A Warehouse Management System Integration Solution

David Jones

October 2017

Dissertation submitted in partial fulfilment for the degree of
Master of Science in Computing for Business

Computing Science and Mathematics
University of Stirling

Abstract

This project discusses a software solution developed to integrate two business' Warehouse Management Systems and supporting processes.

United Closures and Plastics Ltd (referred to as UCP), a manufacturing business in Bridge of Allan, Stirlingshire, manufacture stock of bottle-tops for the Whisky industry. WH Malcolm (referred to as Malcolms) are a warehousing specialist based in Newhouse, Motherwell, and subcontracted by UCP to manage a large proportion of their stock.

Each business operates a separate implementation of a software type known as a "Warehouse Management System" (referred to as WMS). A WMS is an inventory and stock control tool. Each WMS system contains their own distinct inventory databases, and some data exists in both systems simultaneously. Changes to this "shared" data must be maintained manually by a human administrator in both systems, which is time consuming and prone to error.

The primary objectives of the project were to perform a comprehensive analysis these manual administration activities and document the issues. Consequently, build a solution that addresses the problems outlined. An automated "Inter-WMS" system is presented, a bespoke middleware & message exchange solution consisting of two key modules:

The first module, referred to as the *WMS Integration Monitor*, is a console application that automates the synchronisation of stock record updates by checking for in scope changes in one system and simultaneously processing the equivalent change in the alternate system. This automates monotonous manual replication activities and crucially, includes a function referred to as the *Status Update***,** addressing a mission critical loophole in the existing process.

Secondly, the *Deliver to Warehouse* module adds new functionality to the system for a very specific data capture and exchange requirement; the process of sending stock from UCP to Malcolms on a delivery vehicle.

Both modules overlay a Microsoft SQL server database, the *SQL Integration Database,* designed specifically for this requirement and acting as the "glue" between both WMS systems; a "WMS to WMS" integration database with message exchange and transaction history.

Together, these modules and functions "bridge the gap" between the UCP and Malcolms WMS systems replacing excessive and time consuming legacy administration procedures.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following:

- The UCP WMS System introduced in 2.1 was created and implemented by Jungheinrich on behalf of UCP. All native functions existed prior to this project. This is clearly stated throughout with the application only interrogating and updating the existing UCP WMS Database accordingly.
- The Malcolms WMS system introduced in 2.1.5 is entirely owned and operated by the Malcolms business. Integration and data exchanges were subject to Malcolms' approval and support through the use of Malcolms supplied data exchange message formats, specified in 3.5.5, 3.6.5, and 3.6.6.

Signature **Date**

Acknowledgements

I would like to acknowledge my project supervisor, Dr Mario Kolberg, for providing valuable guidance and support during the process of writing this project. Having found this part challenging at times to pull together, Mario has answered questions clearly and provided clarity in areas of uncertainty.

I would like to thank my wife Ella, despite the long days at work and late nights studying, she has been patient and understanding throughout, which I cannot thank her enough for.

Table of Contents

List of Figures

1 Introduction

1.1 Problem Background and Definition

United Closures and Plastics LTD (UCP) are a manufacturing company based in Bridge of Allan, Stirlingshire, employing 350 people. UCP manufacture various types of products known as closures (or "bottle-tops") for the global "Wine and Spirits" industry. They specialise in anti – counterfeit products, primarily designed for the export market. Their customers demand protection on their high value drinks which are counterfeited and refilled in many countries. This is particularly evident for globally recognised Scotch whisky brands.

The anti-counterfeit manufacturing process requires many plastic and metal component parts. These are subsequently assembled to achieve the anti-counterfeit closure. Due to the multistage manufacturing process, there is a need to manage a flat and wide array of semi-finished products in the business. This results in the scheduling of production being an intricate process driven by accurate stock management.

UCP use a software tool known as a Warehouse Management System (referred to as WMS) to control and view their stock. The WMS comprises an Oracle database which contains a full inventory of stock data records, history, properties, and a plethora of functions geared towards a manufacturing business. Changes and updates to stock data records at the UCP factory itself are performed in real time using WMS mobile computers that transact updates directly to the database.

In addition to "on-site" stock management, UCP use an external Warehousing company WH Malcolms (referred to as Malcolms) for external storage of stock, due to limitations in physical space in the factory itself. Malcolms use a separate WMS to manage their own operation. Their WMS also contains its own separate distinct database, containing an inventory of all Malcolms customer stock data records. Some of Malcolms stock therefore belongs to UCP as their customer.

For UCP stock stored at the Malcolms warehouse, associated stock data is therefore held in two WMS databases; the UCP WMS and the Malcolms WMS. This situation is at the source of the issues being described in the rest of this document. Maintaining data consistency across both systems is an extremely laborious and time consuming effort. Any change to this data in one system must be replicated in the alternate WMS. Complicating matters, the types of stock data changes are varied and cover several business needs and processes.

There is no system to "bridge the gap" in this situation, thus human intervention is required. This manual administration effort at both Malcolms and UCP is paid for by the UCP business. It is therefore in UCP's primary interest to reduce this administrative cost from both an efficiency and financial perspective.

Figure 1 highlights the scenario from a top level. When a change is made in either the UCP or Malcolms WMS, a human administrator replicates this change in the alternate system.

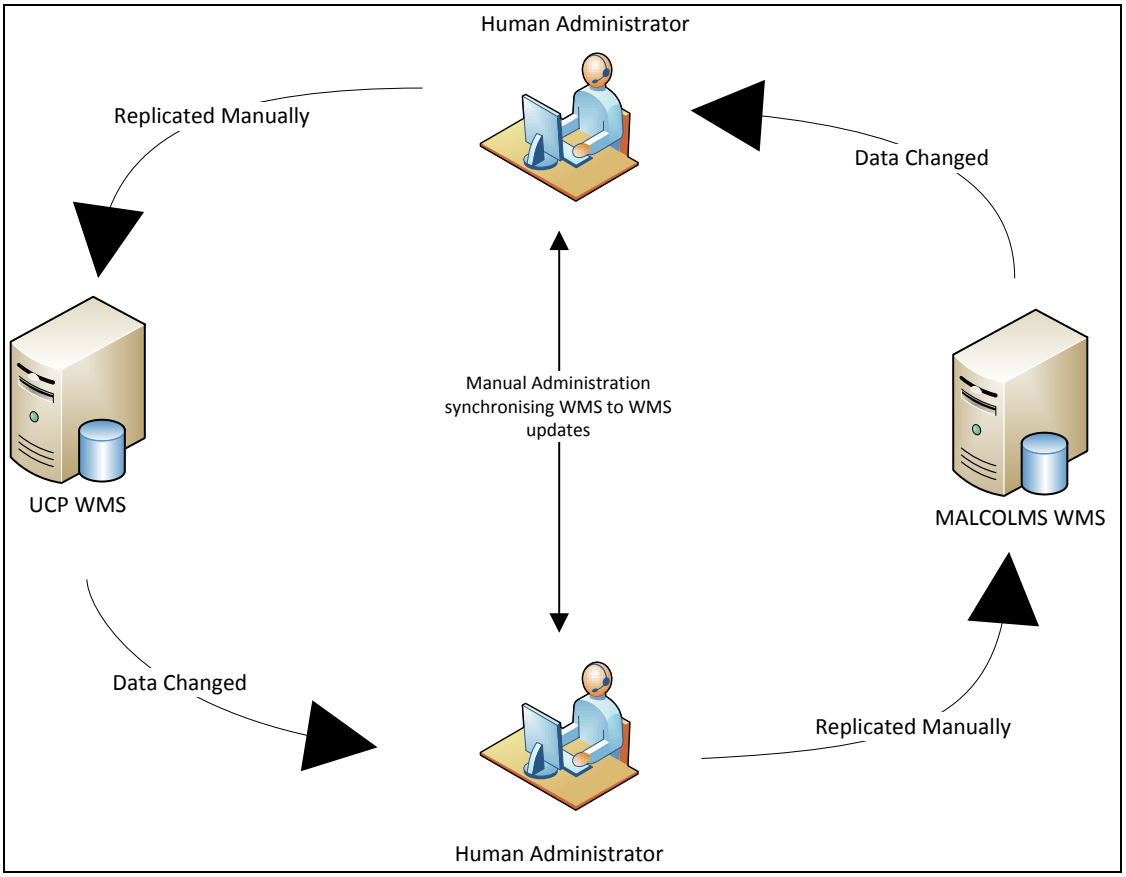


Figure 1: Problem Overview

This process is expanded upon in further chapters (see *2.1.7 Manual Administration Problems* on page 13) but highlights the crux of the problem being addressed.

1.2 Scope and Objectives

The project has been undertaken on behalf of the UCP business, and aims to address several inefficiencies that exist maintaining data consistency between its own WMS implementation, and their warehousing provider's separate WMS. The inefficiencies result in excessive time wasted by office administrators and operational staff members duplicating data through form filling or data entry.

The problems occur when stock is physically moved from one location to another, or stock records are changed in one system, and need to be replicated in another. These activities involve a human interface. The objectives of the project are to address these problems directly by reducing the human interface as much as possible.

The scope of the problems cover two scenarios in terms of the stock data records affected. The first covers inconsistencies relating to stock records that exist in both systems. The second relates to stock records that exist in the UCP WMS only, but are being delivered to the Malcolms Warehouse and therefore must be created in the Malcolms WMS when they arrive.

For adjustments to stock records that exist in both systems, the objective is to provide a fully automated data exchange that passes these changes between one another. The application must be able to run without user intervention, and provide a monitor, trigger, and message creation mechanism to pass change details between systems.

For stock being transferred from UCP to the Warehouse, the objective is to provide a combined data collection and data exchange mechanism. A brand-new function is required to automatically exchange stock records being delivered to Malcolms, replacing the need to manually enter these records when they arrive. At the same time, the UCP delivery logging and filing system must be replaced by integrating historical delivery schedule information as a capability.

1.3 Assumptions

The project is undertaken entirely on behalf of UCP. The business problems being addressed relate directly to this company's requirements. However, as covered in the introduction, the problems and systems discussed cover two businesses and their own WMS systems, UCP and its Warehousing supplier, Malcolms.

