 Assumptions

As it is impossible to have one application that suits all kind of businesses, the following assumptions are made when developing the inventory management tool.

- One type of product can be stored on many different sites.
- One physical site can store more than one type of product.
- A site is itself divided into many different locations.
- Each location is simply a labelled place within a site.
- Only one type of product can be stored at each labelled location.

The example given below illustrates the meaning of the above assumptions.

- In the system shown by Figure 5.2.3, there are two types of product, namely, Baked Beans and Wine.
- There are 300 bottles of Wine, all the 300 bottles of Wine are stored at Shelf_B of Warehouse_ABC.
- The total amount of Baked Beans recorded in the system is 100 cans, in which 60 cans are stored in Warehouse_ABC, 40 cans are stored in Warehouse_XYZ.
- The 60 cans of Baked Beans in Warehouse_ABC are actually stored at 2 different locations within the warehouse, namely, Shelf_A and Delivery_Bay_C.
- The 40 cans of Baked Beans in Warehouse_XYZ are actually stored at 3 different locations within the warehouse, namely, Shelf_A, Box_A, and Belivery_Bay_A.

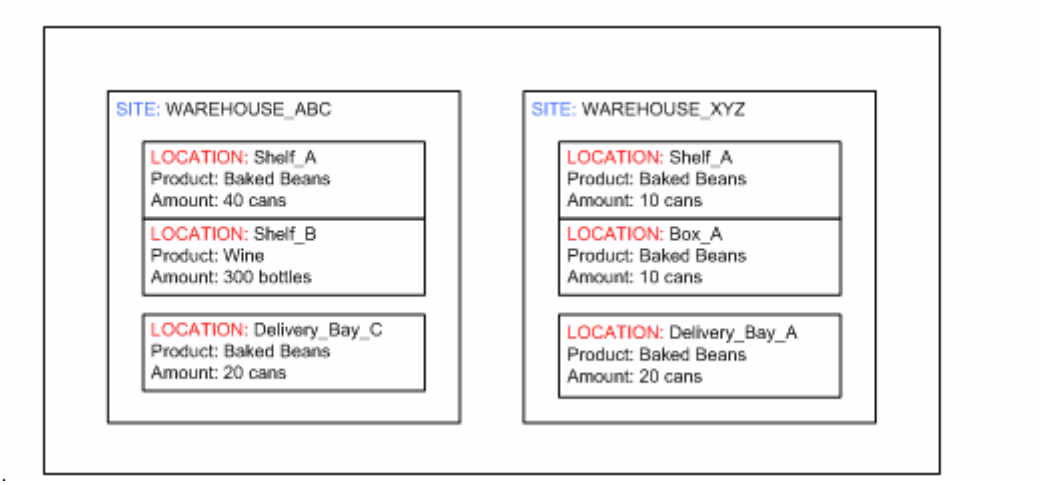


Figure 5.2.3 – A simple inventory system example.

5.2.4 Functional Requirements

- Retrieves product info from the SQL CE database.
- Retrieves site info and location info from the SQL CE database.
- Provides the facilities to list product info.
- Provides the facilities to filter and sort the product list.
- Provides the facilities to find out on which site a particular type of product is stored.
- Provides the facilities to find out the exact location within a site where the product is located.

5.2.5 Non-functional Requirements

Additional non-functional requirements are list below.

- Easy and speedy to use.
- User interfaces are clearly displayed within the limited size screen.
- Able to run on a Pocket PC with very limited amount of memory.
- Able to handle large amount of data.
- Able to incorporate with the Orbiz framework.
- Able to work with large amount of data and transactions.

5.2.6 Tasks

There are a number of tasks associated with this part of the project.

- Prepare a design specification for the inventory management tool.
- Integrate the inventory management tool on top of the existing database.
- Produce meaningful data in the database for testing purposes.
- Identify any weaknesses or problems encountered, and try to improve it on the second part of the project.

5.3 Design

The design phase is a very important phase in the development lifecycle. It involves the design of application flow, how users interact with the application, the layout of user interfaces and how data are stored in the database.

5.3.1 Application Flow Design

The flow design of the inventory management tool is described by Figure 5.3.1. The features of each screen are briefly described in the diagram, and they will be described in details in Section 5.3.2.

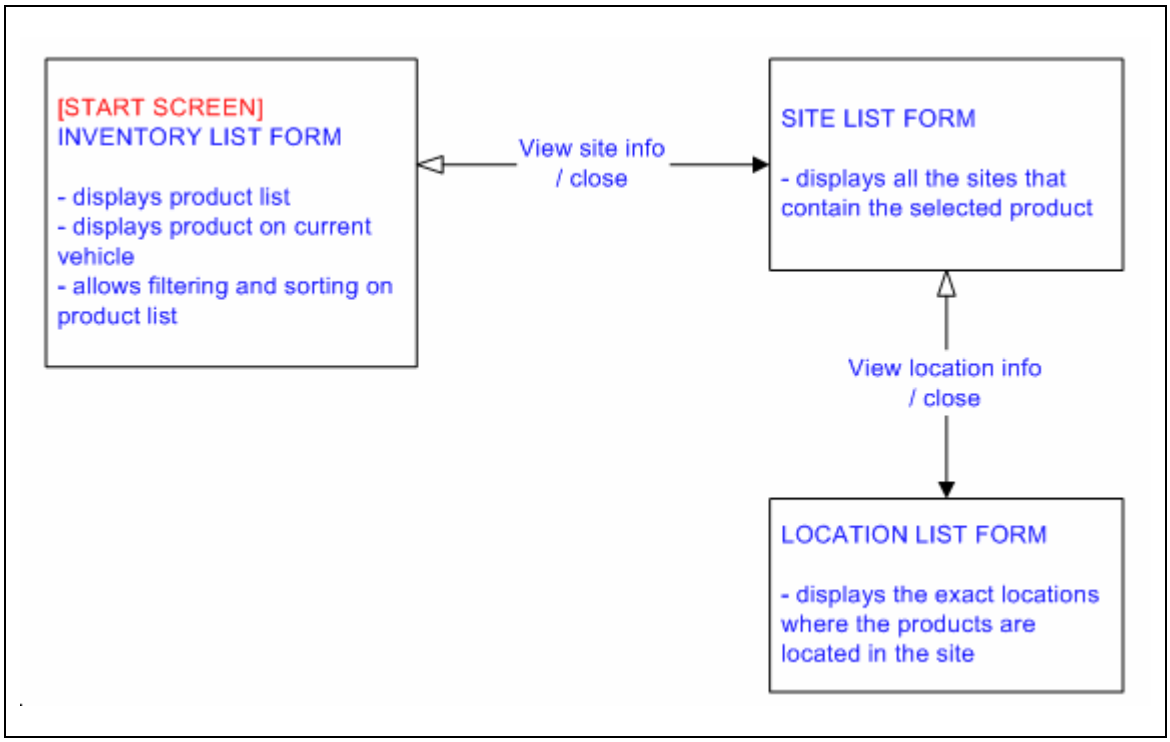


Figure 5.3.1 – Flow diagram of the inventory management tool.

5.3.2 User Interface Design

A high emphasis is put on the design of the user interface, it's because the application is expected to run on a Pocket PC, and the screen size of a Pocket PC is quite small compared to other displaying units. This section describes the screen designs of the Inventory Management Tool.

Note that this section shows the main screens of the inventory management tool only. For other screens of the application, please refer to Appendix I at the end of this report.

5.3.2.1 Inventory List Screen

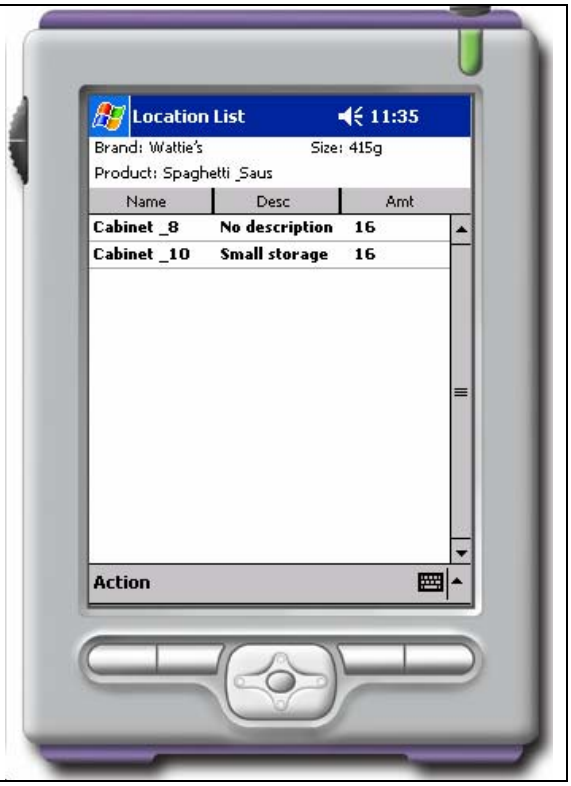
- The inventory list screen is the start screen of the inventory management tool.
- The inventory list is initially empty, there are three combo boxes at the top of the screen, and they are used as filter boxes of the product list. The filters used in this application are the "Brand" filter, the "Product" filter and the "Size" filter.
- There are three column header controls under the filters, named "Brand", "Product" and "Size" respectively. When the user clicks on a column header, the product list will be sorted in the specified order.
- The user can narrow down the product list by changing the three filters. Once a filter is applied, the combo boxes are automatically updated to show the available items only. This allows the user to search through the product list very quickly.
- Products that fulfil the filtering criteria will be displayed in the multi-line list view. Each item in the list shows the brand name, product name, product size, total number of stock available and the amount of stock on order.
- The user can clear the screen and search again using the "Clear" option, or by setting the filters to "nothing".
- The user can view the site info related to a product using the "Site Info" option under the context menu [as shown by the figure on the right]. Or by selecting "Site Info" under the "Action" menu.
- Note that there is a "Vehicle" tab, which is a speedy way to show all the products reside on the currently vehicle. [used by the delivery person]

5.3.2.2 Site List Screen

- The site list screen is shown when the user decided to view the “Site Info” of the selected product. [See Section 5.3.2.1]
- The purpose of the site list screen is to show all the sites that contain the selected product, and the corresponding stock amount on that site.
- A site is considered to be “in warehouse”, “on order” or “on vehicle”, therefore, three tab pages are needed to display these information.
- As stated in Section 5.2.3, a site is subdivided into many smaller locations, so, the user can view the location info using the “Location Info” option under the “Action” menu, or by selecting the “Location Info” under the context menu [as shown by the figure on the right].

5.3.2.3 Location List Screen

- The location list screen is shown when the user decided to view the “Location Info” of the selected product.
- The purpose of the location list screen is to show the exact location where the stock is located. It also gives info on the amount of product stored at the particular location.



5.3.3 Database Design

As mentioned in Section 3.4, the Mobile Inventory Management System project uses the existing data structure as the base database. There are a number of tables stored in the database. Since SQL Server is a rational database, all tables in the database are linked together through the use of primary key and foreign key.

Figure 5.3.3 shows the data tables added to the original database that are used by the inventory management tool. These additional tables have been added because the original database does not capture the storage location of a product. These tables are necessary in order to record site information that is required by the inventory management tool.

As far as this project concerns, only the data tables related to the inventory management system will be discussed in details.

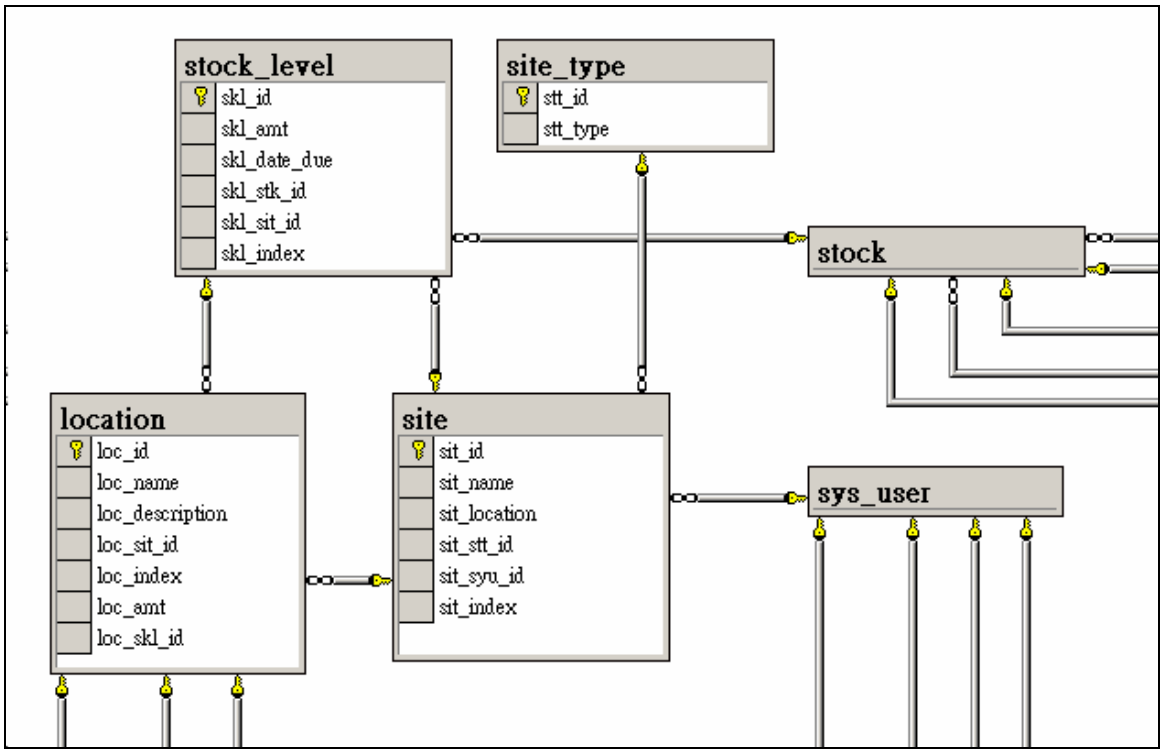


Figure 5.3.3 – Data tables used by the inventory management tool.