Although discussed at some length from a process analysis perspective, it must be noted that the Malcolms WMS implementation is completely off limits when it comes to directly inserting or retrieving data from their WMS. This is a common occurrence when working on an integration project spanning multiple businesses. As a result, Malcolms Application Support resource will be crucial in supporting the delivery of these specific types of functions. Usually, this has been achieved by agree-

ment on a message exchange format that suits both parties. Lack of such input from Malcolms would render the objectives unachievable as a result.

Finally, and for the avoidance of doubt, there is ***no requirement*** to change the underlying functionality of the native WMS software itself. The problems being discussed are solely related to the context of the data exchange between the two WMS implementations in a unique situation; something the systems were never designed to do in the first place. The native WMS implementations in isolation are both functionally and technically sound and entirely meet the business needs in this context. The original vendors have implemented the same core software successfully across many warehousing, logistics, and manufacturing businesses globally. It is the unique situation that is the issue, not the individual WMS systems.

1.4 Achievements

The project has been extremely challenging on many fronts, addressing real industrial problems in a fast paced manufacturing environment.

Having been given a general problem area by UCP, an intensive examination and dissemination of existing processes and problems has been achieved. This has spanned many areas of the business. At times it was often in stressful and unorganised environments. Articulation of these results provided valuable insight into technical, management and cultural challenges ahead. This was very well received by business stakeholders.

Research into technologies has proven enlightening, the detail of which will be remembered and considered during future academic and professional projects. Data integration platforms that were previously overlooked or ignored as being unnecessary and irrelevant in previous professional projects will be seriously considered following a broader investigation into their purpose and capability.

Performing a full software development lifecycle project for the first time has been extremely rewarding. In particular, addressing problems with the design and evolution of two “made to measure” applications. Seeing these applications deliver as expected, and understanding the benefits they will bring, has made all of the challenges worthwhile. In terms of exposure to developing applications from the ground up, technical hurdles were overcome that had never been seen before. This proved to be a major catalyst for learning, in particular in the areas of database management system developed, data exchange, and integration with object oriented design concepts.

Having built a bespoke application for the UCP business, a tool exists that effectively removes the existing problems entirely. This should ensure that UCP can make start to make the most efficient and effective use of the WMS software for some time to come.

1.5 Overview of Dissertation

Following the introduction, the remaining research will be presented as follows.

Chapter 2 (State-of-The-Art) presents research on the existing problem. Section 2.1 describes in some detail the existing software, processes, and tools used to communicate with the Malcolms Warehouse. A major component of this, the UCP WMS software is introduced. This describes what it is, how it works, and the data schema is introduced to help understanding in later chapters. Finally, the resulting problems with the existing system are defined and summarised holistically. Section 2.2 presents research on existing solutions and software that address the same or similar problems, expanding into further detail in section 2.3 which looks at these solutions implemented within the industry. Finally, project management research is presented in the area of the software development lifecycle, supporting the approach to delivering the project from concept through to delivery.

Chapter 3 (Requirements) describes the requirements gathering process, and breaks the problem down into a set of "User Stories", the problem categories that will define the subsequent design and development work being proposed.

Chapter 4 (Design) extends the requirements needs into a set of design phases, for each user story. Technical design considerations in relation to the requirements gathered are discussed, with the application structure prepared and articulated accordingly.

Chapter 5 (Implementation) presents the creation and implementation of the design concepts introduced in Chapter 4. Supporting infrastructure, problems, and additions are discussed further. Closing the implementation, a set of testing scenarios are described including the associated results in 5.4.

Finally, Chapter 6 concludes the research, summarising the results and presenting a critical evaluation of the work done. A discussion on future work closes the research in 6.3.

2 State-of-The-Art

This chapter will introduce the state-of-the-art into several distinct categories. The first will introduce the existing system, a concept that focusses on the way the UCP business operate its own Warehouse Management System and the challenges integrating with its Warehousing supplier's own system. The component parts of the existing system are common throughout many organisations but the problems faced in this situation are quite unique.

The second section reviews alternate data exchange solutions, providing examples of their implementation in industrial environments. Finally, a review of project management methodology in relation to the software development lifecycle is presented.

2.1 UCP's Existing Manual System

This section discusses the existing system and the associated functions, problems and procedures associated with it. The problems will be referenced in later chapters with regards the design decisions being made.

The term "Existing System" refers to all systems, such as software, processes, procedures and stationary that relate to the underlying problem. This combination of software, procedures, and business processes pays attention to the way in which UCP and Malcolms integrate their WMS systems with one another. This chapter introduces the following components of the existing system:

The UCP WMS

An overview of the UCP WMS system, how it works, and the data it contains as a standalone piece of software. In addition, the WMS stock data structure and concept is described and clarifies the terminology used for critical discussion later on.

The Malcolms WMS

The Malcolms WMS is fully owned and operated by the Malcolms business, but some key assumptions are introduced at this stage to set the scene for the work that will be proposed.

Incorporating an External WMS

An introduction to the use of the UCP WMS in relation to data sharing with an external WMS. A discussion on the activities that require WMS to WMS data sharing, and the mechanisms that enable synchronisation across both systems.

Problems with the existing systems

In relation to each WMS to WMS activity, an explanation of the manual administration processes and business impacts is provided.

2.1.1 The UCP WMS: Overview

The UCP WMS infrastructure is hosted at the UCP factory and functions as a standard client-Server database application. The package is comprised of an oracle 11g database and an associated software management application that run on a Windows Failover Cluster providing the necessary server software and database services. Client side, a WMS front end application exists for both Windows desktop PC's and mobile windows hand terminals. The database is the master repository for all stock data records, with the mobile hand terminals being used to transact changes to this data in real time. PC clients provide supervisory reporting and control functions for office staff.

The WMS Software and Database at UCP is provided by a vendor, Jungheinrich, who specialise in warehousing and storage solution.

2.1.2 The UCP WMS: Oracle Database and Stock Data Structure

The Oracle database contains stock data records that belong to the company UCP. In addition, each stock data record has associated properties that support multiple functions within the WMS. These functions are vast and go beyond the scope of this research but some critical observations must be understood:

- ✓ Each physical piece of stock is represented by a stock record in the database
- ✓ Each stock record has several properties, one of which is a binary ***status*** property – this determines if the stock is defective; a property that forms a critical control function discussed later. The status indicates if it can be used, and if a stock record is defective it becomes ***blocked*** in the WMS until an auditor positively ***unblocks*** it following quality checks.
- ✓ Each stock record has an associated ***location*** record in the database. The location record in the database represents a simple concept, e.g. a physical location. Therefore:
    a. Stock located at Malcolms will have a location record of "the warehouse".
    b. Stock located at UCP will have a location record of "the factory", and so on.
- ✓ When stock is produced from a machine, it is packaged in a container made up of one or more "boxes" of product, and this container in its entirety is equal to one stock record. The stock data record in the oracle database represents this container, contents, and properties.

For clarity, the physical layout of this stock is shown in Figure 2. We can see the box, the stock record label (Figure 2, marked with comment), and the load carrier (Figure 2, marked with comment). Further discussion shows how this stock is represented and structured as a record in the database, with a screenshot of a database view in the native WMS software (Figure 4).

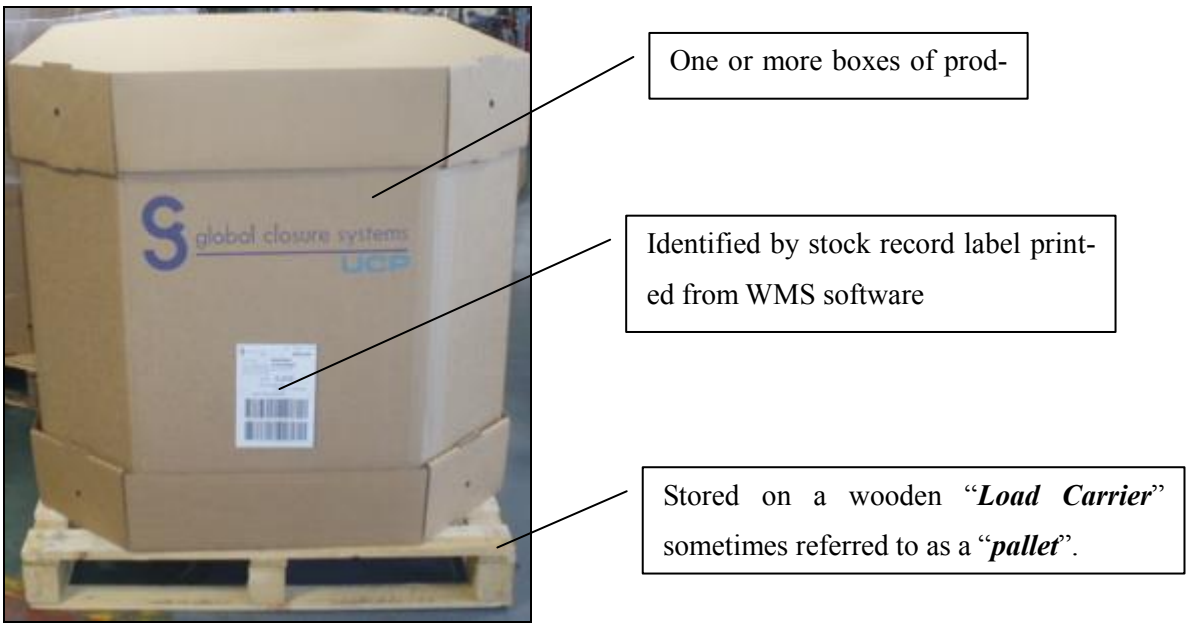


Figure 2: Stock Record Physical Representation

Each stock record has a unique stock identifier (primary key) which is associated to various properties. Note: The stock id is meaningless and arbitrary to a human operator, rather the associated record properties support user needs.

This example of a stock record would also be stored in a physical location. The reality of this location concept is extremely simple. One location record relates to one physical storage location in the plant. Figure 3 shows a storage rack, with storage spaces in the rack. Each storage space is represented as a location record in the WMS database and stock can be associated to exactly one location record only (it can only be stored in one space at any one time). Each of these locations is represented as a location record in the WMS Database.

Figure 3: Storage Locations Physical Representation

An example view of the WMS stock data record is displayed in Figure 4. Each record represents one stock based on the structure described above, with each record having various properties and associated records associated to it. Figure 4 shows a number of columns (commentary on following page) and a number of stock records below.

LC no.	Product Co..	Status	Description		Loc.	Quantity	Batch	Line numb..	P
3503713300001363936	05838540	available	WPM528 IMPORTED G/Y/MAR 0C	30X44 IMPORTED G/YEL/MA..	DGS-A -21-	20,900	24453063	845FS02	9
3503713300001363721	05838540	available	WPM528 IMPORTED G/Y/MAR 0C	30X44 IMPORTED G/YEL/MA..	DGS-A -21-	20,900	24453063	845FS02	7
3503713300001363615	05838540	available	WPM528 IMPORTED G/Y/MAR 0C	30X44 IMPORTED G/YEL/MA..	DGS-A -21-	20,900	24453063	845FS02	6
3503713300001364100	01145666	available	153 BODY TIVA	TIVA ASSEMBLY	DGS-C -18-	133,000	24463579	845IN19	8
3503713300001364001	05838540	available	WPM528 IMPORTED G/Y/MAR 0C	30X44 IMPORTED G/YEL/MA..	DGS-C -21-	20,900	24453063	845FS02	1
3503713300001363950	08512730	available	WNR543 JWRL R/G/ALLIANCE 0C	30X54 JWRL RED/GOLD ..	DGS-D -02-	15,960	24264191	845MD..	1
3503713300001364247	08625427	available	PLS287 MIDAS I/RATCH 1.1 GLD/K	MIDAS INNER RATCHET 1.1 ..	DGS-D -06-	45,000	24402274	845IN06	4
3503713300001363813	08625427	available	PLS287 MIDAS I/RATCH 1.1 GLD/K	MIDAS INNER RATCHET 1.1 ..	DGS-D -06-	45,000	24402274	845IN06	4
3503713300001364353	08542031	available	PLS315 ORCHID OPEN POUR BODY	ORCHID OPEN POUR BODY	DGS-D -14-	21,000	24336545	845IN29	2
3503713300001363844	05838540	available	WPM528 IMPORTED G/Y/MAR 0C	30X44 IMPORTED G/YEL/MA..	DGS-D -15-	20,900	24453063	845FS02	8

Figure 4: Stock Records in UCP WMS

Commentary on the columns displayed in Figure 4, showing a view of WMS stock record columns and properties:

LC No. = Load Carrier (the wooden pallet described in Figure 2).

Status = Critical record property relating to the stock suitability. Either "Blocked" or "Available".

Loc = Location record relating to the physical location the stock is stored. (see Figure 3)

Product = Product type of stored on the Load Carrier.

With the above understanding, the following statement could be made about the first record shown in Figure 4 above:

---

*Stock on **LOAD CARRIER** 350371330001363936 is being stored in the rack **LOCATION** DGS-A -21. It contains **PRODUCT** 05838540 and the stock **STATUS** is AVAILABLE*

2.1.3 The UCP WMS: Front End Application

The UCP WMS front end application is presented in two forms; a "rich client" for Windows PC operating systems, and a Windows mobile client for use on handheld terminals.

The PC Client is primarily used by office administration and supervisor staff. It has a full range of features that allow a user to run reports, update stock, and give instructions to operations staff. It is designed to interrogate the WMS database, run reports, and enable management and business decisions to be made.

The Windows mobile client is used by operational staff, the people who physically handle stock in the factory, such as machine operators and fork lift truck drivers. The handheld mobile clients are design to *transact* data changes to stock records. For example, when a stock is moved, or changed physically, the operational staff member uses the mobile PC to transact such a change. This ensures stock updates and changes in the UCP factory are executed at source and updated in real time.

Figure 5 provides an abstract overview of this "real time" concept. The business has a set of physical activities that result in stock being created or changed (Figure 5, *Physical Activities)*. WMS Data Capture transactions are performed and fed back to the WMS Database using mobile computers (Figure 5, *WMS data capture and transactions).* This enables real time reporting for supervisory office staff using the PC client.

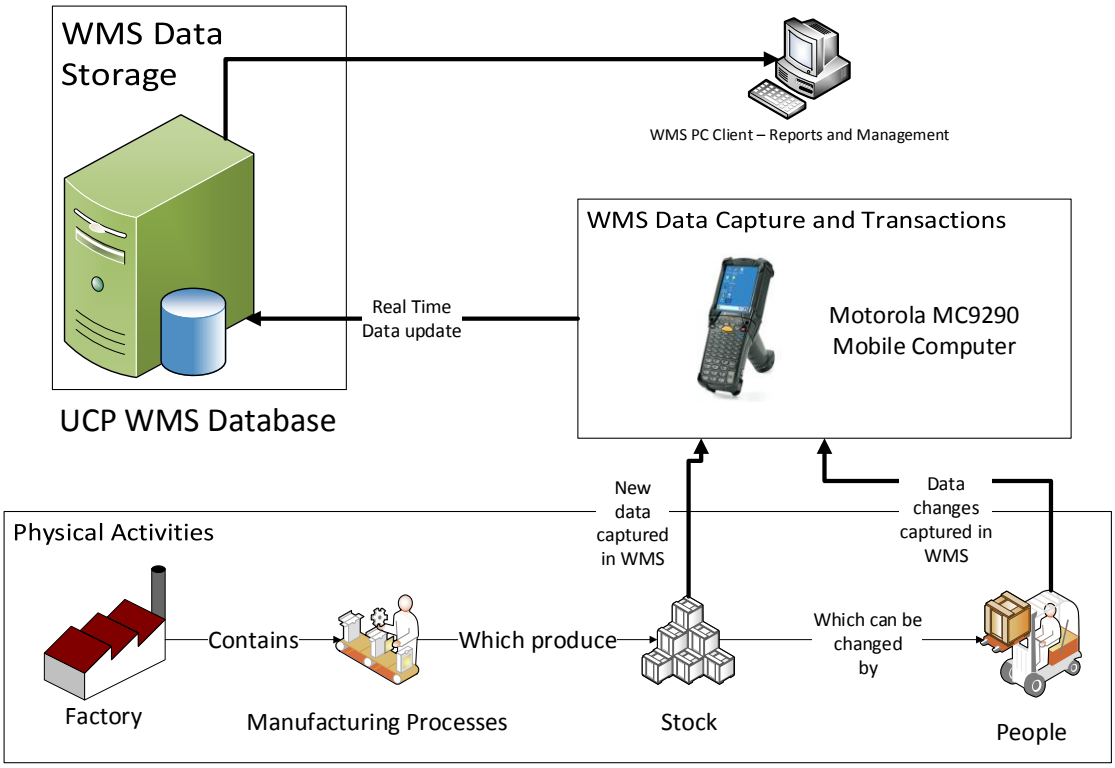


Figure 5: UCP WMS Data Capture

2.1.4 The UCP WMS: Final Discussion

The above discussion covers the WMS system and concept at UCP only. It is responsible for stock control and management at the factory, performing many stock related functions. However, the following observations are critical understandings in the context of the rest of this paper:

- ✓ The WMS Database contains an inventory of all UCP stock data records
- ✓ Each stock data record represents a physical load carrier of stock (commonly known as a "pallet" in the industry
- ✓ Each stock data record is associated to a location record. Location records represent physical storage locations.
- ✓ Each stock record has a number of properties, including stock status
- ✓ Whenever a stock is created, changes location, or has a property changed, the transaction is made on a WMS mobile handheld device which updates the database in real time.

2.1.5 The Malcolms WMS

The underlying functions of the Malcolms WMS is out of the scope of this research, but for the argument being presented we must understand some key points of note. The Malcolms WMS is separate to the UCP WMS, and it has never been designed to work with the UCP Software. It also cannot be

changed functionally or operationally by UCP directly, as the software is fully owned and operated by the Malcolms business. In addition, it is configured to cater to their own business needs and functions. It is a coincidence that both the UCP factory and Warehousing partner operate similar WMS installations as their primary stock management software. Regardless, the two applications in their native form are geared towards different objectives and therefore functionally very different; one being geared toward manufacturing processes, and the other towards logistics and distribution services.

One similarity of each system is the stock data structure (previously discussed in 2.1.2). In fact, the data record structures are almost identical across each database, which lends itself well to any data exchange solutions, as source data will not need to be heavily treated in terms of its basic composition. Such similarity across different WMS systems is not uncommon, as they all follow the same logic and manage the same types of data.

The key understanding here is that the Malcolms WMS is off limits in terms of replacing or changing its core purpose or function. UCP must work with the Malcolms Warehouse in its current form and based on its current use of their existing WMS to seamlessly address the issues using new software and tools developed for this purpose.

2.1.6 Incorporating an External WMS

With an understanding of the WMS concept and its specific use at UCP, the next component of the existing system can be discussed; integration with an external warehousing company that use a separate WMS implementation.

In the previous chapter, the concept of stock data records, associated properties and location records were introduced. Each of these objects were maintained in real time by operators using handheld computers to perform WMS transactions as stock is handled physically. This provided a real-time link to the database, so the stock data reflected the stock reality at all times. Where this process fails is at the point stock needs to be stored at Malcolms. This is primarily due to two main reasons:

1. When the stock is physically moved to the Malcolms warehouse, transactions are performed on mobile handhelds computers in the Malcolms WMS, at their site.
2. The Malcolms mobile handhelds are integrated into a completely separate WMS system, thus the UCP WMS system must be updated manually as a secondary process.

In isolation, the UCP WMS provides an integrated system for executing transactions in real time and associated reporting. However, for transactions performed at the warehouse, using a separate WMS, the link is broken. Thus, for warehouse stock changes, human operators must “bridge the gap” and manually update one WMS when a change is made in the other. This manual human administra-

tion is the issue that must be addressed and discussed further in this paper. The specific activities that are applicable to the manual administration is covered below.