As described in Section 5.2.1, the inventory management tool has the ability to search through the product catalogue, and to find out the stock level of a particular product at a particular location of a site. If a product is on order from a supplier, this tool has the ability to retrieve the date due of the order as well.

In order to achieve this, the following data are required in the database.

- The location info of a site.
- The sites where a particular product is stored.
- The amount of stock stored in each location.
- In case the site is “on vehicle”, the details of the delivery person must be recorded.
- In case the site is “on order”, the due date needs to be recorded.

Four additional tables are added to the database, they are the “stock_level” table, the “site” table, the “site_type” table and the “location” table. Each of these tables is described below.

Table	stock_level
Descriptions	The purpose of this table is to resolve the many-to-many relationship between the stock table and the site table. It holds the stock “amount” on each site and the “date due” if the product is on order.

- skl_id – the primary key of the stock_level table. Identify a particular product on a particular site.
- skl_amt – the amount of stock stored in the site.
- skl_date_due – the date due for an order from supplier. If the product is not on order, the value of this attribute will be null.
- skl_stk_id – the foreign key from the stock table. Identify a particular product.
- skl_sit_id - the foreign key from the site table. Identify a particular site.
- skl_index – a numbered index to identify the stock_level entry.

Table	site
Descriptions	The purpose of this table is to identify a site. For example, “ABC warehouse” in “Auckland”.

- sit_id - the primary key of the site table. Identify a particular site.
- sit_name – the name of the site.
- sit_location – the location of the site.
- sit_stt_id – the foreign key from the site_type table. Identify a particular site type.
- sit_idnex – a numbered index used to identify the site.

Table	site_type
Descriptions	The purpose of this table is to identify a type of site. As long as the inventory management tool concerns, there are only three types of site, they are “in warehouse”, “on vehicle” or “on order”.

- stt_id – the primary key of the site type table. Identify a particular type of site.
- stt_type – the name of the site type.

Table	location
Descriptions	A site is subdivided into many labelled locations. Each location has an unique ID and can only store one type of product.

- loc_id – the primary key of the location table. Identify a particular location in a site.
- loc_name – a labelled name give to the location.
- loc_description – a description to the location.
- loc_sit_id - the foreign key from the site table. Indicates where this location belongs.
- loc_index – a numbered index given to the location.
- loc_amt – the amount of stock stored at that location.

5.4 Implementation

This section describes how the Inventory Management Tool was implemented. It describes the reasons for using the selected approach, the components used in the implementation, and some of the classes written to perform the operations.

5.4.1 Approaches

The strategy used to implement the inventory management tool is the simple, traditional coding approach. That means most code are written inside a single class, and there is no separation between user interface and business logics for each screen. Note that each screen listed in the design phase still owns a standalone windows form, and those codes that are shared between the screens are separated out into new classes.

This coding style is chosen because the inventory management tool is actually a querying tool, the business logic in behind is quite simple. Therefore, using the traditional approach allows speedy development.

As the inventory management tool needs to obtain stock info, site info and location info, a simple way to retrieve these data is to construct a SQL statement from the text in the filters, then make a connection to the SQL CE database, and retrieve the result set using a DataReader object or a DataSet object.

5.4.2 Components

Using Microsoft Visual Studio .NET 2003, the user interface can be easily layout. Each screen designed in Section 5.3.2 is implemented using a windows form provided by Visual Studio. While most of the components required are available from the .NET Compact Framework, some of the important components are however not available. Furthermore, some of the default components provided by .NET Compact Framework are limited in features. Due to this fact, custom controls are written in order to support the specific features required by the inventory management tool.

5.4.2.1 Default components

Listed below are the default components used in the inventory management tool.

Component	Windows Form
Descriptions	The base class of the screens. Acts as a container of other components.
Event handlers	The “Load” event is triggered when the form is loaded, it is used to set the initial values to the form, and initialises other components.

Component	MainMenu Control / MenuItem Control
Descriptions	The main menu control are associated with the form, they are used to implement the “Action” menu in each screen.
Event handlers	The “Click” event is triggered when the user taps on a menu item, it is used to redirect users to the “Site List” screen and the “Location List” screen. It is also used to close the application.

Component	Label
Descriptions	Labels are simply used to display static info on the screen.

Component	ContextMenu Control / MenuItem Control
Descriptions	Similar to the MainMenu control, but it is associated with a list view control in the application.
Event handlers	The context menu will pop up when the user presses & holds the stylus on the screen. It opens the “Site List” form and the “Location List” form depending on the selected item in the list view control.

Component	Tab Control / TabPage Control
Descriptions	These controls allow the user to switch between different tabs in a single form.
Event handlers	When the selected tab page changes, the product list and the filter boxes will be updated accordingly.

Component	Combo Box
Descriptions	Combo boxes are used as filter boxes in the application.
Event handlers	The “SelectedIndexChanged” event is triggered when the user changes the selected item in the filters, which in turn updates the list view and the items in the filter boxes.

5.4.2.2 Custom Controls

Listed below are the custom controls built for the inventory management tool. Detailed descriptions are given. These controls are used through out the project.

Component	Task List Control
Descriptions	By default, Visual Studio provides a "ListBox" control. However, the list items inside the default "ListBox" control doesn't support "multi-line text", which is very inconvenient to use when a lot of info need to be displayed for each item. For example, in Section 5.3.2.1, each item in the product list has to display the brand name, product name, size, amount on hand and the amount on order. There is no way to fit all this info into one line of text. Even if it is possible to fit all the text within one line, the user will not be able to read the info clearly. Therefore, the "Task List Control" is written to support a multi-line list view, which is used across difference screens in the project.
Event handlers	The "Task List Control" itself doesn't react to any system event. It changes the selected index when the user taps on an item in the list. An entry with speciality will be highlighted with other colours to alert the user. The behaviour of the "Task List Control" is very similar to the default "List box" control.

Component	Column Header Control
Descriptions	The "Task List Control" described above doesn't have column headers associate with it, this is needed in order to tell the user what info is displayed by the columns. They are also needed to sort the list items in order. An initial thought would be to use a "Button" control to act as the column head. However, in .NET Compact Framework, the text alignment property of a button is not available, and the text will not be clearly shown if the button width is too small. Therefore, the "Column Header Control" is written to support text alignment, and it will trim column header label if the header width is too small. Furthermore, given that the screen width of a Pocket PC is usually 240 pixels, experience shows that it is not a good idea to have more than three columns in a "Task List Control", having more columns will make the screen very crowded, hence makes it harder for the user to read the text in the list view. Experience also shows that the use of horizontal scrollbars is not recommended. By default, .NET Compact Framework doesn't have any splitter control provided. Therefore, the "Column Header Control" must be able to use to adjust the column width of the columns inside the "Task List Control".
Event handlers	The "Click" event is triggered when the user taps on the column header. In the inventory management tool, this is used to sort the items in the "Task List Control". The sort order alternates between ascending and descending. When the user taps and presses on the boundary of a column header, the splitter function is active. The "MouseDown" event is used to capture the initial tapped position. When the user lifts the stylus off the screen, the "MouseUp" event is triggered, which in turns adjusts the width of the "Custom Column Header" control and the columns inside the "Task List Control".

5.4.3 Code Samples

It is impossible and irrelevant to include all the code used to implement the inventory management tool. Therefore, extractions of code used in the application are described in this section.

5.4.3.1 Code Sample I – Load data from SQL CE

Descriptions	(1) Listed below is the method used to retrieve data from the database. The resultant data is used to update the items in the filter boxes. (2) A similar implementation is used to retrieve product info, site info, and location info.

```
public object[] ReloadFilterItems(string brand, string product, string size, string attribute,
    bool allowCache)
{
  ArrayList output=new ArrayList();

  if(allowCache) {  //NOTE (1)

    string key="["+brand+"|"+product+"|"+size+"]["+attribute+"]";
    if(allowCache && cachedData[key]!=null)
    {
      return new object[] { null, (ArrayList)cachedData[key] };
    }
    else { cachedData[key]=output; }
  }
  string errorMessage=null;
  SqlCeDataReader dr=null;

  SqlCeConnection connection=SqlDataManager.GetInstance().GetConnection(); //NOTE (2)
  connection.Open();
  try
  {
    string sqlStatement="...";  //NOTE (3)

    output.Add("");
    dr=(new SqlCeCommand(sqlStatement, connection)).ExecuteReader(); //NOTE (4)

    while(dr.Read()) { output.Add(dr[attribute].ToString()); }  //NOTE (5)
  }
  catch(Exception ex) { errorMessage=ex.ToString(); }  //NOTE (6)
  finally
  {
    if(dr!=null) { dr.Close(); } //NOTE (7)
    if(connection!=null) { connection.Close(); }
  }
  return new object[] {errorMessage, output}; //NOTE (8)
}
```

Notes	(1) In the code segment, the “cachedData” variable is a HashTable. If “allowCache” is set to “true”, then the app first checks whether the required data exists in the cache, and returns the cached copy if it exists. The required data is retrieved from the database if it doesn’t exist in the cache, and a copy of the data will be saved in the memory. In the inventory management tool, the cache function is disabled to save memory, since the same query rarely appears twice.

	(2) The SQL connection is provided via the “SqlDataManager” object, this class is implemented using the singleton pattern. The class is used to provide SQL connection to the local SQL CE data file through out the project. (3) The building of SQL statement uses the input parameters passed from the caller of this method. It is not shown here because it is a long operation and requires many lines of code. (4) The “DataReader” class is used to read data from the SQL CE data source. “DataReader” object is a forward-only reader, it doesn’t read the whole dataset into memory before processing. Instead, it keeps a current pointer to the result set, and returns the result as the application iterate through the result set. (5) The while loop is used to iterate all the data in the result set. (6) In case of error, the error message is sent back to the application, and the app decides the appropriate actions to take. (7) The “finally” section makes sure all the opened connections are terminated. This is necessary because a “SqlCeException” will be generated if a connection is opened twice. (8) The result set is returned to the caller of the method. It will update the filters correctly.

5.4.3.2 Code Sample II – Column Header Control

Descriptions	(1) Listed below is the implementation of the custom column header control. A lot of code in the sample is converted into readable pseudo code (highlighted in brown), it's because the original code is too long to include here. (2) The "Column Header Control" is used in both the inventory management tool and the warehouse management tool of this project. It is necessary in order to display the "Task List Control" correctly, and allows users to adjust the column width.

```
public class ColumnHeaderControl: System.Windows.Forms.Control   //NOTE (1)
{
  public ColumnHeaderControl(int totalWidth, int numOfCol, int minWidth,
    int hotSpotWidth)   //NOTE (2)
  {
    //DO: calculate default column width;
    //DO: Initialize headers;
    //DO: Initialize column labels;
    //DO: Initialize column splitters;
  }

  public bool HitTest(int x, int y)    //NOTE (3)
  { //DO: check whether the splitter is being clicked. Or the header is being clicked }

  public Point MouseDownPos    //NOTE (4)
  { //UPDATE: record the MouseDown position; }

  public void SetLabel(int index, string newLabel) //NOTE (5)
  { //UPDATE: change the label of a column header; }

  protected override void OnPaint(PaintEventArgs e)   //NOTE (6)
  {
    Graphics g=e.Grahpics;
    g.Clear(this.BackColor);
    g.FillRectangle(new SolidBrush(SystemColors.Control), 0, 0, this.Width, this.Height);

    for(int i=0; i<_hotSpots.Length; i++)
    {
      Rectangle r=_hotSpots[i];

      e.Graphics.FillRectangle(new SolidBrush(Color.Black), r.X+r.Width/2-1, r.Y,
        _splitterWidth, r.Height);
    }
    for(int i=0; i<_labels.Length; i++)
    {
      if(_labels[i]!=null && _labels[i]!="")
      {
        Font f=new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
          System.Drawing.FontStyle.Regular);

        string s=TrimText(_labels[i], e.Graphics, f, _cols[i].Width-8);
        int leng=g.MeasureString(s, f).ToSize().Width;

        g.DrawString(s, f, new SolidBrush(Color.Black),
          _cols[i].X+_cols[i].Width/2-leng/2-4, _cols[i].Y+2);
      }
    }
  }
```

```
  private string TrimText(string original, Graphics g, Font f, int colWidth)   //NOTE (7)
  {
    string theText=original;
    while( g.MeasureString(theText, f).ToSize().Width>colWidth && (theText.Length>1))
    {
      theText=theText.Substring(0,theText.Length-1);
    }
    return theText;
  }

  public void Recalculate(int x, int y)   //NOTE (8)
  {
    //DO: calculates the change in X position by subtract the (MouseUp.X - MouseDown.X);
    //UPDATE: sets the header size, and the splitter positions;
    //UPDATE: sets the splitter position;
  }

  public int GetColWidth(int i)    //NOTE (9)
  { //DO: return the width of a specific column; }

}
```

Notes

(1) Since the column header control is a windows form control, therefore it must extend the “Control” class. The “Control” class gives the column header control the ability to handle events (e.g. Click event), and to draw itself on the screen similar to other controls.
(2) The constructor of the column header control sets the initial width of the columns. It also sets the “hot spot” for the column splitters, and the text label of the column.
(3) When the user taps on the column header control, the “HitTest” method checks whether the user tapped on the column splitter, or the column header.
(4) If the tap position lies inside a splitter’s “hot spot”, then the tapped position is recorded.
(5) The “SetLabel” method allows users to change the text display in a specific column.
(6) The “OnPaint” method draws the column header control onto the screen. It sets the colour of the control, draws the control background and the column splitters. It then loops through the column labels and display the text within the available space.
(7) The “TrimText” method calculates the length of the displayed text, and trims the text so that it can be displayed correctly with the available column size.
(8) The “Recalculate” method is called when the user drags the column splitter. The “MouseUp” event is fired and gives the position where the user lifts the stylus off the screen. This method calculates the change in X position and adjusts the size of each column accordingly.
(9) The “GetColWidth” method returns the width of a specific column.