2.1.7 Manual Administration Problems - Scope

The problem applies to a very specific portion of all the stock data held within both systems. If we understand that:

1. The UCP WMS contains all stock data records that belong to UCP, some of which is *located at the Malcolms Warehouse*, and;
2. The Malcolms WMS contains stock records for all of their customers, *some of which belongs to UCP.*

We can therefore clearly identify the definition of the problem data as: "UCP Stock data records that are physically stored at the Malcolms Warehouse". Such data records will be held across two WMS databases and consistency must be maintained across each, exposing the scope of the entire problem. This "shared stock data" across two WMS databases is visualised in the abstract in Figure 6. This shows two WMS implementations (UCP and Malcolms WMS databases) with each containing a set of Stock Data records, split into two sets. The blue set indicates stock data records that are completely out of the scope of the issue. The red set highlights the shared stock data problem; in the UCP database, this relates to stock data records with a location relating to the warehouse. In the Malcolms WMS database, this relates to stock data records that are the property of their UCP customer:

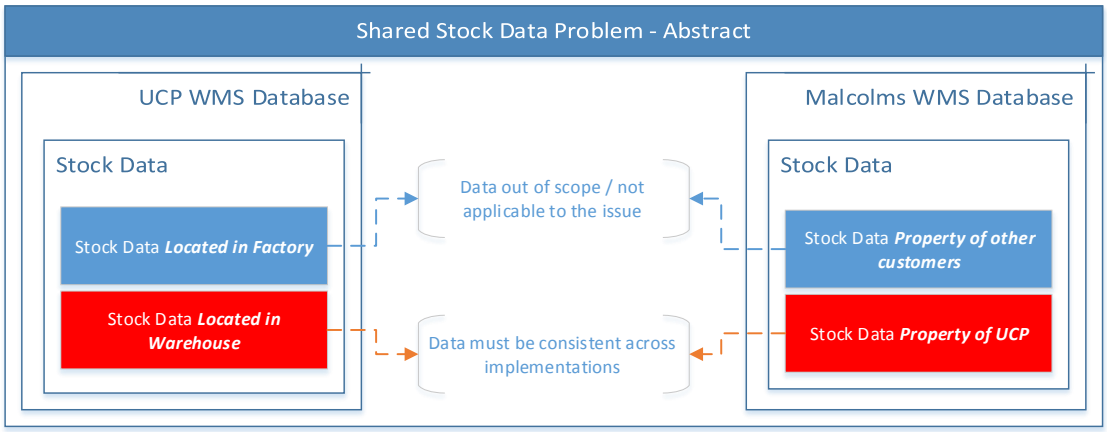


Figure 6: Shared Stock Data Concept

The specific manual administration activities that fall within the scope of the shared stock data are listed in the subsequent commentary in this chapter. Each of these activities form the basis of the human interface required to link the two systems, ultimately defining the problems that need to be addressed.

2.1.8 Problem 1: Delivery to Warehouse

The first issue describes the process of UCP delivering some stock to Malcolms. Stock is physically loaded onto a vehicle and the stock records updated with the vehicle location in the UCP WMS. As soon as the delivery vehicle leaves the factory, the integrated transactions provided through the UCP WMS are broken and must rely on manual inputs for subsequent updates. When the stock arrives at the warehouse, an administrator enters the stock data delivered into the Malcolms WMS. At the same time, he updates the stock data location in the UCP WMS through a remote access terminal. At this point, a state of shared stock exists on both systems.

Figure 7 shows the manual effort that relates to this concept, with manual administration procedures (paper forms and data entry) creating the "link" from the UCP WMS (shown on the left) to the Malcolms WMS (shown on the right).

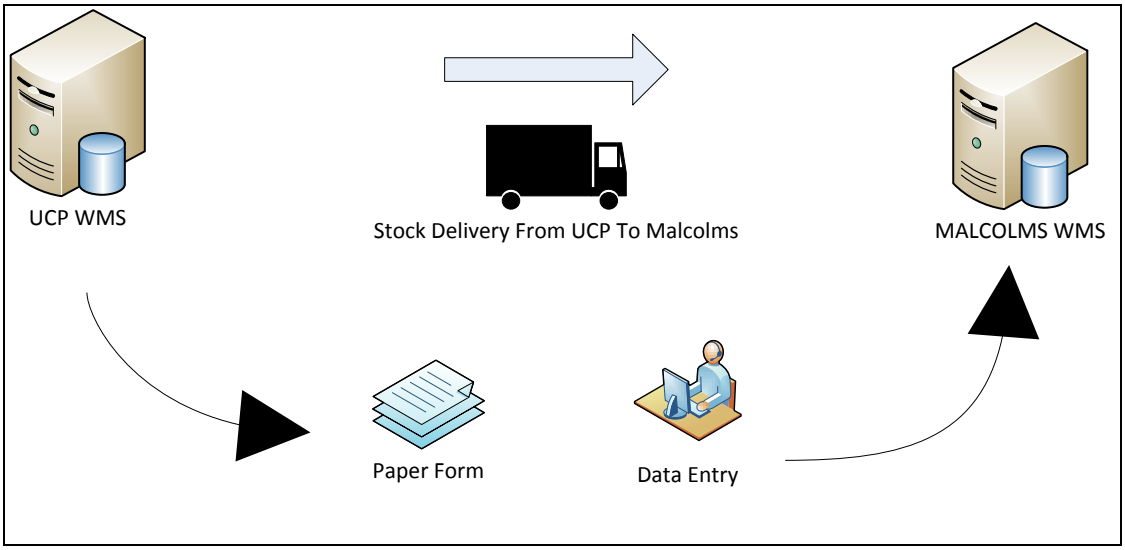


Figure 7: Problem Overview Delivery to Warehouse

Table 1 expands on this concept, providing a detailed set of activities that must occur to make this "link". Each steps is described, with estimated effort (time spent) and an actor highlighting the people involved at each stage.

Scope of Activities: Stock is delivered from the UCP factory to the Warehouse

Purpose of Activities: The activities described allow the Malcolms administrator to both *create* the stock in the Malcolms WMS upon arrival, and *update* the stock data location in the UCP WMS. The copies of the stock record labels provide the information needed. An additional data label is applied to UCP stock during the manufacturing process to aid this specific need.

Frequency: Performed in full for each delivery. 4 – 5 deliveries each day. 35-45 stocks on every delivery.

Activity	Actor
Copies of stock labels taken from each stock being delivered Effort: 5 minutes	UCP delivery operator
Two delivery advice forms completed. One filed at UCP and the other handed to delivery driver Effort: 6 minutes	UCP delivery operator
Stock delivered to Malcolms Warehouse	Delivery Driver
Copies of stock labels handed to Malcolms administrator Effort: 1 minutes (covers steps 5)	Delivery Driver
Delivery advice note handed to Malcolms administrator	Delivery Driver
Delivery advice note filed by Malcolms administrator Effort: 2 minutes	Malcolms Administrator
Data on stock labels entered into spreadsheet Effort: 10 minutes	Malcolms Administrator
Spreadsheet data imported into Malcolms WMS using import tool developed at Malcolms. Effort: negligible	Malcolms Administrator
Stock data updated in UCP WMS through VPN connection Time: 3 minutes	Malcolms Administrator

Table 1 - Manual Administration - Delivery TO Warehouse

The steps are time consuming and involve significant data duplication. In terms of waste identification, we can see an overall effort of almost 30 minutes is spent for every delivery made. In addition, six instances of data duplication / data entry activities. This represents a huge manual effort that is prone to error.

2.1.9 Problem 2: Stock Status Update

Maintaining a stock record's status property is one of the most business critical transactions performed in the UCP WMS. A binary value, it represents "good or bad" stock, with the definition "Available or Blocked" being used at UCP. If the status is "blocked" the stock cannot be delivered to a customer or used in a subsequent manufacturing process. This is necessary to ensure that non-conformant quality issues can be stopped in their tracks before stock is delivered or manufactured further. As stock is continuously being delivered to customer (24 hours per day, 365 days a year) the immediate update of stock status is mission critical. Failure to do so could result in defective stock being delivered to the customer. For UCP stock stored at the warehouse, status changes need to be emailed or phoned through to Malcolms so that a change made in the UCP WMS can be replicated and in a timely manner. Figure 8 shows an overview of the human link involved in this process, starting with a Quality Analyst making a change in the UCP WMS (Figure 8, left of the image), and an email or telephone call linking this change to manual data entry in the Malcolms WMS (Figure 8, to the right of the image).

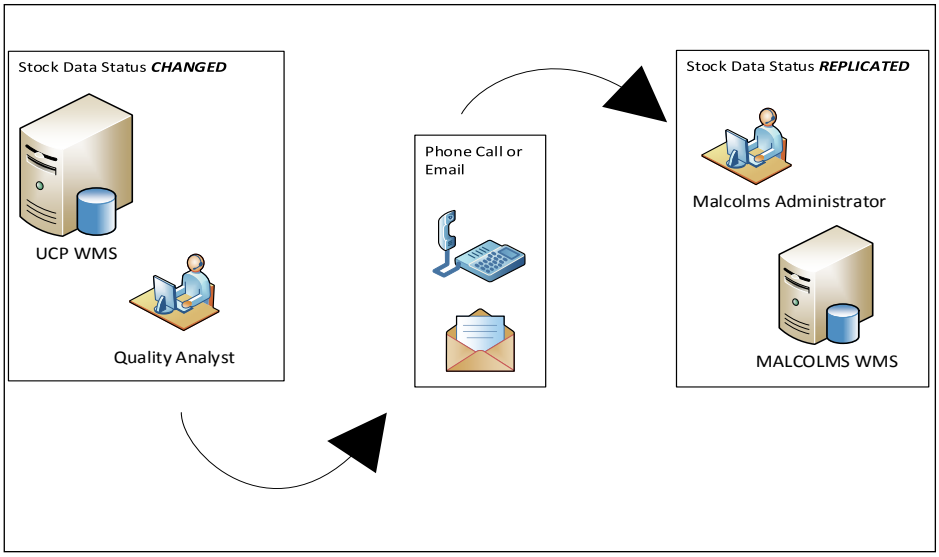


Figure 8: Problem Overview Status Update

As UCP is a 24/7 operation, but Malcolms administrators available from Monday to Friday 9-5, if stock status is changed outside of Malcolms office hours, there is a time window where blocked stock could be delivered by the customer. These manual administration activities are described step by step in Table 2.