5.5 Testing

The first part of testing involves making up dummy data in the database, the SQL script shown is Appendix III is used to create random data. The resulting database contains 35 types of product, 30 sites, each site contains 70 locations, and there are 400 stock level entries linking these tables together.

Given that the business logics behind the inventory management tool are quite simple, and the current implementation is done using the traditional approach. Therefore, testing of the inventory management tool is done manually. Furthermore, there is no easy way to automate testing on user interfaces, so, "user experience" is the best way to test the usability of the application.

Listed below are the test cases used for the "Inventory List Screen". Note that the table below only shows a subset of test cases.

#	User Actions	Expected Results
1	User taps on the "Clear" option.	The inventory list is cleared. The filter boxes are reset and contain all the available items.
2	User sets the brand filter to "Wattie's".	All products with the brand name "Wattie's" are displayed in the inventory list. The items in the filter boxes are updated according to the inventory list.
3	User taps on a product in the inventory list.	The selected item is highlighted.
4	User selects a product in the list, then taps and holds the stylus on the screen.	The context menu shows up, with the "Site Info" option enabled.
5	User taps on the "Site Info" menu inside the "Action" menu when no product is being selected.	The "Site Info" menu is disabled, thus nothing happen.
6	User taps on the "Vehicle" tab.	The inventory list is cleared. The filter boxes are updated to show products on the current vehicle.
7	User taps on the "Product" column header.	Items in the inventory list are sorted by their product name.
8	User taps on the "Site Info" menu when a product is being selected.	The screen switches to the "Site List Screen".
…	…	…

5.6 Deployment

Deployment of the inventory management tool is relatively easy, because there is a "Build Cab File" option provided by Visual Studio .NET. When this option is clicked, Visual Studio .NET automatically generates installation files for the application. When the cab file is copied and run onto the Pocket PC, the application automatically installs itself onto the device and ready to use.

5.7 Section Summary

- There are five important phases in the development lifecycle, they are the requirement analysis phase, design phase, implementation phase, testing and deployment.
- The inventory management tool is a querying tool, which displays product info, site info and location info of a particular product. The data are originated from the SQL CE database located on the Pocket PC.
- The design phase involves the design of user interfaces, the design of application flow and database design.
- In addition to the tables in the existing SQL Server database, four tables are added in order to provide stock level info and location info of a product.
- The traditional programming approach is used to implement the inventory management application, an advantage of using this approach is the speed of development.
- Using the traditional approach, all the code related to the user interface, the business logic and the data model are stored in a single “Form” class. User actions will be handled by the method in the Form class. For example, when the user changes the brand filter in the “Inventory List Screen”, the “SelectedIndexChanged” event triggers a method call to retrieve data from the database, then updates the inventory list and displays the result back to the screen. All these functions are done within the Form class.
- The main disadvantage of using traditional programming is its inflexibility. Since all the business logics are implemented together with the user interface, that means if the same application is to be re-implemented with different user interface requirements, then the whole class file needs to be changed. That means the ability of code reuse is very low using the traditional programming style.
- A lot of default controls are provided by Visual Studio .NET, which satisfy most user interface requirements. However, custom built controls are needed in order to provide enhanced features to the user interface. Two custom controls are written for the inventory management tool, they are the “Task List Control” and the “Column Header Control”.
- Retrieval of data from a SQL CE database involves setting up a connection to the database, opening the connection, issuing a query command to the database, and iterate through the result set using a DataReader object. The database connection must be closed at the end of the operation, otherwise an exception will be raise when the application tries to open a connection again.
- The “MouseDown” event is triggered when the user taps on a control, and the “MouseUp” event is triggered when the user lifts the stylus off the control. The “splitter” inside the “Column Header Control” is implemented using these events.
- Testing is a very important phase in the development lifecycle. Since the inventory management application is written using the traditional programming approach, so, all the business logic are bounded by the user interface, and therefore, testing of the application is done manually.
- System testing has been done in Section 5.5. However, it gives no indication to the operating performance of the application.
- The deployment of the inventory management tool is simple, it involves building the installation cab file using Visual Studio .NET, and run it on the Pocket PC. The underlying Windows CE operating system on the Pocket PC must have support for .NET Compact Framework.

6. Warehouse Management Tool

6.1 Overview

This section addresses the development of the warehouse management tool. Again, the development lifecycle of the warehouse management tool is divided into five different phases as shown below.

- Requirement Analysis
- Design
- Implementation
- Testing
- Deployment

Each of these phases will be discussed in details, a section summary will be provided at the end of this section.

6.2 Requirement Analysis

This section specifies the goal of the warehouse management tool. It lists all the functional and non-functional requirements of this tool, and explains how this tool is used by a warehouse staff.

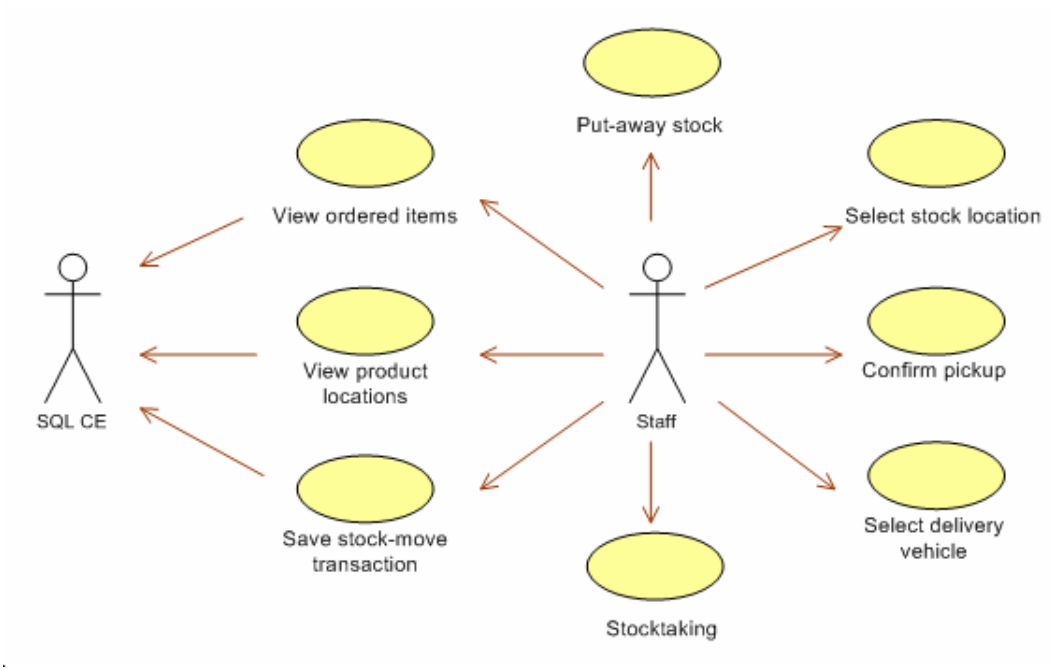
6.2.1 Objectives and Scope

The warehouse management tool is the second client application in the Mobile Inventory Management System project. The primary function of this application is to provide a convenient way to record the “stock-move” process occurred in the system. The meaning of the word “stock-move” simply implies moving stock from one location to another location. The second objective of the warehouse management tool is to allow staff members to do stocktaking on products stored at a particular location.

As it is far too complex to model all the stock movements occurred within a warehouse, the client application developed for the project is a mini version of a real-life warehouse management tool. The scope of this application is narrowed down to model the stock movements between different locations within the system. Therefore, it doesn’t model any stock moving out of the system, such as dispatching goods from the delivery van to customers. Listed below are the three processes modelled by the warehouse management application of this project, note that a real-life warehouse management application will provide a lot more functions than this.

- **Pickup** – Picking goods in a customer order for delivery.
- **Put-away** – Moving stock from one location to another location within the system.
- **Stocktaking** – Verify the amount of stock recorded in the database against the actual value.

6.2.2 Use Case Diagram



6.2.3 Pickup

In order to fulfil a customer order, all the products listed in an order must be put onto the delivery van before delivery. The warehouse management application allows a staff to select a customer order and shows the list of items required to fulfil that order. As the staff looks through the list of products, he needs to confirm the product barcode to make sure the correct product was being pickup. He also needs to confirm the source location of the picked item, which updates the database to reflect a true view on the stock level at the source location.

6.2.4 Put-away

As described in Section 5.2.3 of this report, one type of product can be stored in different sites, and a physical site is subdivided into many locations. The “put-away” function models the stock movement between any two locations within the system. When a staff member moves a type of stock from one location to another, the warehouse management application helps him to record the transaction in the database. It also updates the stock level at the source location and the destination.

The movement of stock can occur between different locations on the same site [See example 1 below], or it can occur between two locations on two different sites [See example 2 below].

- Example 1 – Moving 10 cans of baked beans from Room_#1 of Warehouse_ABC to Room_#2 of Warehouse_ABC.

- Example 2 – Moving 5 bottles of wine from the Dispatching_Bay_#1 of Warehouse_ABC into Shelf_#3 of Vehicle_XYZ.

From the above examples, it is obvious the put-away process is essentially the basis of any stock movements occurred in the system. In fact, it is true that the “pickup” process described previously is a division of the “put-away” process. However, from users’ perspective, the screen designed for the two processes are quite different, also, the two processes are functionally different and can not be combined into one.

6.2.5 Stocktaking

Stocktaking is a necessary process for warehouse management. This process is carried out in order to validate the stock amount recorded in the system against the actual number of stock available in the warehouse. The variance between the actual value and the recorded value usually arises because some goods were stolen or damaged, this variance must be recorded in the database in order to reflect the actual stock level in the system.

The warehouse management tool aims to provide a convenience way for stocktaking, it tells the staff the expected amount for a product stored at a particular location. It gives the facility to record the actual amount and calculates the variance when appropriate.

6.2.6 Functional Requirements

The list below shows a minimal set of functional requirements needed from the warehouse management application.

- Retrieves customer orders, ordered items and product info from the SQL CE database.
- Retrieves site info and location info from the SQL CE database.
- Provides the facilities to pickup items listed in a customer order.
- Provides the facilities to record stock moving between locations within the system.
- Provides the facilities to perform stocktaking.
- Save updated info in the SQL CE database on the Pocket PC.

6.2.7 Non-functional Requirements

Listed below are the non-functional requirements for the warehouse management application.

- Easy and speedy to use.
- User interfaces are clearly displayed within the limited size screen.
- Able to run on a Pocket PC with very limited amount of memory.

6.2.8 Assumptions

- The Pocket PC used has barcode reading capability.

6.2.9 Tasks

There are a number of tasks associated with this part of the project.

- Prepare a design specification for the warehouse management tool.
- Prepare a scope document for the warehouse management tool.
- Prepare a MVC specification for the warehouse management tool.
- Integrate the warehouse management tool on top of the database used in part 1.

6.3 Design

The second phase of the development lifecycle is the design phase. It involves the design of application flow, the layout of user interfaces and how data are stored in the database.

6.3.1 Application Flow Design

The flow design of the warehouse management tool is described by Figure 6.3.1. The features of each screen are briefly described in the diagram, and they will be described in details in Section 6.3.2.

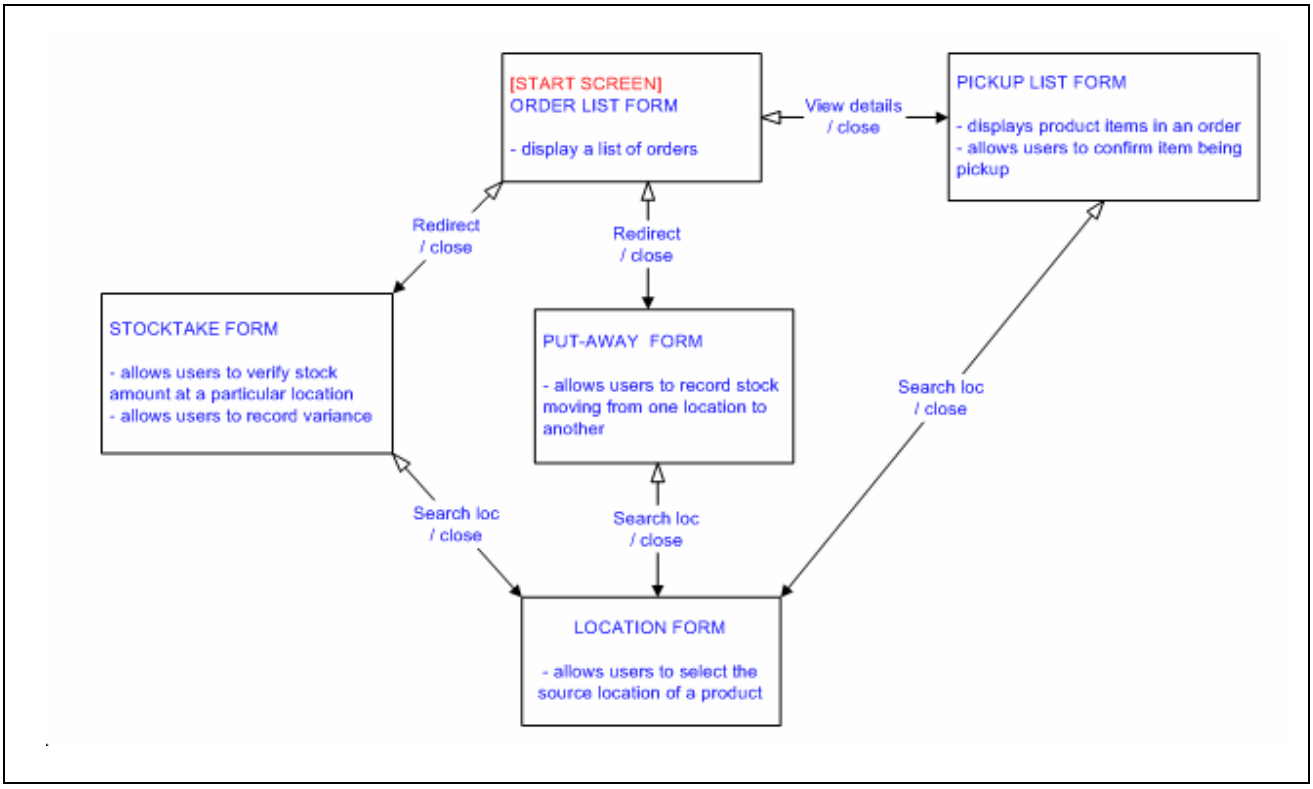


Figure 6.3.1 – Flow diagram of the warehouse management tool.