Table 2 expands on this process by describing the specific steps performed in order to make this system to system update using telephone or email communication. Again, a set of activities are described along with effort (time) and the actors within the business who perform them.

Scope of Activities: Changing the stock status property on a stock record with a warehouse location. ***Purpose of Activities:*** If stock has been identified as defective at the warehouse, the quality control department at UCP change the stock record's status to ***BLOCKED.*** A blocked stock cannot be delivered to the customer and therefore a Malcolms administrator must replicate this change in the Malcolms WMS software. ***Frequency:*** Variable – depends on manufacturing faults / quality complaints.	
Activities	***Actor***
Quality department identify some stock that is defective	Quality Auditor
Stock is identified in UCP database along with any related stock ***Effort:*** Negligible	Quality Auditor
Stock is blocked in the UCP WMS database. ***Effort:*** Negligible	Quality Auditor
Stock Id's are emailed or phoned through to Malcolms Operator ***Effort:*** 2 minutes	Quality Auditor
Malcolms Administrator changes the stock record's status to "blocked" in the Malcolms Database. ***Effort:*** Negligible	Malcolms Administrator
Written confirmation is provided to the originator via email. ***Effort:*** 2 minutes	Malcolms Administrator

Table 2 - Manual Administration - Stock Status Updates

Time spent processing the above activities is relatively low. However, the major problem that isn't initially obvious is the communication link between the Quality Auditor at UCP speaking to the Malcolms Administrator to replicate the status update at the Warehouse. UCP stock status records can be updated 24/7, as quality checks are made at any time of day. However, the Malcolms administration staff are only available Monday to Friday, 9-5. Therefore, stock status changes that are transacted "out of hours" will not be replicated at the warehouse until some considerable time later, when the administrator picks up his email.

This time window represents a considerable risk, in some cases resulting in business-critical impacts such as defective stock being delivered to a customer.

2.1.10 Problem 3: Delivery FROM Warehouse

Finally, when stock is delivered from the warehouse to the customer, it leaves the Malcolms WMS database. In contrast, UCP stock records must be retained indefinitely, so stock delivered from the warehouse must have its record subsequently updated with a new location.

Figure 9 shows this problem overview, starting with a situation of stock being delivered in the Malcolms WMS (Figure 9, to the right). A Malcolms administrator (Figure 9, centre) makes the link to the UCP WMS by updating the stock records that have been delivered in the UCP WMS Database (Figure 9, to the left).

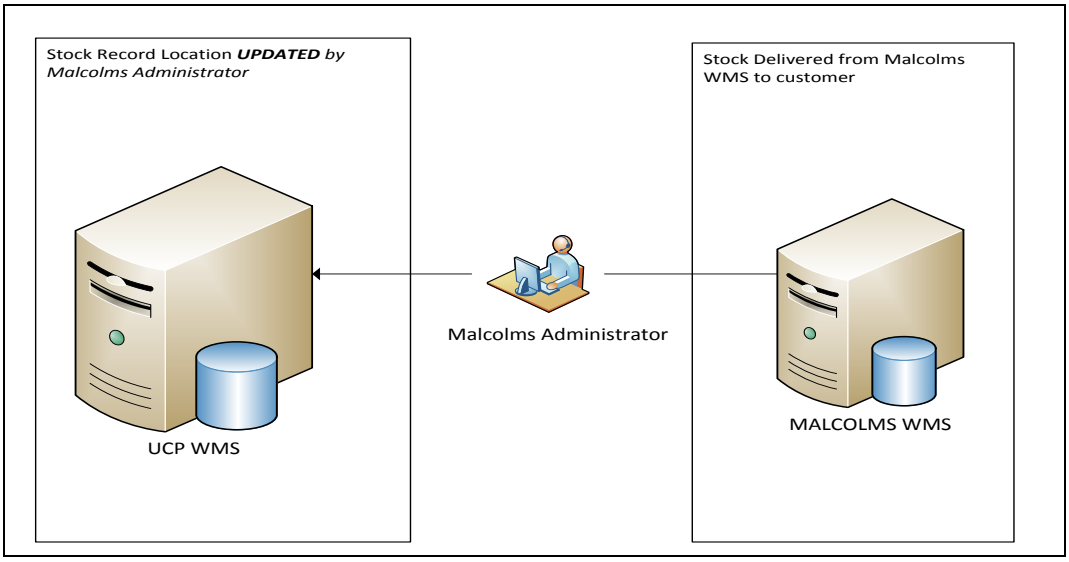


Figure 9: Problem Overview Delivery FROM Warehouse

The manual link provided by the Malcolms Administrator is made up of the following sets of activities described in Table 3. Again, this provides a description of activities (Table 3, Activity Column) and the staff members who perform them (Table 3, Actor column).

Scope of Activities: Warehouse Stock is delivered from the Malcolms Warehouse to another location ***Purpose of Activities:*** Remove the stock data from the Malcolms database (no longer exists at Malcolms) and update the stock location in the UCP database. ***Frequency:*** 5 – 10 deliveries per day. 40 stock items per delivery.	
Activity	***Actor***
Delivered stock transacted on Malcolms WMS	Malcolms Fork Truck Driver
Delivered stock updated on Malcolms WMS ***Effort:*** instant – system generated.	System Generated
Report of stock data delivered created ***Effort:*** negligible	Malcolms Administrator
Stock data location updated in UCP WMS ***Effort:*** 5 minutes	Malcolms Administrator

Table 3 - Manual Administration - Delivery FROM Warehouse

These activities provide a relatively low cost in terms of effort (time and business impacts), but they still represent a manual data update effort that exposes the WMS to data entry errors, and must therefore be eliminated.

2.1.11 Problems Summarised

The above processes describe the human interface efforts required to maintain stock consistency across both WMS implementations. Excessive human resource is invested in this process, and if procedures are always followed the problems are less apparent. However, it is inevitable that errors will occur with such manual administration tasks, with some causing business critical issues. A subsequent problem of correcting erroneous data is also a burden on the business. The costs of this retrospective "cleaning" are hard to quantify, but the problems show there must be significant incidence.

Some of the processes inherently create a major business risk, such as the Status Update (2.1.9). The Delivery to Warehouse process (2.1.8), due to a lack of an existing function, does not pose such a commercial risk but is extremely laborious involving excessive data duplication. A huge effort to transfer data that already exists in one system, to another, and capturing some reporting requirements. Finally, the Delivery From Warehouse process (2.1.10) presents lesser concerns from an effort and risk perspective, nevertheless data entry errors can still be expected.

2.2 Exploring Existing Software Solutions

The synchronisation of data across the UCP and Malcolms WMS implementations form a major discussion point in the research being addressed.

As a pre-requisite to introducing a new bespoke application to address this problem, a review of existing tools that achieve similar objectives are introduced in 2.2. Following on from this, examples of their implementations in the industry is provided in section 2.3.

Generically referred to as "Middleware", these software types essentially function as a hidden translation layer, enabling communication and data management for distributed applications [8].

Options exists in both open source and commercial format, all performing a huge array of different functions but non-the-less fall into the middleware bracket as described. The main commercial vendors are IBM, BEA Systems, Oracle, Sun Microsystems, and Microsoft ***[9]***. The major vendors originally provided solutions that catered for specific message exchange mechanisms, such as message queuing, REST, SOAP, and other application specific protocols. As requirements grew flat and wide, a need emerged for a consolidated solution covering all messaging and protocol standards. Vendors have addressed this based on the Enterprise Service Bus (ESB) concept. ESB based solutions allow applications to communicate using the protocols they support, with a translation and message routing facility mapped to associated endpoints [47]. This provides a consolidated "integration" framework that can be customised to suit the business need through code or built in codeless configuration tools. Examples of such tools are summarised below.

2.2.1 BizTalk Server (Microsoft Commercial Solution)

Microsoft's primary middleware platform is the BizTalk application. Designed as an integrated development environment combined with HTTP and SMTP as delivery mechanisms. Standard adapters exist that allow documents to be sent and received from Biztalk in XML, EDI, or flat file formats. A BizTalk implementation is discussed further in the industrial example provided in 2.3.

2.2.2 Websphere (IBM Commercial ESB Middleware Solution)

Introduced in 1998, IBM's integration platform is comprised of a central web application server that includes native support for Java Message Service (JSM), Message Queuing, and SOAP API's. Additional adapters required for interfaces with alternative products or legacy protocols are supported. The application includes an IBM HTTP platform that can provide additional web based services, such as online ordering systems into the solution. An example of this implemented in the industry is provided in 2.3.2.

2.2.3 OpenText (Commercial EDI Solution)

OpenText provide end to end electronic data interchange services, a global standard in computer to computer communications. Offering bespoke packages comprising of a range of industries and situations, OpenText enable a customer to use their services to support internal software development by providing EDI translation services. An example of the service provided is described in 2.3.3.

2.2.4 Mule (Open Source ESB Solution)

Mule provide an open source ESB solution that is supported by a large online community. A vast array of API's are natively supporting covering SOAP, REST, JMS, MQ, JBI, JBA, System I/O plus many more, with developments ongoing. As an open source application, the application can be extended referencing existing documentation and requests passed to the community for consideration. Furthermore, less complex requirements can be created using a "codeless" approach, making basic translation functions simple to implement, considering so many protocols are internally supported.

Running as a silent background process, waiting for traffic to pass it will write associated logs to a file. Internal transformers translate incoming messages into "Mule" objects. These mule objects provide the translation mechanisms and route the translated message accordingly.

2.3 Exploring Existing Industrial Solutions

2.3.1 Yantra WMS Integration with External Warehouse using Microsoft BizTalk 2004

Aspire Systems, a software development firm, have published a case study [46] on their WMS to WMS integration solution developed for their client, an e-distributor of chemical products.