6.3.2 User Interface Design

Once again, the user interface design is a very important task in this project. This section describes the graphical user interface of the Warehouse Management Tool, and how each screen provides the required functionalities to the users.

This section shows the main screens of the inventory management tool. For other screens of the application, please refer to Appendix **I** at the end of this report.

6.3.2.1 Order List Screen

- The order list screen is the start screen of the warehouse management tool.
- The order list initialised with all the customer orders stored in the local SQL CE database.
- Each order item in the list gives info about the order ID, name of the client company, customer name, the order date and total price of the order.
- There is a splitter control at the top of the order list, it is used to adjust the column width of the list view.
- The user can redirect to the “Pickup Form” by clicking “Pickup” option under the context menu or by selecting the “Pickup” option under the “Action” main menu. The “View” option provides the same functionality, but it will redirect to the “View” tab in the “Pickup form”.

• Under the “Action” Menu, the “Close” option simply closes the warehouse management tool. The “Putaway” and “Stocktake” options will redirect the user to the “Putaway Screen” and the “Stocktake Screen” respectively. • Note that the “Task List Control” mentioned in Section 5.4.2.2 is used again in order to provide a multi-line list view.	

6.3.2.2 Pickup Screen

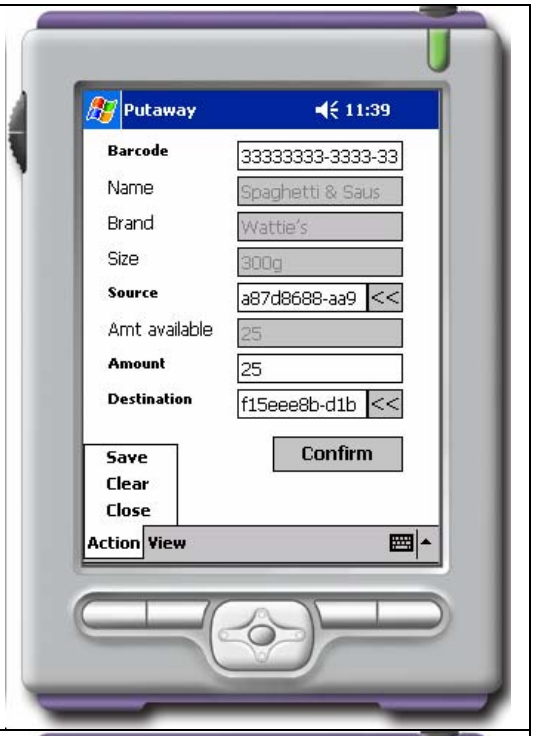
• The “Pickup” screen is shown when the user clicked the “Pickup” option in the “Order List Form”. • The purpose of the “Pickup Form” is to allow users to look through each item in a customer order. It requires the user to confirm the barcode, the source location of the product, and the amount of stock to be pickup to fulfil the customer order. • An item number is shown at the top indicating the total number of ordered items in the current customer order. • The product ID, product name, brand name, product size and the ordered amount is shown as labels, indicating the product to be confirmed. • There are three input fields on the form, they are used to confirm the barcode, the location ID and the	

amount of stock to be pickup.

- The user is expected to use a Pocket PC with barcode reading ability, so he can scan the product barcode very easily.
- For the input of the location ID, the user can either scan the location ID with a barcode reader, or alternatively, he can click on the [<<] button beside the location field. This will redirect the user to the "Location Form".
- The "amount" input field sets the amount of stock to be picked from the source location.
- The "Next" button shows info of the next ordered item in the customer order.
- When the "Confirm" button is clicked, it flags that the current product is being picked. It will update the list of confirmed items shown in the "View" tab.
- When the "View" menu is clicked, all items belong to the customer order will be displayed [See the figure on the right]. It highlights which items are picked and saved in the database already [highlighted with green colour]. Which items are confirmed but not yet saved [highlighted with blue colour], and which items are not yet confirmed.
- In the "View" tab, the user can choose to display a selected product, or he can decide to reset a previously confirmed item.
- As described in Section 6.2.3, the "Pickup" process models the stock movement from the source location onto a delivery vehicle. Therefore, the user must be able to select a delivery vehicle as the target location.
- When an item is confirmed, it is not yet saved in the database, the user must select the "Save" option under the "Action" menu in order to update the database.

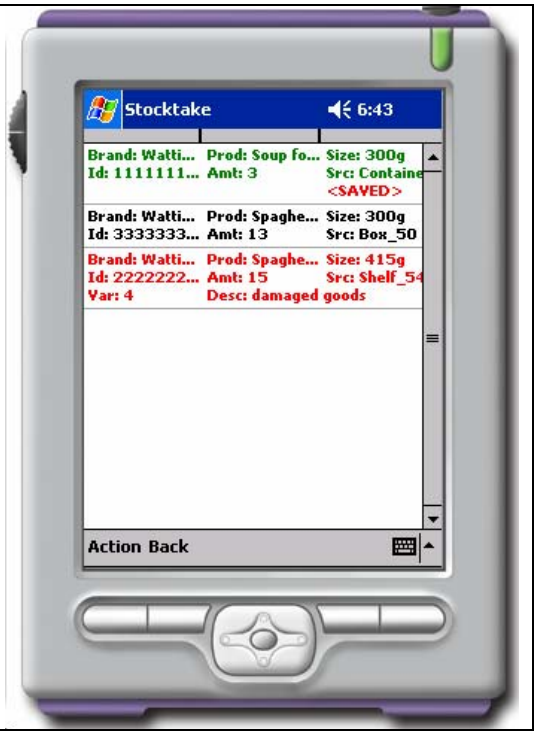
6.3.2.3 Put-away Screen

- The “Putaway Form” is shown when the user selects the “Putaway” option in the “Action” menu of the “Order List Form”.
- The purpose of the “Putaway Form” is to record stock movement from one location to another location within the system.
- The design of the “Putaway Form” is similar to the “Pickup Form” mentioned in Section 6.3.2.2.
- All the input fields in the form are cleared when the form starts.
- When the barcode is set by the user [Using a barcode reader], the product info will be retrieved from the database and displayed on screen automatically.
- The user must set the source location ID and the target location ID in order to record the stock movement. It can be done using a barcode reader, or by selecting a location via the “Location Form”.
- There is a “View” tab associated with the “Putaway Form”, it displays all the stock-move entries recorded by the user. Also, it highlighted which transactions are saved in the database already.
- When an item is confirmed, it is not yet saved in the database. The user must select the “Save” option under the “Action” menu in order to update the database. If the user decided to close the form, the application will prompt for saving before exiting the application.



6.3.2.4 Stock-take Screen

- The “Stocktake Form” is shown when the user selects the “Stocktake” option under the “Action” menu of the “Order List Form”.
- The design of the “Stocktake Form” is similar to the “Putaway Form” mentioned in Section 6.3.2.3.
- All the input fields in the form are cleared when the form starts.
- When the barcode is set by the user [using a barcode reader], the product info will be retrieved from the database and displayed on screen automatically.
- The user must set the location ID of the product, it can be done using a barcode reader, or by selecting a location from the “Location Form”.
- When the location ID is set, the form returns the expected amount of stock at the selected location.
- The user enters the actual quantity in the “amount” input field.
- When the user presses the “Confirm” button, the application checks whether the expected amount recorded in the system equals the actual amount entered by the user.
- If the two values are the same, then this entry will be recorded in a list of confirmed items.
- If the two values are different, then the user will be redirect to another screen, in which the user is able to enter a reason for the variance. Once the reason is confirmed, this entry will be added to the list of confirmed items.
- There is a “View” tab associated with the “Stocktake Form”. It displays all the stocktaking entries confirmed by the user. It highlights which of these entries are saved in the database, which entries are not yet saved, and those entries where a variance exists.

• When an item is confirmed, it is not yet saved in the database. The user must select the “Save” option under the “Action” menu in order to update the database. If the user decided to close the form, the application will prompt for saving before exiting the application.	

6.3.2.5 Location Screen

• The “Location Form” is used by each of the three forms in the warehouse management application. It allows users to select the location ID when moving a stock from one location to another. • There are two filters in the location form, they are the “Region” filter, the “Site” filter. • When the “Location Form” starts, the application seeks for all locations that contain the required product. Then the “Region” filter and the “Site” filter are filled appropriately. • The user can find out the required location by changing the filters. Descriptions about the location are updated when the user changes the selected item in the location box. • When the “Confirm” button is pressed, the selected location ID will be returned to the requesting form. • The “Back” button is used to return to the parent form without setting the location ID.	

6.3.3 Database Design

In addition to those data table added in Section 5.3.3, the Warehouse Management Tool requires additional tables to capture stock movement and the stocktaking operations.

These tables have been added for tracking reasons. Whenever a stock is moving from one location to another, an entry must be made in the database to verify the transaction. Also, the system must record the person who is responsible for the transaction. In case of stocktaking, the variance must be recorded in the database, so that managers can analyse the reasons for stock variance. Figure 6.3.3 shows the tables used in the warehouse management tool.

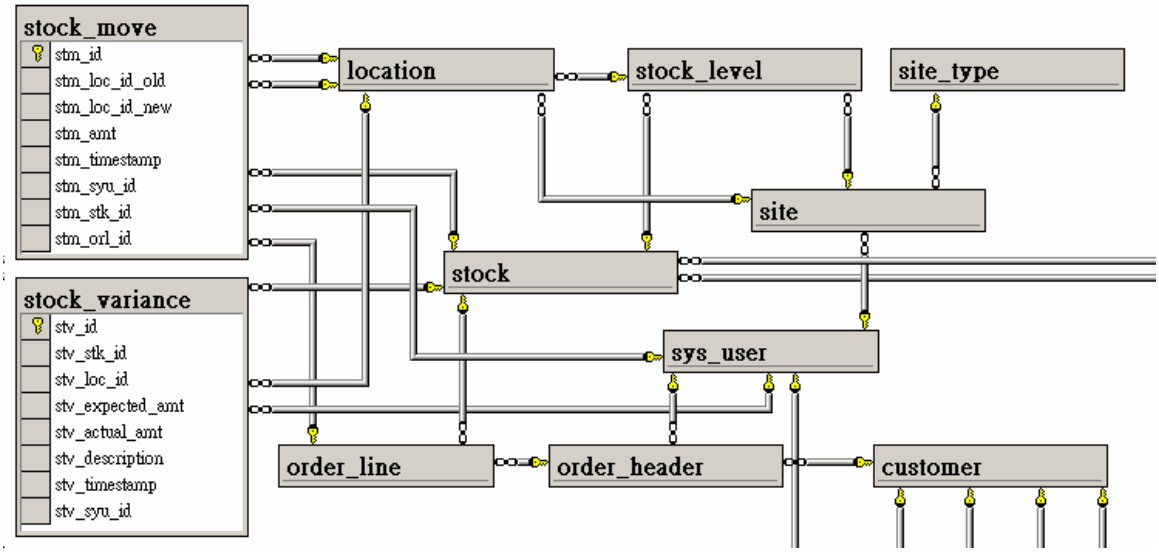


Figure 6.3.3 – Data tables used by the warehouse management tool.

Two additional tables are added to the database, they are the “stock_move” table and the “stock_variance” table. Each of these tables is described below.

Table	stock_move
Descriptions	The purpose of this table is to record any stock movement generated by the “Pickup Form” and the “Putaway Form”. Each entry records the source location, destination, the amount of stock being moved, the person who is responsible for the transaction and a timestamp of the transaction.

- stm_id – the primary key of the stock_move table. Identifies a particular entry in the stock_move table.
- stm_loc_id_old – a foreign key to the location table. Indicates the original location of the product before the movement.
- stm_loc_id_new – a foreign key to the location table. Indicates the new location of the product after the movement.
- stm_amt – the amount of stock moved.
- stm_timestamp – a timestamp to the transaction.
- stm_syu_id – a foreign key to the system user table. Indicates who is responsible for the transaction.
- stm_stk_id – a foreign key to the stock table. Indicates the stock being moved.
- stm_orl_id – a foreign key to the order line table. Indicates which order the transaction is associated with.

Table	stock_variance
Descriptions	The purpose of this table is to record the variance when stocktaking is done. This table is used by the “Stocktake Form”. Each entry in the table record the product being verify, the expected stock amount recorded in the system, and the actual stock amount counted by the user. In case a variance exists between the two values, a description of the difference is also recorded. This table also tracks the transaction time and who is responsible for the transaction.

- stv_id – the primary key of the stock_variance table. Identifies a particular entry in the stock_variance table.
- stv_stk_id – a foreign to the stock table. Indicates the product involved in stocktaking.
- stv_loc_id – a foreign key to the location table. Indicates the location of the product.
- stv_expected_amt – the amount recorded the system.
- stv_actual_amt – the actual amount counted by the user.
- stv_description – a reason for the variance.
- stv_timestamp – a timestamp to the transaction.
- stv_syu_id – a foreign key to the system user table. Indicates who is responsible for the transaction.

6.4 Implementation

This section describes how the Warehouse Management Tool was implemented. It describes the reasons for using the selected approach, the components used in the implementation, and some of the classes written to perform the operations.

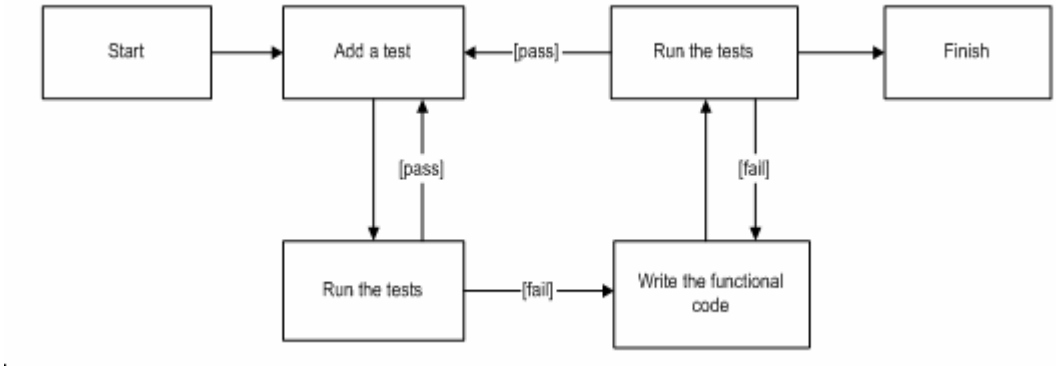
6.4.1 Approaches

From the experience gathered during the development of the Inventory Management Tool, it shows that the traditional programming approach is only suitable when the business logic of the application is simple. For the Warehouse Management Tool, the logics involved are a lot more complex than the Inventory Management Tool. It involves retrieving and updating the database, and it involves a lot of error checking on the user inputs. Furthermore, testing the application manually doesn't seem to be efficient. Therefore, alternative approaches are chosen to implement the Warehouse Management Application.

By doing researches, the "Test Driven Development" methodology and the "MVC paradigm" seem to be a good choice to implement the Warehouse Management Application. Each of these approaches will be explained in the following sections.

6.4.2 Test Driven Development (TDD)

Test-driven development (TDD) attempts to produce high quality, well-tested code by writing the test cases before writing the actual code. With traditional testing, a problem in the code is identified if a successful test case fails, the developer fixes the code to fulfil the test case and run the test cases again. With TDD, the developer starts by writing a test unit to verify a specific function in the application, if the test unit passes, then the developer continues to write another test case. But if the test unit fails, then the developer needs to write the code to fulfil the function. This process is summarized by the figure below.



- The developer starts by writing test cases to test the current prototype.
- If the test cases passed, then the developer continues to write more test cases to test the functionalities of the application.
- If a test case fails, then the developer add in the code to fix the problem, and continues to fix the code until all the test cases pass.

Note that TDD does not replace traditional testing. Instead, it defines a proven way to ensure effective unit testing. Furthermore, the test cases used in TDD are working examples for invoking the code, which proofs the code functions in a way the developer expected it to.

6.4.3 MVC paradigm

"MVC" stands for Model-View-Controller, there are the three types of object used in the MVC paradigm. The idea of MVC model is to separate the graphical interface from the business logic of the application. The View object manages the graphical interface of the application. The Controller object wholes all the business logic of the application. And the Model object manages the behaviour and data of the application.

In the Warehouse Management Application, each of the forms designed in Section 6.3.2 are separated into the three classes mentioned above. For example, the "Pickup Form" is implemented using three different classes, with the "PickupForm" class representing the View object. The "PickupFormController" class representing the Controller object. And the "PickupFormModel" class representing the Model object. When the user interacts with the View object, the event handlers in the View object simply calls a method in the Controller class, and the method inside the Controller class will do the required functions. The Controller class will make any necessary updates to the data stored in the Model class and notified any subscribers for the changes. The View class subscribes to the events inside the Controller class through the use of delegates. Therefore, the View class will be updated wherever the Controller signals a changes.

The use of MVC paradigm facilitates Test-Driven Development, because it allows developers to test the business logic of an application without worrying the graphical interface.

6.4.4 Components

The components used to implement the warehouse management tool are exactly the same as the ones described in Section 5.4.2. The purpose for each of these components is very similar to the inventory management tool, and therefore will not be repeated here.

To see the descriptions on how these components are used in the warehouse management tool, please refer to the design specification on Section 6.3.2.

6.4.5 Code Samples

6.4.5.1 Code Sample I – MVC Demo

Descriptions	(1) As described in Section 6.4.3. MVC coding technique is used to implement the warehouse management application. Three classes are shown below, they are the “LocationFormModel” class, the “LocationForm” and the “LocationFormController” class, and these classes represent the “Model-View-Controller” paradigm respectively. (2) The “LocationFormModel” class acts as a storage place for all the variables used in location screen. (3) The “LocationForm” class presents the graphical interface of the location screen. It contains handlers to captures system events, and it subscribes to the events in the “LocationFormController” class. (4) The “LocationFormController” class holds the business logic of the location form. Basically, all actions and computations are handled by this class. (5) The code samples shown are the trimmed version of the actual classes, most code related to business logic and error checking has been truncated. These code samples aim to illustrate how to work with MVC coding. (6) Again, pseudo codes are highlighted in brown.

```
internal class LocationFormModel
{
  private int _selectedLocationIndex=-1, _selectedSiteIndex=-1;   //NOTE (1)
  private ArrayList _locationList=new ArrayList(), _siteList=new ArrayList();
  protected internal LocationFormModel() {}
  protected internal int SelectedIndex { get{...}  set{...} }    //NOTE (2)
  protected internal int SelectedSiteIndex { get{...}  set{...} }
  protected internal ArrayList LocationList { get{...}  set{...} }
  protected internal ArrayList SiteList { get{...}  set{...} }
}
```

```
class LocationForm : System.Windows.Forms.Form
{
  private LocationFormController _controller;   //NOTE (3)
  private void InitController(string stockId)
  {
    _controller=new LocationFormController();   //NOTE (4)
    _controller.OnSiteDataLoaded += new VoidEventHandler(this.DoSiteDataLoaded); //NOTE (5)
    _controller.OnLocationDataLoaded += new VoidEventHandler(this.DoLocationDataLoaded);
    _controller.OnFormConfirmed += new StringIntEventHandler(this.DoFormConfirmed);
    _controller.InitController(stockId);
  }
  private void cbLocationName_SelectedIndexChanged(object sender,System.EventArgs e)//NOTE (6)
  {
    _controller.PerformSetSelectedIndex(the_selected_index);
  }
  private void btnConfirm_Click(object sender, System.EventArgs e){...}
  private void btnBack_Click(object sender, System.EventArgs e)  //NOTE (7)
  {
    //DO: Closes the form and return to the pervious form
  }
  protected internal void DoSiteDataLoaded(){   //NOTE (8)
    //DO: fill the combo box using the list of sites.
  }
  protected internal void DoLocationDataLoaded() {}
  protected internal void DoFormConfirmed() {}
}
```

```
class LocationFormController
{
  private LocationFormModel _model=new LocationFormModel();   //NOTE (9)

  public event VoidEventHandler OnSiteDataLoaded;   //NOTE (10)
  public event VoidEventHandler OnLocationDataLoaded;
  public event StringIntEventHandler OnFormConfirmed;

  protected internal void PerformLoadLocationData(string siteId,
     string stockLevelId) //NOTE (11)
  {
    _model.LocationList=LoadLocationData(siteId, stockLevelId);
    NotifyLocationDataLoaded();  //NOTE (12)
  }
  protected internal void PerformLoadSiteData(string stockId) {...}
  protected internal void PerformSetSelectedLocationIndex(int selectedIndex) {...}
  protected internal void PerformSetSelectedSiteIndex(int selectedIndex) {...}
  protected internal void PerformConfirmForm() {...}

  protected internal void NotifyLocationDataLoaded()   //NOTE (13)
  {
    if (OnLocationDataLoaded != null)
      OnLocationDataLoaded();
  }
  protected internal void NotifySiteDataLoaded() {...}
  protected internal void NotifyFormConfirmed() {...}
}
```

Notes	
	(1) The Model class contains variables that are used across the 3 MVC classes. (2) The Model class provides getter and setter properties to local variables. (3) The View class contains a pointer to the Controller class. (4) When the View class is initiated, the Controller class is created. The View class then subscribe to various event handlers provided in by Controller. (5) For example, when the Controller raises an “OnSiteDataLoaded” event, then the “DoSiteDataLoaded” method in the View class will be called. (6) When a system event occurs, for example, when the “SelectedIndexChanged” event is fired, the View class doesn’t do any actual processing or computation, it simply calls the corresponding method provided by the Controller class. (7) For those events which don’t implicate the business logic, the View class will handle it without calling the Controller class. (8) A method that subscribes to an event in the Controller class will perform its actions when the event is fired. (9) When the Controller class is initiated, it creates a Model object for data storage. (10) The Controller class has various events in which the View class can subscribe to. (11) The actual business logic and computation is handled in the Controller class. In the example, the location list in the Model will be filled by data retrieved from the SQL CE database. (12) When the Controller finishes the required operations, it calls the notify method to signal the changes. (13) If there is a subscriber to an event fired in the Controller class, the corresponding method in the View class will be called to update the graphical interface.

6.4.5.2 Code Sample II - Transaction

Descriptions	(1) The code below shows the "PerformUpdateDatabase" method used in the "PutawayFormController" class. As described in Section 6.3.2.3, an entry is kept on any stock moving between locations within the system. The purpose of this method is to update the database when the user decides to "Save" these entries into the SQL CE database. (2) This method illustrates an important issue when updating a database, which is the concept of "transaction". The "transaction" concept means that an update to the database will be commit only if the database remains "consistence" during and after the update. If an error occurs during several related updates, then the database must be able to rollback to a consistence state. (3) In order to maintain data consistency, a "SqlTransactionObject" is written. A "SqlTransactionObject" contains methods to perform query and updates to the database. This object allows a group of SQL commands to be executed before committing the transaction. If any one of the SQL commands in the group gives rise to an error, then the "SqlTransactionObject" will raise an exception, which is caught by the application, and rolls back the pervious updates. (4) A lot of code in the sample is converted into readable pseudo code (highlighted in brown), this is because the original code is very complex to understand without looking at other code in the same file. (5) Similar methods are written inside the "PickupFormController" class and the "StocktakeFormController" class, which are used to update the database as well.

```
protected internal void PerformUpdateDatabase()
{
  ArrayList list=_model.ConfirmedItems;   //NOTE (1)
  ArrayList problemItems=new ArrayList();

  for(int i=0; i<list.Count; i++)
  {
    PutawayObject o=(PutawayObject)list[i];   //NOTE (2)
    if(o.IsUpdated)
    {
      LocationObject source=(LocationObject)o.SourceLocation;
      LocationObject target=(LocationObject)o.TargetLocation;

      SqlTransactionObject sql=new SqlTransactionObject();   //NOTE (3)
      bool noError=true;

      try
      {
        Cursor.Current = Cursors.WaitCursor;  //NOTE (4)
        Cursor.Show();

        //CASE 1: IF (the stock move between 2 locations in the same physical site)//NOTE (5)
        {
          //IF (the source location has more stock than required)
            //UPDATE: deducts the stock amount on the source location

          //ELSE IF (all the stock in the source location are moved)
            //UPDATE: deducts the stock amount on the source location
            //UPDATE: removes the link between the location entry and the stock level entry

          //DO: updates the amt at target location
        }
```

```
        //CASE 2: IF (the stock moves to a different site)
        {
          //IF (the source location has more stock than required)
            //UPDATE: deducts the stock amount on the source location

          //ELSE IF (all the stock in the source location are moved)
            //UPDATE: deducts the stock amount on the source location

          //DO: decreases the amount attribute in the stock_level entry because stock is
                moving to another site

          //IF (the stock amount on the stock_level entry is 0)
            //UPDATE: deletes the entry from stock_level table


          //IF (the target location already got an associated stock_level
                entry on the product)
            //UPDATE: updates the amount at target location
            //UPDATE: updates the stock amount in stock_level table

          //ELSE IF: (the target is not related to any stock_level entry)
            //INSERT: creates a new stock_level entry for the target site.
        }

        //DO: add an entry in the stock_move table to record the transaction
      }
      catch(Exception exception)  //NOTE (6)
      {
        problemItems.Add(o);
        sql.Rollback();
        o.IsUpdated=true;
        noError=false;
      }
      finally
      {
        Cursor.Current = Cursors.Default;
        Cursor.Show();
      }

      if(noError)   //NOTE (7)
      {
        sql.Commit();
        o.IsUpdated=false;
        o.IsConfirmed=true;
      }
    }
  }
  _model.ProblemItems=problemItems;  //NOTE (8)
  NotifyDatabaseUpdated();
}
```

Notes	(1) Two array lists are used, the “list” array contains all the confirmed items stored in the “PutawayFormModel” class. The “problemItems” array is used to store any incorrect updates (2) Each item in the confirmed list is checked, if the item is newly updated, then it will be saved in the database. (3) A “SqlTransactionObject” is created to maintain a transaction. (4) Since the SQL updates always induces a waiting period to the user, therefore, the cursor is changed to a “wait cursor” when the database is

	being updated. (5) The actual update to the database depends on the amount of stock being relocated, and it also depends on the source and destination of the stock. (6) If an exception arises, the updates related to the current stock-move entry are being rollback. The error entry will be kept in the "problemItems" array. (7) The transaction will be stored in the database only if all the necessary SQL commands are correctly issued and passed. (8) The list of error items will be stored in the model class. The "PutawayForm" will be notified once the database is updated.

6.5 Testing

The use of Test-Driven Development makes it easier to test the application. Instead of testing the whole application after the application is done, testing is done as the application is developed.

In additional to the application testing described in Section 5.5. Testing for the Warehouse Management Tool is also done via a NUnit module. NUnit is a tool used to test a .NET class.

Figure 6.5 shows the NUnit module used to test each function in the Controller class of the "Pickup Form". With the help of MVC paradigm, testing can be done on the business logics without worrying the graphical interface. Using NUnit, the test module displays an error message whenever a test case fails. The example shows that the "Amount" field in the Model class is not correctly set, and the database connection can not be establish when the application tries to update the database.