The client demanded a solution to address challenges within the business in relation to the synchronisation of data between multiple disparate business systems, referring to excessive manual overheads supporting existing processes, drawing similarities to the problem discussed in this paper.

The scope of work covered a recently acquired Warehouse Management System, Yantra, and the integration work involved providing electronic interfaces to link the Yantra WMS to their existing ERP, CRM, E-Commerce suite, and an external third party Warehouse system. Thus, the project encompasses the integration of many local and remote software implementations alongside the provision of a WMS to WMS interface.

Aspire provisioned a solution based on the BizTalk platform, a Microsoft "Inter-Organizational Middleware System" (IOMS) that allows companies to implement communication channels between applications using a range of adapters and API's tailored to several enterprise level software systems out of the box. The BizTalk communication channels can be customised using the Visual Studio IDE allowing developers to customise message exchanges using standard formats such as EDI standards and XML documents.

The client's requirement included data exchange mechanisms between their "GreatPlains" ERP system (Now Microsoft Dynamics) which is supported by BizTalk native adapters, therefore the solution was well placed in this instance. For the Yantra Warehouse Management System integration with an external warehouse, bespoke data exchanges were included that translated Yantra XML message files into EDI formats demanded by the third party Warehouse. The Biztalk solution is shown in Figure 10 (on page 23), with the Client WMS (1) and External WMS (2) marked. The BizTalk platform manages the file exchange between both using XML data file messaging, translated within the BizTalk framework into EDI formats that are aligned to the external warehouse requirements. In addition, the BizTalk platform provides native adapters and API's for managing cross functional interfaces for ERP and alternative business applications providing an integrated message exchange framework for a relatively complex requirement.

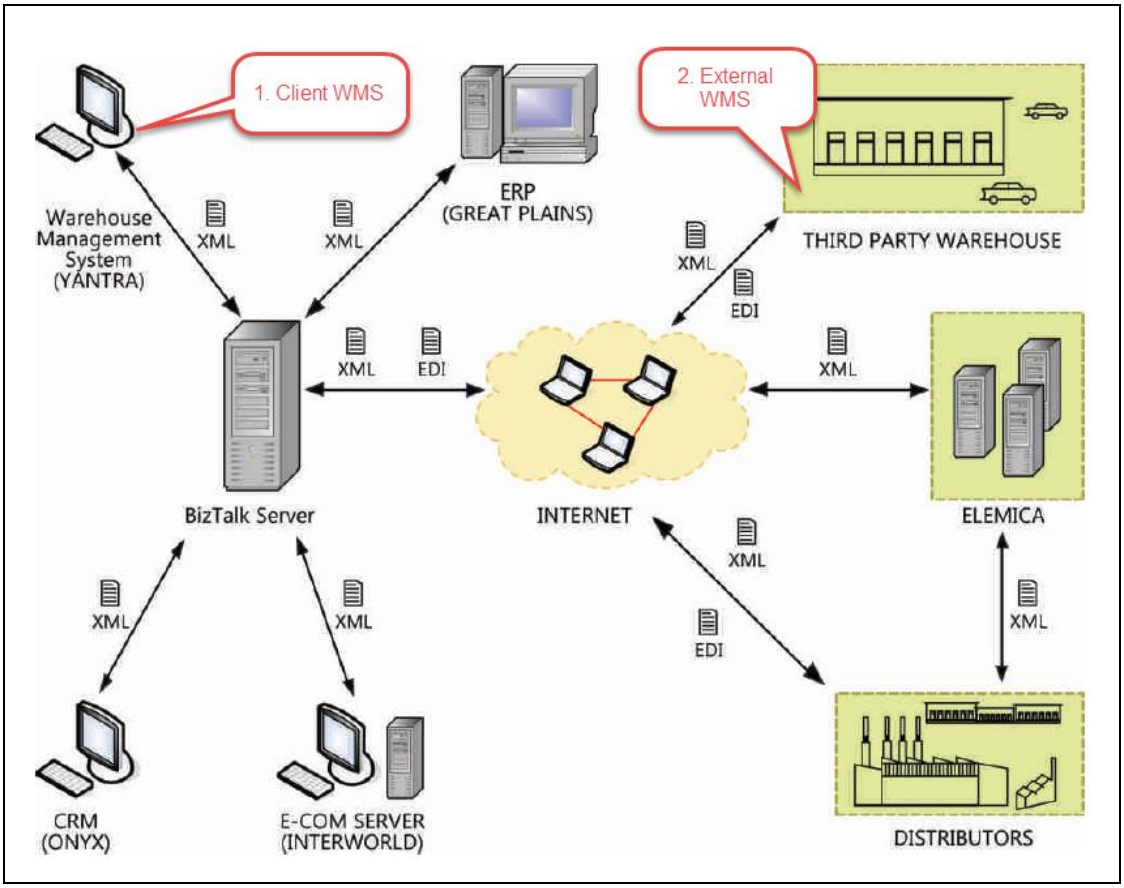


Figure 10: Yantra WMS Integration using BizTalk [46]

2.3.2 Whirlpool SAP ERP Integration

Whirlpool manufacture household kitchen products, a well-known washing machine brand. Their business use a SAP ERP system as their primary application. A significant manual administration effort existed in relation to Whirlpool's "middle tier" distribution channels. Smaller companies used to extend the firm's global outreach. These smaller distributors contributed to upwards of 10% of Whirlpool's revenue stream, but the process of placing orders with these distributors' involved excessive phone calls and emails.

IBM Websphere server was introduced to address this need. Providing web services as part of the package, a consolidated HTTP ordering system was implemented which consolidates the manual ordering effort. The Websphere application is configured to translate the inputs into the relevant communication protocols with the subsidiary businesses automating the ordering process entirely.

2.3.3 ON Semiconductors Business to Business EDI integration

OpenText are a major global player in the field of Electronic Data Interchange solutions, and an example of the services offered to a manufacturing firm can be taken from the solution provided to their manufacturing client ON Semiconductors [26].

Electronic Data Interchange (EDI) is the "computer to computer" exchange of business documents in a format that conforms to a globally recognised syntax. EDI intends to replace human intervention entirely by providing a computer-to-computer data exchange that is completely automated [13].

The company's supply base is vast, numbering in the hundreds and using a plethora of technologies. OpenText as the EDI partner now supply the entire EDI solution that sits between all suppliers and their business applications. The solution not only passes data but monitors for errors and reports exceptions to key business users. Such a large-scale EDI deployment is entirely managed by the outsourced services provided by OpenText and ensure a holistic data exchange solution can be achieved in a complex business environment.

2.4 Project Management

Prior to taking on the business development project, research has been performed around project management, paying attention to proven and innovative methods in approaching the software development lifecycle. The UCP business has communicated prior struggles with more traditional software development approaches, so an alternative method has been presented.

2.4.1 Agile – Software Development Life Cycle

Agile is a project management methodology. Although the methodology has its origins in software development, the underlying principles can be applicable to a spectrum of project scenarios. [39]

On a generic level, the fundamental principles of Agile are that it breaks down projects into smaller iterations that are more manageable. Each iteration is based on business needs or functions, and the result of the work (software development or otherwise) generate outputs based on real user and business requirements for demonstration and feedback.

In the software development world, Agile sets out the following core principles [41].

- **Diligent Backlog Refinement through User Stories**

Problems and requirements are rolled into "user stories". These user stories make up the iterative approach to development, and contain properties within themselves that provide a sustainable and flexible development life cycle. User stories are prioritised and budgeted regularly. They form the fundamental control mechanisms for all phases of design, delivery, and acceptance. (Robert and Micah, 2007) summarise the Agile User Story as follows: *"a mnemonic token of an ongoing conversation about a requirement. A user story is a planning tool that the customer uses to schedule the implementation of a requirement, based on its priority and estimated cost."* [42]

- **Whole Team Approach**

Customers, managers, developers, designers are all involved in the user story definition. Strict rules surround the team so far as the business define the requirements and priority of each user story, and the developers budget the effort without any crossover. It is up to the designated project manager to organise the sequence based on this business impact versus budgeted effort trade off. [42]

- **Flexibility**

User stories contain a list of top level functional requirements, often called tasks. These are deliberately kept as generic statements, and revised throughout the sprint / iteration development phase. As the product comes to fruition, developers and analysts will have greater clarity on the situation and can make more informed decisions or adjustments as the development progresses. This removes the problems of fixed objectives and disagreements about specifics in terminology or concepts, and should ensure a more organic evolution of the solution.

- **Test Driven Development**

A key component of the Agile approach. TDD is the concept of performing Unit Testing functions at each stage of the development and change process. For all business logic operations, a set of unit tests must be determined to maintain a state of consistency throughout development.

Extreme Programming, a specific variant of Agile in Software Development, takes this a step further and suggests that all development should stem from tests. The idea being that a test is created, written in the way the developer expects the function to look (which will fail, in absence of actual method) they then builds the method based on the test, to make it work. The solution is developed in this way forward, refactoring and adding functions iteratively.

The suggestion is that this is not only a way to comprehensively test all functions, it is a structured approach to software building that also "documents itself" through unit testing [43].

2.4.2 Agile Versus Traditional Development Lifecycle

Unlike the more traditional project development methods, Agile co-ordinates all key elements and inputs across all project members from designers, developers, and users throughout. In contrast, the more traditional Waterfall approach strictly follows a "gated" sequence: Research, design, specification, develop, and deploy for the entire solution. This inevitably results in a breakdown of communications, as a business user is likely to be unable to identify a potential flaw in the application business logic before the software build has even begun [16][18][22].

Throughout the design and development phases we have therefore chosen to use the Agile methodology which takes a flexible approach to software design.

3 Requirements

Prior to design, a set of business requirements have been gathered. This exercise will drive critical design decisions described in chapter *4 Design* (on page 39).

Traditionally, the waterfall approach to software development would usually incorporate the outputs of a requirements gathering exercise into a formal specification document. This would describe, technically and functionally, the working of the software. This ensures technical documentation is created in advance and can be shared among developers and users alike. However, this means business users are unable to review functionality until after the specification has been written. Therefore, refactoring is long winded, involving adjustments to both functions and specifications for each change.