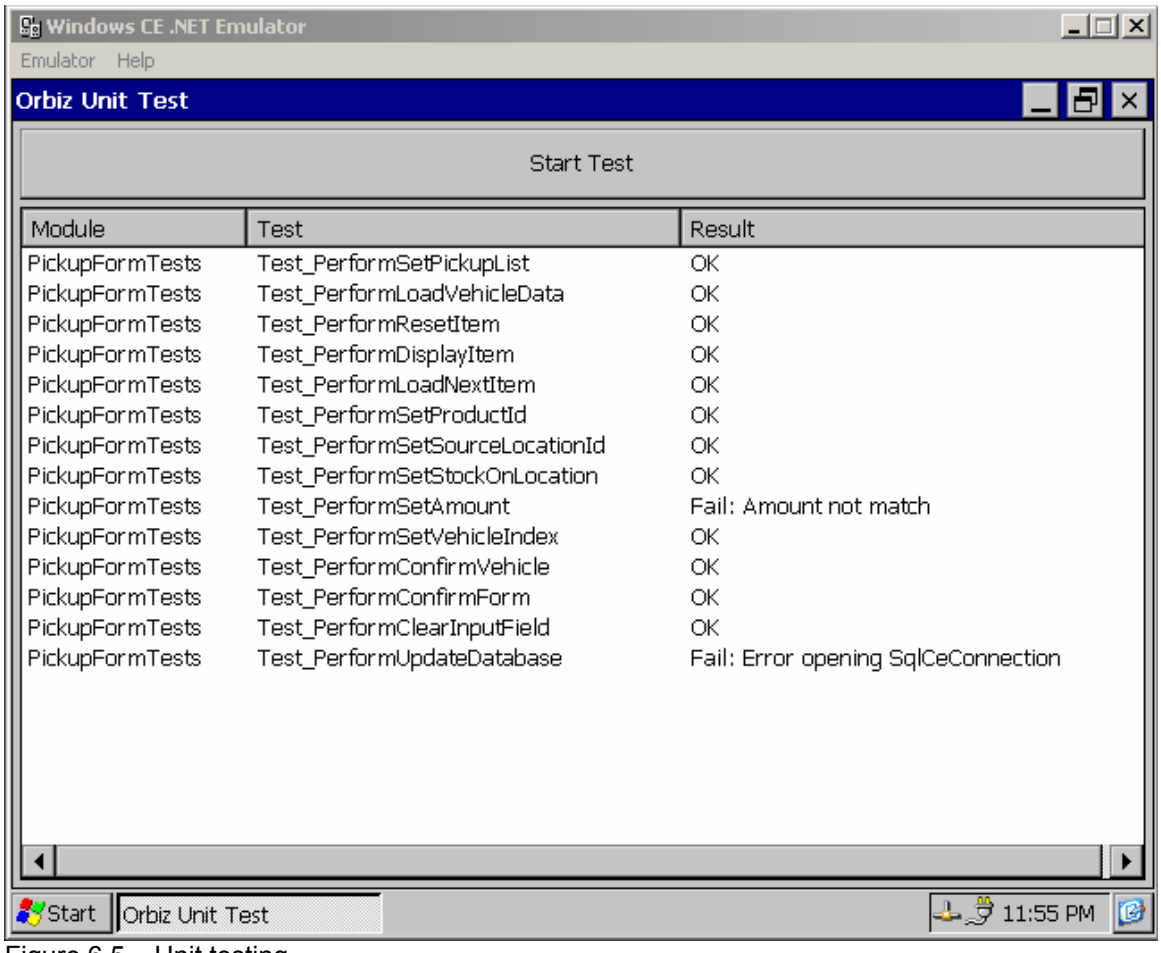


Figure 6.5 – Unit testing

6.5.1 Test Case Example

Descriptions	(1) When the “PerformLoadVehicleData” method inside the “PickupFormController” class is called, it loads all the vehicle data from the database, and it stores the result into the “PickupFormModel” class. (2) The “Test_PerformLoadVehicleData” method shown below is a test case in the NUnit model. It is used to verify the “PerformLoadVehicleData” method. (3) Testing is done by comparing the expected result with the actual result, the test case will raise an exception if the results are different.

```
public void Test_PerformLoadVehicleData()
{
  //PRODUCE THE EXPECTED RESULT
  DataSet ds=new DataSet("DS");  //NOTE (1)
  ds.ReadXml(@"\VehicleData.xml");
  DataView dv=ds.Tables[0].DefaultView;
  Hashtable expected=new Hashtable();
  for(int i=0; i<dv.Count; i++)
  {
    string id=dv[i]["id"].ToString();
    string name=dv[i]["name"].ToString();
    expected.Add(id, name);    //NOTE (2)
  }

  //PRODUCE THE ACTUAL RESULT
  PickupFormController _controller=new PickupFormController();  //NOTE (3)
  _controller.PerformLoadVehicleData();
  ArrayList actual=_controller.Model.Vehicles;   //NOTE (4)

  //COMPARE THE ACTUAL RESULT WITH THE EXPECTED RESULT
  TestCondition.Assert(expected.Count == actual.Count, "Fail: num of items not match.");
  for(int i=0; i<actual.Count; i++)
  {
    VehicleObject vo=(VehicleObject)actual[i];  //NOTE (5)
    if(expected[vo.Id]==null)
      TestCondition.Assert(false, "Fail: vehicle id not found");
    if(expected[vo.Id]!=vo.Name)
      TestCondition.Assert(false, "Fail: invalid vehicle data");
  }
}
```

Notes	(1) The expected result is produced by reading a predefined dataset from an XML file. (2) The expected result is stored into a Hashtable for further comparison. (3) The actual result is produced by actually calling the “PerformLoadVehicleData” method of the Controller object. (4) If no errors occur, the vehicle data is expected to be storage inside the Model object. (5) The actual dataset is compared to the expected result. If they don’t match each other, an exception will be raise, and caught by the NUnit module.

6.6 Deployment

The deployment phase is exactly the same as described in Section 5.6. Since both the Inventory Management Tool and the Warehouse Management Tool are combined into one project. Therefore, the cab files generated by Visual Studio .NET actually include the whole Mobile Inventory Management System application. [See Appendix I for screenshots.]

6.7 Section Summary

- The development of the warehouse management tool is divided into five phases. They are the requirement analysis phase, design phase, implementation phase, testing and deployment.
- The primary function of the warehouse management tool is to provide a convenient way to record stock movements. A record is kept in the database when a product changes its location within the system. The second objective of the warehouse management tool is to facilitate stocktaking.
- The warehouse management tool is a mini version of a real-life warehouse management tool, it only models the “Pickup” process, the “Put-away” process and the “Stocktaking” process.
- The “Pickup” process involves picking up stock to fulfil a customer order. The “Put-away” process involves moving stock from one location to another. The “Stocktaking” process involves validating the stock amount at a particular location. Each process is associated with a user form, which provides the graphical interface to capture user actions.
- The design phase involves the design of user interfaces, the design of application flow and database design.
- Two additional tables are added to the SQL Server database, they are needed to capture stock movements and the stocktaking process.
- The “Test Driven Development” methodology and the “MVC paradigm” are used to implement the warehouse management application.
- The aim of TDD is to produce a well tested application. It achieves this by writing test cases before writing the actual functional code. Instead of testing the whole application at the end of the development, testing is done on each function of the application as the application is developed. Using TDD, a set of test cases can be run over and over again, it speeds up the testing process.
- The idea of MVC model is to separate the graphical interface from the business logic of the application. The View object manages the graphical interface of the application. The Controller object wholes all the business logic of the application. The Model object manages the behaviour and data of the application.
- Each form used in the warehouse management application is implemented using the Model-View-Controller approach.
- The main advantage of using MVC model is its reusability. Since all the business logics are implemented inside the Controller class and all the data are stored in the Model class, that means changing the graphical interface of the application only involves coding the View class again. Both the Controller class and the Model class can be reuse without making major changes to the code.
- The main disadvantage of using TDD and MVC model is the complexity of the code. It requires a lot of effect to separate the business logic completely from the graphical interface. It requires a lot of time writing test cases to test every aspect of the business logic.

- Custom controls are written to provide specific functions to the user. The “Task List Control” and the “Column Header Control” are used again to implement the warehouse management tool.
- The method used to retrieve data from the SQL CE database is similar to the way used in the inventory management tool.
- Updating the database involves building up SQL statements from user inputs, and then executes the “ExecuteNonQuery” command from a SqlCeCommand object. When a group of related SQL statements are executed, a transaction is needed to make sure the database has been updated correctly. If an exception happens during a transaction, all the related updates must be roll back as well, this ensures the database is consistent after the update.
- Testing is a very important phase in the development lifecycle. Testing for the warehouse management tool is done using NUnit. With the help of the MVC model, each part of the application can be tested using predefined code. The idea is to write test cases that simulate a user action, and then compares the actual result with the expected result. However, manual testing on the graphical interface is still needed to ensure the usability of the application.
- The deployment of the warehouse management tool is simple, it involves building the installation cab file using Visual Studio .NET, and run it on the Pocket PC. The underlying Windows CE operating system on the Pocket PC must have support for .NET Compact Framework.

7. Discussions / Challenges

- The use of Microsoft SQL Server 2000 and the Microsoft SQL CE database is a very good choice for this project. SQL Server 2000 is a very powerful database system that handles a lot of problems related to synchronization and confliction of updates. For example, when two independent users updated the stock amount of a product, each user thinks that the data stored on the local device shows the most recent view of the database. After each user synchronizes the local SQL CE database with the main Server, the main server will check for confliction between the updates. If conflicts occur, SQL Server will try to resolve the conflicts using user predefined rules, such as priority-based conflict resolution or custom defined confliction resolution. SQL Server will keep an entry in a conflict table to denote the conflict.
- In order to test the mobile inventory management application, a dummy database is created using SQL scripts. As mentioned in Section 4.1.1, Pocket PCs are limited in storage space, although it is usual to have external memory card plugged onto the device to increase the storage, but this increase is considerably small compared to a hard disk used in desktop computers. For the dummy database, it took about 300Mb of disk space in the main server, it is usually not feasible and uneconomical to store a 300Mb data file on the Pocket PC. But surprisingly, when the SQL CE database synchronizes with the main server, the data file produced on the Pocket PC is only about 2Mb. This significant decrease probably means that SQL Server used a lot of disk space to support replication, or it might mean that SQL Server wasted a lot of space to whole indices and other unnecessary functions. Further investigation shows that a lot of functions are not propagated to the SQL CE database. For examples, stored procedures, views, use-defined functions and triggers are not propagated to a SQL CE subscriber when replication occurs. Moreover, when configuring publication in SQL Server, the user can choose which tables are replicated to the subscribers.
- With the current setup, the mobile inventory management application retrieves data from the local SQL CE data file, which is synchronized with a SQL Server database resides in the main server. An alternative method for data retrieval is to use the combination of Web Service and XML files. The mobile inventory management application can alternatively retrieve data from a Web Service, and store the data as XML files on the local device. Since a Web Service is a common standard, therefore the application doesn't need to worry about the type of database used in the backend. This approach might be a better choice for those companies that don't want to use Microsoft SQL Server at their backend. While SQL CE database doesn't support compression natively, the use of XML files allows developers to add on other functionalities such as compression and encryption. However, the cost of using this approach is that the developer needs to handle all those complex confliction problems. In this project, the warehouse management tool is not suitable to use the Web Service technique mentioned above, this is because the updated database on the Pocket PC needs to be synchronized with the main server. But for the inventory management tool, since data only goes from the main server onto the Pocket PC, and the application never updates the data, therefore the Web Service technique might be feasible.
- The programming approach used in the inventory management tool is different to the approach used in the warehouse management tool. The simple, traditional programming approach is used when business logics of the application are simple. There is no separation between business logics and the graphical interface, which means all the functions are hard coded into the form class. An application developed using the traditional programming approach usually requires less time to develop. The test driven approach is facilitated by the MVC paradigm, it requires longer development time and carefully planned test units. However, it is quite good when the quality of the application is important, because testing is done whenever a new function is added to the application. Moreover, the separation between Models, Views and Controllers classes makes it very easy to change the program, and it is especially good if multiple versions are needed for the same application. With the help of MVC model, migration of the application to another platform is relatively easy. It mainly involves changing the View class to suit the new platform, and most business logic in behind stays the same.

- When the inventory management tool was first developed, the Pocket PC emulator provided by Visual Studio .NET is used to test the application. The application actually ran very slowly under the emulator, further testing confirmed that the application took a lot of time retrieving data from the SQL CE database. In order to increase the running performance of the application, the idea of caching frequently used data has been considered. On a Pocket PC, there is no such thing call the virtual memory, it's because all programs and data files share the same physical RAM. When a running application requires more memory to operate, the memory allocation unit simply assign more physical RAM to the application. The maximum amount of memory an application can use is limited by the total memory minus the amount of memory allocated to file storage. Initially, the Pocket PC emulator has a total memory of 16Mb, in which approximately 8Mb are engaged by the default Windows CE programs in the emulator. When the inventory management application is deployed and run in emulator, the .NET runtime environment and the data file took about 7Mb of memory, then means the emulator only has about 1Mb of free memory. When caching is done on frequently used data, the inventory management tool seems to run a bit smoother. However, as more and more data are cached into the memory, the Pocket PC starts to complain on the lack of memory. By allocating more memory to the Pocket PC emulator, the lack of memory problem seems to be solved. However, when the application is tested on a real Pocket PC, caching data into memory seems to have no real benefit on the performance. Furthermore, it is very hard to predict how much data the user stored in the database, and there is no way to predict how much memory is available on the Pocket PC running the application. Therefore, the idea of caching data into memory becomes unfavourable.
- In the current project implementation, each site is divided into many locations. Each location has a unique ID and it declares the exact position of a product. This is not an effective way to identify the location of a product, because when the user needs to move a group of product from one location to another location, it is very time consuming to scan the location ID for each individual product. A solution to this is to add a location type that acts as a "container" to a group of locations, hence that the user can quickly redefine the location of a group of products without scanning the location ID for each product.

8. Conclusions

The Mobile Inventory Management System project has been completed. Both the inventory management tool and the warehouse management tool function in the way specified in the initial plan.

This project reveals a number of differences between developing Pocket PC applications and normal desktop applications. These differences are mainly due to the limited screen size and the limited amount of memory available on a Pocket PC. These limitations largely impact the design of user interface, and they affect the way the application was implemented. It also illustrates that techniques such as caching data might not be feasible for Pocket PCs, because the amount of memory available is usually small, so it should not be done without careful planning.

The development lifecycle of this project is divided into five different phases, each phase has its importance to the development of the application. If a mistake is made in one phase, all the following phases will be affected as well. For example, if the database design is not done correctly, then the problem will propagate to the implementation phase, and it will be time consumption and troublesome to fix a design error once a prototype is developed.

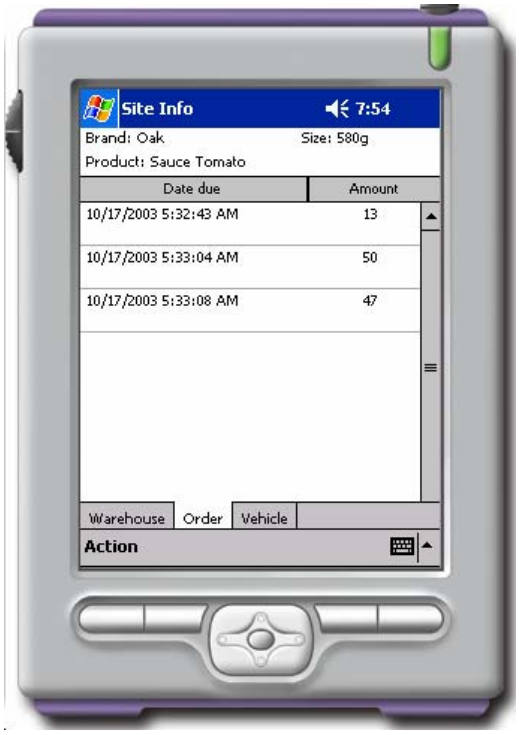
A number of approaches are used during the project, such as the traditional programming approach, the test driven development approach and the MVC paradigm. Each of these approaches has its advantages and disadvantages. No one approach can override another approach completely, the choice as to which approach to use depends on the nature of the application. For example, if the application is simple, and the application doesn't require any future development, then the traditional coding style might be the best approach to use.

From the project, it illustrates that there are many alterative ways to implement an application. For example, the current implementation uses SQL Server and SQL CE database for data retrieval, whereas a Web Service and XML files combination is a feasible solution too. Concerning coding, different methods can be used to achieve the same result. For example, both the DataReader object and the DataSet object can be used to retrieve data from the database. The DataSet object keeps the full dataset in the memory, whereas a DataReader object only keeps a pointer to one data row at a time. Again, the decision to decide which alterative to use heavily depends on the project requirement and the behaviour of the application.

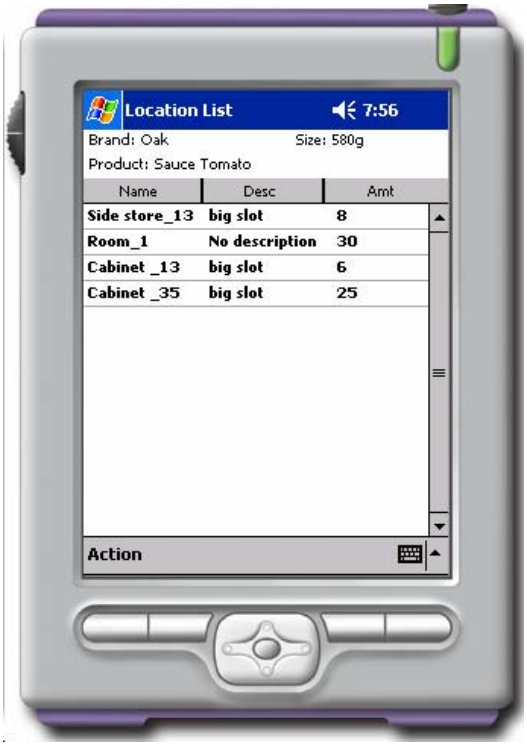
9. Appendix I – Screenshots

1) The “Inventory” button, the “Warehouse” button and the “Synchronize” button will initialize the Inventory Management Tool, the Warehouse Management Tool and the “Synchronization Form” respectively.

2) After synchronization is run, a SQL CE data file is created on the Pocket PC.

3) When the Inventory Management Tool starts, the product list is cleared. The user can display a product by changing the filters.	4) After the filters are set, the product list will be updated.
	
5) To display site info, the user can select the “Site Info” option in the context menu, or selects the “Site Info” under the Action menu.	6) The “Site List Screen” displays all the sites that contain the selected product. The first tab page displays the stock amount in each warehouse.
	
7) The second tab page displays stocks that are on order from suppliers.	8) The third tab page of the site form displays stocks that are on vehicle.

9) The user can see the exact location of the selected stock by opening the “Location List Screen”.



10) The “Location List Screen” lists the exact location where the selected product resides.

11) On the “Inventory List Screen”. The user can select the vehicle tab, which display all stocks reside on the current vehicle.

12) Closing the Inventory Management Tool will return to the opening screen.

13) When the warehouse management tool starts, the "Order List Screen" is displayed. It lists all the customer orders stored in the database.

14) The user can check out all products related to a customer order by selecting the "Pickup" option in the context menu.

15) The "Pickup Screen" shows all the items related to a customer order. It displays info about the requested stock.

16) When the [<<] button is clicked, the "Location Form" is shown. All the sites that contain the requesting product will be listed in the filters.

17) The user can choose the correct site by changing the filters.

18) After the user selected the correct location, the application displays the amount of stock available on the selected location.

19) When the user confirmed the location, the location ID is set in the location text field.

20) When the user clicks the confirm button, the application checks is there enough stock available on the source location to fulfill the order.

21) Another location is needed if the source location doesn't contain enough stock.

22) When the confirm button is clicked, the application checks whether the input amount equals to the ordered amount.

23) After the item is correctly confirmed, the source location info will be shown in the "Details" page. The "Details" screen is accessible by clicking the "View" menu.

24) Because stocks are expected to be moved on to a delivery vehicle, therefore the application requires the user to selects a target vehicle when the user wants to save the updates.

25) The “Vehicle screen” is displayed, so that users can select the target vehicle.

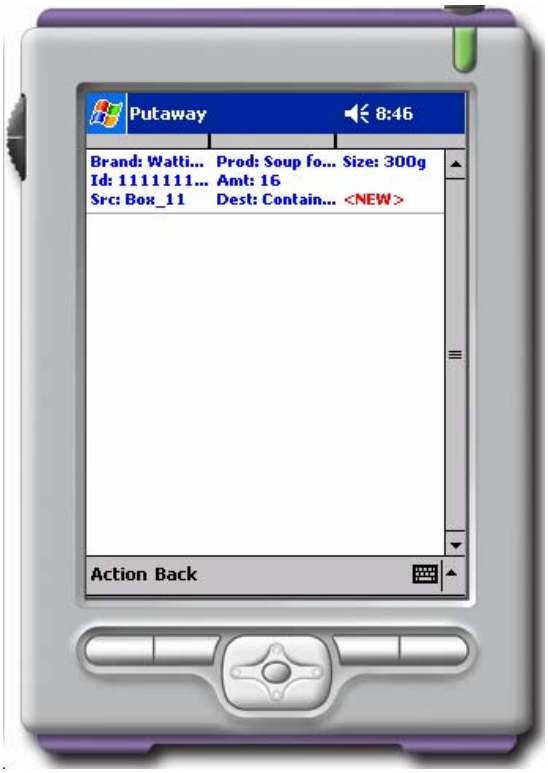
26) When the user clicks the “Save” option under the Action menu, the database will be updated.

27) The “Details” page will show which items are yet to be pickup, and which update has been saved.

28) The “Put-away Screen” is displayed when the user selects “Put-away” under the “Action” menu in the “Order List Form”.

29) When the user scans the barcode of a product onto the barcode text field, info about that product will be displayed.

30) After the user enters the source location ID and destination location ID, he can confirm the form by clicking the “Confirm” button.



31) An entry will be made in the “Details” page. The “Details” page is accessible by clicking “View” menu in the “Put-away” form.

32) By clicking the “Save” option under the “Action” menu, the entry will be saved into the database.

33) If the user selects the same location ID twice before saving it, then an error message will be displayed. This checking is to ensure data integrity of the database.

34) The “Details” page will show any stock movement occurred. Any entries that are already saved in the database will not be saved again.

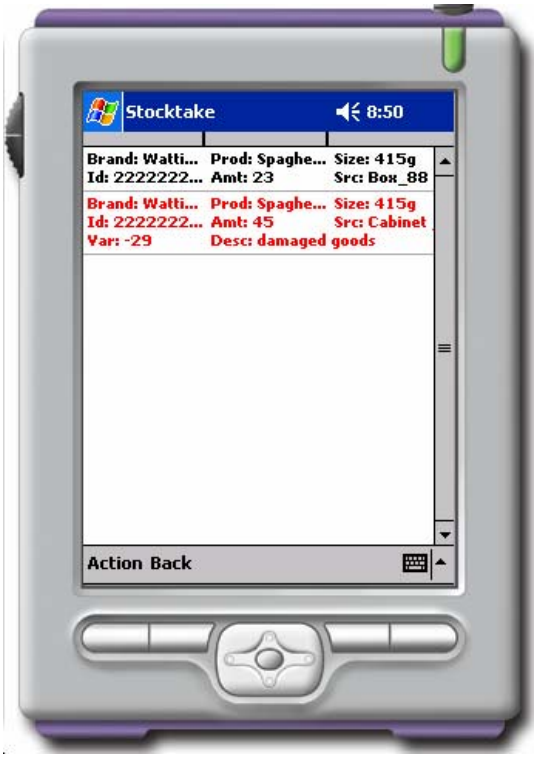
35) The “Stocktaking” screen is accessible by clicking “Stock-take” option under the “Action” menu in the “Order List Form”.

36) When the user scans a barcode of a product, the barcode will be validated, and product info will be shown on the screen.

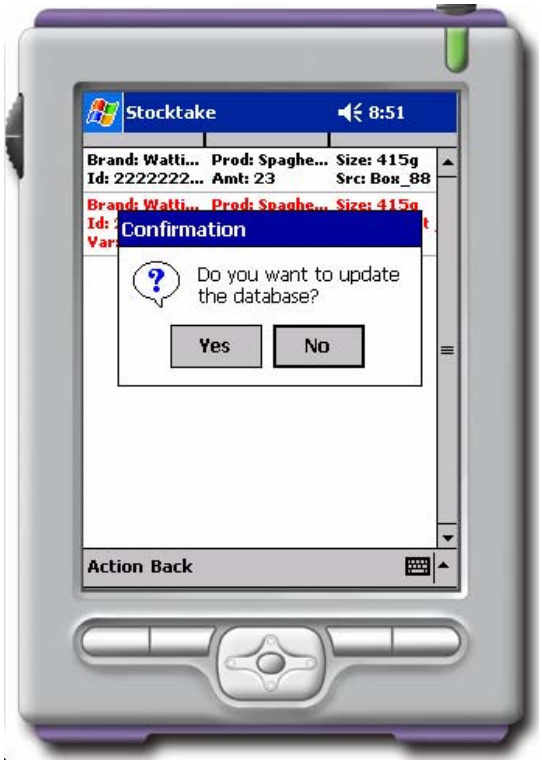
37) After the user fills in the source location ID and the amount field, he can click the “Confirm” button to confirm the check.

38) If the expected amount differs from the actual amount, the application will ask the user whether the entry is valid.

39) If the user confirmed that the two values are different, then the “Variance” page will be shown. The user needs to enter a reason for the variance.



40) An entry with variance will be shown in red in the “Details” page.



41) If the user wants to quit the application, the application will ask whether the user wants to save the entries into the database before exiting.

42) The “Details” page will indicate which items are saved in the database already. The “Details” page is accessible by clicking the “View” menu in the “Stock-take” form.