As described in 2.3 Project Management, *Agile – Software Development Life Cycle* (on page 24) a different approach has been taken. The output of the Agile approach is significantly different, allowing greater flexibility in the design phase. This is one of the core reasons for drawing inspiration from this approach during this project. Rather than defining a technical specification as the output of the requirements gathering process, the Agile approach groups the requirements into functionally and operationally similar development iterations. These iterations are designed, tested and refactored simultaneously. This should ensure software design involves all business stakeholders as well as developers, during these phases. Therefore, the requirements gathered in an Agile project are to generate a flexible *design concept* known as a User Story (discussed later) rather than a written specification.

Considering these points, the requirements must have a business focus, with the technical needs married to the business needs as part of the Design phase in the following chapter.

3.1 Approach: Gathering Requirements

3.1.1 Assemble critical business users (section 3.2)

A group of critical business users and stakeholders have been assembled to define clear roles and responsibilities that will support the direction in terms of gathering requirements, setting priorities, and providing business direction where needed. Some underlying principles have been observed as part of this process:

- ✓ Include at least one ***Super User*** in the group, familiar with hands on use of the existing software and tools. This user must provide inputs that relate to the details and nuances of the day-to-day use of existing systems, an area where a manager may be less informed.

✓ Include at least one supervisor or line manager in the project team to support decision making and overall direction.

3.1.2 Consolidate issues into Problem Categories (Section 3.3)

Some stakeholders are unfamiliar with concepts and detail outside of their day job. These categories add context and clarity to the problems being addressed and provide supervisors and managers with a holistic overview of the problem being addressed, supporting the requirements gathering process.

3.1.3 Create a list of requirements for each Problem Category (Sections 3.4 & 3.5)

Finally, a list of requirements are documented. This project has a primary focus on replacing manual data inputs with automated data exchanges, so scope is somewhat restricted. However, there are several additional needs within this scope that are important to the business. Key points:

✓ Each requirement must be non-technical and address the issues in the problem brief, either directly or indirectly.

✓ Each requirement must fully articulate the business impact to drive project prioritisation.

3.2 Critical Business Users

The follow user group has been assembled to support the decision making and requirements gathering process. Assessment of the existing problem raised some concerns about a lack of clarity in this area, so this should be well defined. The role definitions are shown in Table 4 with commentary on their responsibility. Table 5 introduces the team members and their role.

Table 4 Critical Business Users: Roles and Responsibilities

ROLE	RESPONSIBILITIES
Super User (SU)	A user with hands on experience of the day to day operation, software and tools. Excellent understanding of daily issues faced, how to overcome them, and the nuances of existing work practices. The Super User must be able to provide input during the requirements gathering, making sense of problems, and testing new processes. Inputs from Super Users may be either valuable or distracting, therefore although inputs should be invited, they must be considered. Super Users often have an excellent understanding of their own specific jobs, but lack clarity on top level process.
Stakeholder (ST)	A user, typically a line manager, supervisor, or director in a specific area relevant to the project or its success. Will be involved in defining overall project goals and direction. Not essential to be involved in day to day project work, if input is not distracting. Must ensure stakeholders do not interfere with the work of a Super User, who is often of a lower "rank" in the organisation, who must provide valuable operational experience and feedback.

Table 5 Critical Business Users: Members

NAME	JOB TITLE	PROJECT ROLE	COMPANY
Ian McLaren	Operations Director	ST	UCP
David Robertson	Operations Manager	ST	Malcolms
Andy Dawson	Planning Manager	ST, SU	UCP
Robert Dawson	UCP Administrator	SU	Malcolms
Edward Gillespie	Stock Administrator	SU	UCP

3.3 Problem Categories – User Story Definition

User Stories represent abstract project management references in an Agile software development lifecycle. Each User Story group logically and functionally related development iterations of similarly themed business functions. (See 2.4.1 *Agile – Software Development Life Cycle* on page 24). These will be introduced during the requirements gathering process, to add clarity and scope to what may seem like a plethora of topics.

Prior to the requirements gathering exercise, all team members were provided a detailed summary of the problem being addressed in the UCP business (see 2.1). This discussion covered many processes, issues, and concepts. The issues described were "flat and wide" in their nature therefore proved a difficult starting point. Addressing this, a summary of the logically related issues have been identified and form the basis of our development work going forward. With regards the requirements gathering process, switching the focus from the numerous "issues" to the consolidated set of "root causes" articulated in the User Story definition, helped support requirement gathering significantly.

3.4 User Story 1: Deliver to Warehouse

3.4.1 Situation

The delivery of stock from the UCP factory to the Malcolms warehouse. The background to the problem described in 2.1.8 (*Problem 1: Delivery to Warehouse* on page 14).

3.4.2 Scope of Requirements

- ✓ Stock records exists in the UCP WMS software, and they are destined for delivery to Malcolms.
- ✓ Stock records do not exist in the Malcolms WMS software, but must be created in this software when a delivery is performed by UCP.

3.4.3 Requirement Goals

Remove the paper based manual administration required to link the UCP and Malcolms WMS systems when stock is transferred from UCP to Malcolms on vehicle transport. Replace the form-filling process currently used with an electronic "Deliver to Warehouse" function reducing the excessive administrative overhead.

3.4.4 Requirement List

Table 6 and Table 7 consolidate the requirements, each containing a description of the requirement itself, and the priority assigned. The requirement description includes commentary on justification, in relation to the scope of the original problem.

Table 6 - Despatch to Warehouse Requirements (Part 1)

#	DESCRIPTION OF REQUIREMENT	PRIORITY
1	**Requirement:** Create a delivery GUI which allows the operator to perform a new "Delivery to Malcolms" function **Additional Notes:** This function will form the basis of the remaining requirements. **Justification:** The GUI will replace form filling interface for the operator, and allow automatic retrieval of WMS stock data that was previously recorded manually.	High
2	**Requirement:** For the stock being delivered, automatically retrieve stock records from the UCP WMS **Additional Notes:** Stock record data already exists in the UCP WMS database at the time of a delivery. **Justification:** Remove the need for a delivery operator to collect duplicate stock labels (subsequently used for data input into the Malcolms WMS)	High
3	**Requirement:** For stock that has arrived on a delivery, automatically import stock records into the Malcolms' WMS **Technical note:** The scope of this requirement is important to understand. The process of importing stock data into the Malcolms' WMS electronically must be implemented by the Malcolms IT team. This project and development will address the development of the facility to pass the data via a message exchange mechanism to the Malcolms WMS, in a format that works for both businesses. **Justification:** Related to the requirement above, stock labels are used as a reference to manually input data into the Malcolms WMS.	High

Table 7 Requirements List (Part 2)

#	DESCRIPTION OF REQUIREMENT	PRIORITY
4	**Requirement:** Delivery reporting: For every stock sent from UCP to Malcolms, capture the user ID of the operator that loaded it onto the vehicle without secondary operator inputs. **Additional Notes:** All UCP operational staff log on to the UCP WMS with their own user ID, so this data exists in the UCP WMS. **Justification:** Specific management traceability requirement.	Medium
5	**Requirement:** Historical reporting capability on all deliveries from the factory to the warehouse. **Additional Notes:** Delivery vehicle location, date, time and the stock records of each delivery must be reported. **Justification:** A problem with the paper record delivery logs are the difficulty in retrieval. An electronic solution must provide this detail quickly and easily.	Low
6	**Requirement:** Email alert sent to the Warehouse at point of despatch. **Additional Notes:** Described as a "nice to have" by the Malcolms super user. Not essential, as the overall requirement is to handle the stock data transfer from UCP to Malcolms WMS automatically. However, this means a super user will lose visibility of the stock data that their operators are receiving, as the paperwork will no longer be passed to them. **Justification:** The use of an electronic delivery process will mean Malcolms super users do not get a summary of the stock being unloaded, which was previously filled on the paper despatch form. Email alert requested but not described as essential.	Low
7	**Requirement:** Reduce the likelihood of operators forgetting to load pallets onto a vehicle in the UCP WMS. In the new delivery process, the operators must have to validate the stock records being sent. **Additional Notes:** As a minimum requirement, it is requested that the operators confirm the total stock records on the vehicle before delivering. Ideally, a full list of all lorry contents and properties will be listed. **Justification:** The paper delivery record and duplicate labels often contain errors.	Medium

3.4.5 Malcolms Delivery Message: Specification

As a critical output need, the specification of the delivery message is provided. The generation of this message will facilitate the automatic import of stock records into the Malcolms WMS. The specification provided is comprised of a header (representing the delivery) and detail (representing the stock record contents of the delivery) and is shown below.

Table 8 describes the layout of the delivery message header file. This is of a .CSV file type, made up of three columns. The column definitions are described in the "Data" column. One message header must be provided for every delivery.

Table 9 describes the layout of the delivery message detail file. This is of a .CSV file type, made up of ten columns. The column definitions are described in the "Data" column. One message header must be provided for every delivery, containing one row of data as specified for every stock record on the delivery. Column 2 in Table 9 specifies the join to its associated delivery header file.

Table 8 – Delivery Message File Specification (Header)

COLUMN	DATA
1	**ID** – A unique identifier for each delivery sent
2	**Load Reference** – The delivery vehicle location in the UCP WMS
3	**Despatch Time** – The date and time the vehicle was sent
4	**Flag –** Set to "1" to indicate delivery started

Table 9 – Delivery Message File Specification (Detail)

COLUMN	DATA
1	**EntryID** – A unique identifier for each stock record delivered
2	**ShuttleHeaderId** – The delivery ID (Column 1 in Table 8)
3	**SSCC** – The unique serial shipping container code [1] (aka "load carrier")
4	**Manufactured Time** - The time the stock was manufactured
5	**Loaded Time** – The time the stock was loaded onto the vehicle
6	**Operator Id** – The ID of the user who loaded the pallet onto the vehicle
7	**Item Number** – The stock item number (aka "product code")
8	**Quantity** – Quantity in base UOM

9	**Box Quantity** – Quantity in boxes
10	**Status** – Stock record "status" property
11	**Machine** – The machine the stock was made on
12	**Batch –** The production batch number

3.5 User Story 2: Status Update

3.5.1 Situation

The process of updating a stock record's "status" property in the UCP WMS, and replicating the change in the Malcolms WMS. Background to the specific problem is provided in 2.1.9.