10. Appendix II – References

http://www.orbiz.biz

http://www.extremeprogramming.org/

http://www.codeproject.com/csharp/model_view_controller.asp?target=mvc

http://www.codeproject.com/dotnet/tdd_in_dotnet.asp?target=test%7Cdriven

http://msdn.microsoft.com

http://www.microsoft.com

http://dse.resultspage.com/search.php?sessionid=3fa104ed0b0d65ee273fc0a87f990709&site=&w=wireless

http://www.cdg.org/

http://www.cdmaonline.com/

http://www.bluetooth.com/

11. Appendix III– SQL Script

On top of the original database, the following SQL script is written to generate the dummy database used in the mobile inventory management system project.

```
--clear old data
PRINT 'DELETING STOCK_MOVE TABLE ...'
DELETE from stock_move
PRINT 'DELETING STOCK_VARIANCE TABLE ...'
DELETE from stock_variance
PRINT 'DELETING LOCATION TABLE ...'
DELETE from location
PRINT 'DELETING STOCK_LEVEL TABLE ...'
DELETE from stock_level
PRINT 'DELETING LOCATION TABLE ...'
DELETE from location
PRINT 'DELETING SITE TABLE ...'
DELETE from site
PRINT 'DELETING SITE_TYPE TABLE ...'
DELETE from site_type
PRINT 'DELETING ORDER_LINE TABLE ...'
DELETE from order_line
PRINT 'DELETING ORDER_HEADER TABLE ...'
DELETE from order_header

--clear the stock on hand
UPDATE stock SET stk_stock_on_hand=0
--clear the stock on order
UPDATE stock SET stk_stock_on_order=0

--config ------------------------------------------------------
DECLARE @NUM_SITE int
SELECT @NUM_SITE =30
DECLARE @NUM_SKL int
SELECT @NUM_SKL = 400
DECLARE @NUM_STOCK_AMOUNT_MAX int
SELECT @NUM_STOCK_AMOUNT_MAX = 50
DECLARE @NUM_LOC int
SELECT @NUM_LOC =70

DECLARE @NUM_STOCK int
SELECT @NUM_STOCK = 35 --Num of stock in the stock table, FIXED
DECLARE  @NUM_ORDER_HEADER int
SELECT  @NUM_ORDER_HEADER=4
-----------------------------------------------------------------------------------------
--declare var
DECLARE @SIT_NAME varchar(50)
DECLARE @SIT_LOCATION varchar(50)
DECLARE @STT_TYPE varchar(50)
DECLARE @STK_NAME varchar(50)
DECLARE @BRA_NAME varchar(50)
DECLARE @LOC_NAME varchar(50)
DECLARE @SKL_AMOUNT int
```

```
DECLARE @SKL_DATE_DUE datetime
DECLARE @stt uniqueidentifier
DECLARE @sit uniqueidentifier
DECLARE @prl uniqueidentifier
DECLARE @stk uniqueidentifier
DECLARE @skl uniqueidentifier
DECLARE @loc uniqueidentifier
DECLARE @syu uniqueidentifier


--clear old data
PRINT 'DELETING ORDER_HEADER TABLE ...'
DELETE from order_header
PRINT 'DELETING ORDER_LINE TABLE ...'
DELETE from order_line

--declare arrays
DECLARE  @SIT_NAMES varchar(200)
SELECT @SIT_NAMES =
'1,ABC,2,HH,3,SSE,4,Super,5,HHR,6,XPEE,7,Happy,8,ZS1967,9,OP4398,10,RE1967,11,JDEE
W,12,DF8383,13,DJNWQQ,14,'
DECLARE  @NUM_SIT_NAMES int
SELECT @NUM_SIT_NAMES = 13

DECLARE  @SIT_LOCATIONS varchar(200)
SELECT @SIT_LOCATIONS =
'1,Auckland,2,Hastings,3,Otago,4,Wellington,5,Nelson,6,Taupo,7,Queenstown,8,Dunedin,9,Hamil
ton,10,Palmerston North,11,New Plymouth,12,Timaru,13,Invercargill,14,'
DECLARE  @NUM_SIT_LOCATIONS int
SELECT @NUM_SIT_LOCATIONS = 13

DECLARE  @STT_TYPES varchar(200)
SELECT @STT_TYPES = '1,Warehouse,2,Vehicle,3,Order,4,'
DECLARE  @NUM_STT_TYPES int
SELECT @NUM_STT_TYPES = 3

DECLARE  @STK_NAMES varchar(200)
SELECT @STK_NAMES = '1,Spaghetti,2,Beans Chilli,3,Soup Big Red,4,Beans
Mexican,5,Spaghetti & Saus,6,Jam Raspberry,7,Soup for One,8,Peanut Butter Crunchy,9,Peanut
Butter Smooth,10,Baked Beans,11,'
DECLARE  @NUM_STK_NAMES int
SELECT @NUM_STK_NAMES = 10

DECLARE  @BRA_NAMES varchar(200)
SELECT @BRA_NAMES = '1,Eta,2,Heinz,3,Wattie     ,4,Good Taste Co.,5,Craig     ,6,Oak,7,'
DECLARE  @NUM_BRA_NAMES int
SELECT @NUM_BRA_NAMES = 6

--declare other var
DECLARE @COUNTER int
DECLARE @LIMIT_COUNTER int
DECLARE @LIMIT_COUNTER_MAX int
SELECT @LIMIT_COUNTER_MAX = 100
DECLARE @RAND_NUM int
DECLARE @RAND_GUID uniqueidentifier
```

```
DECLARE @RAND_NUM2 int
DECLARE @COUNTER2 int
DECLARE @COUNTER3 int

-------------------------------------------
--var for Order table
DECLARE @cus uniqueidentifier
DECLARE @order_number varchar(50)
DECLARE @orh_total_price int
DECLARE @orh uniqueidentifier
DECLARE @orl_amount int
DECLARE @orl_price int
-------------------------------------------
--insert predefined sites
INSERT INTO site_type(stt_type) VALUES ('Warehouse')
INSERT INTO site_type(stt_type) VALUES ('Vehicle')
INSERT INTO site_type(stt_type) VALUES ('Order')
-------------------------------------------
--insert predefined sys user
DELETE from sys_user WHERE syu_full_name='Tony Lee'
INSERT INTO sys_user (syu_full_name, syu_user_name, syu_password, syu_email_address,
syu_mobile_phone, syu_ddi_phone, syu_sql_user_name, syu_order_number, syu_order_prefix,
syu_admin, syu_deleted)
   VALUES ('Tony Lee', 'Tony', 'password', 'tlee@xtra.co.nz', '0211231231', '091231231',
'tlee054',0,'ton','F','F')
UPDATE sys_user
   SET syu_id='00000000-0000-0000-0000-000000000000'
   WHERE syu_full_name='Tony Lee'
-------------------------------------------
PRINT '############################'
PRINT 'CREATE SITE '
PRINT '############################'
--create site table
SELECT @COUNTER = 0
SELECT @COUNTER2 = 0
SELECT @COUNTER3 = 0

WHILE @COUNTER < @NUM_SITE
   BEGIN
   --loop until a valid site is obtained
      SELECT @stt = null
      SELECT @syu = null
      SELECT @LIMIT_COUNTER = 0
      WHILE (@stt is null) AND (@LIMIT_COUNTER < @LIMIT_COUNTER_MAX)
         BEGIN
            --EXEC sp_random_varchar @SIT_NAMES,@NUM_SIT_NAMES,@SIT_NAME OUT
            EXEC sp_random_string 5, @SIT_NAME OUT
            EXEC sp_random_varchar
@SIT_LOCATIONS,@NUM_SIT_LOCATIONS,@SIT_LOCATION OUT
            --EXEC sp_random_varchar @STT_TYPES,@NUM_STT_TYPES,@STT_TYPE OUT
            SELECT @RAND_NUM=RAND()*10
            SELECT @STT=null
            IF (@RAND_NUM<7)
               BEGIN
                  SELECT @STT_TYPE='Warehouse'
```

```
            END
          ELSE
            BEGIN
            SELECT @RAND_NUM=RAND()*10
            IF (@RAND_NUM<6)
              BEGIN
                SELECT @STT_TYPE='Vehicle'
              END
            ELSE
              BEGIN
                SELECT @STT_TYPE='Order'
              END
            END
          SELECT @stt = stt_id FROM site_type WHERE stt_type = @STT_TYPE
        END
      if (@STT_TYPE = 'Vehicle')
        BEGIN
          SELECT @syu=syu_id FROM sys_user WHERE syu_full_name = 'Tony Lee'
        END

      if (@STT_TYPE = 'Warehouse')
        BEGIN
          SELECT @SIT_NAME='W#'+@SIT_NAME
        END
      if (@STT_TYPE = 'Vehicle')
        BEGIN
          SELECT @SIT_NAME='V#'+@SIT_NAME
        END
      if (@STT_TYPE = 'Order')
        BEGIN
          SELECT @SIT_NAME='O#'+@SIT_NAME
        END
    --insert site info
      INSERT INTO site(sit_name, sit_location, sit_stt_id, sit_syu_id, sit_index)
        VALUES (@SIT_NAME,@SIT_LOCATION, @stt, @syu, @COUNTER)
    --ADD associate LOCATIONS
      if (@STT_TYPE = 'Warehouse') OR (@STT_TYPE = 'Vehicle')
        BEGIN
          SELECT @sit=sit_id FROM site WHERE sit_index=@COUNTER
          SELECT @COUNTER2 = @NUM_LOC
          WHILE @COUNTER2 >0
            BEGIN
              EXEC sp_random_string 5, @LOC_NAME OUT
              if (@STT_TYPE = 'Warehouse')
                BEGIN
                  SELECT @LOC_NAME='W#L#'+@LOC_NAME
                END

              if (@STT_TYPE = 'Vehicle')
                BEGIN
                  SELECT @LOC_NAME='V#L#'+@LOC_NAME
                END
            --insert site info
              INSERT INTO location( loc_name, loc_description, loc_sit_id, loc_index, loc_amt)
VALUES (@LOC_NAME,'Location # ' + RTRIM(LTRIM(STR(@COUNTER3))), @sit,
```

```
@COUNTER2, 0)
                SELECT @COUNTER2 = (@COUNTER2 - 1)
                SELECT @COUNTER3 = (@COUNTER3 + 1)
              END
        END
      SELECT @COUNTER = (@COUNTER + 1)
   END

PRINT
'################################################################################
#'
PRINT 'CREATE STOCK_LEVEL'
PRINT
'################################################################################
#'

--Change stk_barcode
--UPDATE stock SET stk_barcode=stk_index
------------------------------------------------------
--create stock_level table
SELECT @COUNTER = 0
WHILE @COUNTER < @NUM_SKL
   BEGIN

   --get random amount
      SELECT @SKL_AMOUNT = (RAND()* @NUM_STOCK_AMOUNT_MAX+1)

   --loop until a random stock is obtained (or excess limit)
      SELECT @RAND_NUM = (RAND()* @NUM_STOCK)
      SELECT @stk = stk_id FROM stock WHERE stk_index = @RAND_NUM

   --loop until a valid site is obtained (or excess limit)
      SELECT @RAND_NUM = (RAND()* @NUM_SITE)
      SELECT @sit = sit_id FROM site WHERE sit_index = @RAND_NUM


   --if site type is "order", then generate due date
      SELECT @SKL_DATE_DUE = getdate()

      SELECT @stt = sit_stt_id FROM site WHERE sit_id = @sit
      SELECT @STT_TYPE = stt_type FROM site_type WHERE stt_id = @stt
      if (@STT_TYPE = 'Order')
         BEGIN
         --random date
            --SELECT @SKL_DATE_DUE = ( ( STR( RAND() * 11 + 1  ) + '/' +  STR( RAND() * 27 +
1  ) + '/' +  STR( RAND() * 1 + 2003  ) ) )
            SELECT @SKL_DATE_DUE = getdate()
         END

      select @skl=null
      select @skl= skl_id  from site, stock_level, stock where stk_id=skl_stk_id and
sit_id=skl_sit_id and sit_id=@sit and stk_id=@stk

      if(@skl is null)
         BEGIN
```

```
                SELECT @skl=newid()

            --insert stock_level info
                INSERT INTO stock_level(skl_id, skl_amt, skl_date_due, skl_stk_id, skl_sit_id,
skl_index)
                    VALUES (@skl, @SKL_AMOUNT, @SKL_DATE_DUE, @stk, @sit, @COUNTER)
            END

        PRINT @STT_TYPE
        --if the site type is not on order
        if (@STT_TYPE != 'Order')
            BEGIN
                PRINT 'NOT ORDER'
                SELECT @COUNTER2 = (RAND()*3)+1

                WHILE (@COUNTER2>0)
                    BEGIN

                        SELECT @loc = null
                        WHILE (@loc is null)
                            BEGIN
                                PRINT 'LOC is NULL'

                                SELECT @RAND_NUM = (RAND()* @NUM_LOC)
                                SELECT @loc = loc_id FROM location
                                    WHERE loc_sit_id = @sit
                                        AND loc_index=@RAND_NUM
                                        AND loc_amt=0

                            END
                        UPDATE location
                            SET loc_skl_id= @skl,
                            loc_amt= (RAND()* 30)+1
                            WHERE loc_id=@loc

                        PRINT 'guid='
                        PRINT @RAND_GUID

                        SELECT @COUNTER2 = (@COUNTER2 - 1)
                    END
            END
        SELECT @COUNTER = (@COUNTER + 1)
    END

PRINT '################################################################'
PRINT 'CREATE ORDER HEADER TABLE'
PRINT '################################################################'
--create order_header table
SELECT @COUNTER = @NUM_ORDER_HEADER
WHILE @COUNTER > 0
    BEGIN
    --get sys user
        SELECT @syu=syu_id FROM sys_user WHERE syu_full_name = 'Tony Lee'
    --get random customer id
```

```
      SELECT @RAND_NUM = (RAND()* 72)
      SELECT @cus=cus_id FROM customer WHERE cus_index = @RAND_NUM
   --get random order number
      SELECT @RAND_NUM = (RAND()* 3) + 1
      EXEC sp_random_string @RAND_NUM, @order_number OUT
      SELECT @RAND_NUM = (RAND()* 100000)+10
      SELECT @order_number = @order_number +  LTRIM(RTRIM( STR(@RAND_NUM)))
      --SELECT @order_number = @order_number
   --get random price
      SELECT @RAND_NUM = (RAND()* 1000) + 20
      SELECT @orh_total_price = @RAND_NUM
   --insert order_header data
      INSERT INTO  order_header(orh_syu_id, orh_cus_id, orh_order_number, orh_total_price,
orh_payment, orh_date)
         VALUES (@syu, @cus, @order_number, @orh_total_price, 'CASH ',  getdate())
      SELECT @COUNTER = (@COUNTER - 1)
      SELECT @orh=orh_id FROM order_header
         WHERE orh_syu_id=@syu
         AND orh_cus_id=@cus
         AND orh_order_number=@order_number
         AND orh_total_price=@orh_total_price

      SELECT @RAND_NUM2 = (RAND()* 5)+1

      SELECT @COUNTER2 = 0
      WHILE @COUNTER2 < @RAND_NUM2
         BEGIN

            SELECT @RAND_NUM = (RAND()* 35)
            SELECT @stk=stk_id FROM stock WHERE stk_index = @RAND_NUM

            SELECT @RAND_NUM = (RAND()* 9)+1
            SELECT @orl_amount=@RAND_NUM

            SELECT @RAND_NUM = (RAND()* 50)+1
            SELECT @orl_price=@RAND_NUM
            INSERT INTO  order_line(orl_orh_id, orl_stk_id, orl_amount, orl_price, orl_confirmed)
               VALUES (@orh, @stk, @orl_amount, @orl_price, -1)
            SELECT @COUNTER2 = (@COUNTER2 + 1)
         END
   END
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```