3.5.2 Scope of requirements

✓ Stock data records that exist in both the UCP and Malcolms WMS

✓ When a Quality Auditor at UCP identifies defective product, the stock record is updated in the UCP WMS using a native function within the software. This change must be communicated manually via telephone or email to the Malcolms administrator to replicate the change using an equivalent function in the Malcolms WMS.

✓ UCP operate a 24/7 operation, and can identify defective stock at any time. Malcolms Administration is Monday to Friday 9-5. This leaves a significant time window where stock record status may be incorrect in the Malcolms WMS

✓ This exposes a major business risk, as the associated stock record(s) could be delivered to a customer during this "time window" and potentially generating a customer complaint.

3.5.3 Requirement Goals

Remove the need for a Quality Auditor to phone or email the Malcolms administrator to replicate a Status Update in the Malcolms WMS, removing the time delay risk and administrative effort.

3.5.4 Requirements List

The requirement list for this User Story is limited to a specific business need and requirement; automation of a status update across both systems, triggered by a change made by a UCP Quality Auditor. In contrast to the ***Deliver to Warehouse*** requirements in section 3.4 which requires a brand new deliv-

ery function, the status update is a native function in both WMS systems and therefore the requirements surround a straight data exchange between the two WMS systems.

Table 10: User Story 2 Requirements (Status Update)

#	DESCRIPTION OF REQUIREMENT	PRIORITY
1	**Requirement:** Automatically trigger a status update in the Malcolms WMS as it is changed by a quality auditor at UCP **Additional Notes:** This function will form the basis of the remaining requirements. **Justification:** A native function in the UCP WMS, a process must be defined to trigger the equivalent change in the Malcolms WMS.	High
2	**Requirement***:* Create a trigger *mechanism* from the UCP WMS **Additional Notes:** As a native function in the UCP WMS, such transactions are captured in the software. A method for leveraging this change must be created. **Justification:** To exchange data updates across systems, the application must be able to identify the transactions that need to be exchanged, and those that do not.	High
3	**Requirement:** Define a *Status Update* message file specification, for import into the Malcolms WMS. **Additional Notes:** Once a trigger mechanism is devised in the UCP WMS, the data will need to be imported into the Malcolms WMS. A message file specification will need to be provided by Malcolms that will facilitate this process. **Justification:** This allows the data "sent" from the UCP WMS, to be applied to the Malcolms WMS. See 3.5.5 below for the message file specification.	High

3.5.5 Malcolms Status Update Message: Specification

The Status Update message specification is summarised in Table 11, which describes the layout of the delivery message detail file. This is of a .CSV file type, made up of seven columns. The column definitions are described in the "Data" column. Provisioning the message files facilitates immediate the immediate change of a stock status on the associated record in the Malcolms WMS. One status update message file can contain one or more status update records, but limited to only one record update per line.

Table 11 Status update message file specification

COLUMN	DATA
1	**ID** – An identifier representing a unique stock status update message
2	**Transaction Time** – The time when the transaction was made in the factory WMS
3	**SSCC** – Serial Shipping Container Code – Synonymous with the stock record's "load carrier" property
4	**Item Number** – The stock record's product property
5	**Status** – The New / Latest Stock Status, either BLOCKED or AVAILABLE
6	**Status From** – The previous stock status, see column a 5
7	**Status To** – The changed status

3.6 User Story 3: Warehouse Update

3.6.1 Situation

The process of either receiving or delivering stock from the Malcolms WMS. Background to the specific problem is provided in 2.1.10.

3.6.2 Scope of requirements

✓ Stock data records that exist in both the UCP and Malcolms WMS

✓ When a Malcolms delivery or receipt (receiving stock at the warehouse) is performed, operators at the Malcolms Warehouse transact this on the Malcolms mobile computers.

✓ These transactions need to be manually replicated in the UCP WMS by a Malcolms administrator, to synchronise the changes.

3.6.3 Requirement Goals

Remove the need for the Malcolms administrator to manually update the UCP WMS, when Malcolms make a delivery or receive stock.

3.6.4 Requirements List

The requirement list in Table 12 contains many similarities to those described in the previous user story (3.5.4 above, comprising a similar automated data exchange requirement, with source trigger and output destination (UCP and Malcolms) being reversed.

Table 12 Warehouse update requirements

#	DESCRIPTION OF REQUIREMENT	PRIORITY
1	**Requirement:** Automatically trigger a stock record location update in the UCP WMS when it is made in the Malcolms WMS. **Additional Notes:** This function will form the basis of the remaining requirements. **Justification:** A native function in the Malcolms WMS, a process must be defined to trigger the equivalent change in the Malcolms WMS.	High
2	**Requirement:** Create a trigger message file from the Malcolms WMS **Additional Notes:** With the Malcolms WMS being off limits in terms of direct database access, Malcolms have provided a message specification format for stock deliveries and receipts performed at the Warehouse. This message must form the basis of the trigger mechanism. **Justification:** To exchange deliver and receipt information form the Malcolms WMS, a data feed must be provided in the absence of direct database access.	High
3	**Requirement:** Update UCP WMS stock record location and transactions history **Additional Notes:** With the output of the data exchange being a stock record update in the UCP WMS, a stock record update must be performed pertaining to the message content received. **Justification:** Directly synchronises the UCP WMS stock record based on the Malcolms message received See 3.6.5 and 3.6.6for the message file provided by Malcolms.	High

3.6.5 Malcolms Stock Delivered Message: Specification

The Malcolms delivery message file contains records of all stock delivered. One file provided per delivery performed, containing one line for each stock record on the delivery. File format is CSV, and

the delivery message is named with “UCPORD” as the file prefix identifier, which identifies the file as being of a “delivery” message type. The eight columns provided in the Malcolms WMS message are shown in Table 13.

Table 13 Malcolms delivery message specification

COLUMN	DATA
1	**SSCC** - Serial Shipping Container Code – Synonymous with the stock record’s “load carrier” property.
2	**Item Number** - The stock record’s product property.
3	**Number of Boxes Delivered** – The quantity in boxes delivered.
4	**Default Number of Boxes on Stock** – The default number of boxes on a full stock record.
5	**Date** – Date and time the stock was delivered
6	**Document Number** – Delivery reference number, also known as “trip” number
7	**Delivered From** – Delivered from location, prefixed to “NEWHOUSE” which means “Malcolms”. Could change in the future if more Malcolms warehouses are used by UCP.
8	**Delivered To** – Destination delivery location

3.6.6 Malcolms Stock Received Message: Specification

The Malcolms receipt message file contains records of all stock received from a UCP delivery. One file will be provided per delivery received, containing one line for each stock record on the delivery. File format is CSV, and the delivery message is named with “UCPPA” as the file prefix identifier, which uniquely identifies the file as a “stock received” message type.

The four columns provided in the Malcolms WMS message are shown in Table 14 on page 38.

Table 14 Malcolms receipt message specification

COLUMN	DATA
1	**SSCC** - Serial Shipping Container Code – Synonymous with the stock record's "load carrier" property.
2	**Item Number** - The stock record's product property.
3	**Number of Boxes Received** – The quantity in boxes delivered.
4	**Date** – Date and time the stock was delivered

4 Design

This provides the design for each user story. Subsequent implementation is provided in the following chapter,

The software design is broken down into the three main user stories, with requirements and problems described previously.

User Story 1 is the design of a new ***Deliver to Warehouse*** function introduced to the existing system. The design addresses the problems in 2.1.8. The function must allow the delivery operator to interactively verify the contents of a delivery and automate the exchange of data records from UCP to Malcolms.

User Story 2 is the design of a ***Status Update*** message handling and exchange application, addressing the problems outlined in 2.1.9. This application automates a Stock Record's status property in the Malcolms WMS when it is changed in the UCP WMS.

User Story 3 is the design of a ***Warehouse Update*** message handling and exchange application, addressing the problems outlined in 2.1.10. The application automates the updating of the UCP WMS' stock record location when stock is delivered or received in the Malcolms WMS.

All user stories are underpinned by a new SQL database implementation which stores the "WMS to WMS" transaction history.

4.1 User Story 1: Delivery to Warehouse

The design is presented as a series of design Components (4.1.1) which describe the application structure. These are translated into a series of Classes (4.1.9) in the form of a UML class diagram. Further discussion is provided by way of a Workflow (4.1.10) diagram.

To introduce these parts, a summary of technical requirements are provided, driven by the overall business requirements.

- ✓ Exchange stock records pertaining to each delivery seamlessly from the UCP to Malcolms WMS Databases
- ✓ Retrieve stock records from the UCP WMS as the critical input, with minimum user inputs.
- ✓ Send stock records via a .CSV message file as the critical output, which will facilitate the needs of an import tool used by Malcolms
- ✓ Present a user interface for the operator to perform delivery activities in the application
- ✓ Prompt the delivery operator to provide positive validation of the stock data he is despatching
- ✓ Present the stock record details to the delivery operator to view and cross check against their delivery
- ✓ Has a user interface that is simple and user friendly for staff with limited IT literacy

4.1.1 Design Components: Overview

Design encompasses two distinct application layers that overlay business logic; A Data Layer and a Presentation (graphical user interface) Layer in accordance with the requirements in section 3.4.

Binding these functions is the addition of a new SQL database (referred to as the *SQL Integrator* Database), which acts as the "glue" between the UCP and Malcolms WMS implementations.

A summary of the design concept is presented visually in Figure 11, with supporting commentary on each component thereafter.

Figure 11 shows the UCP WMS to the left, and the Malcolms to the right. The application components described in the rest of this chapter are presented as an abstract set of design ideas that pass the data exchange needs through the SQL Integrator Database, from the UCP WMS, to the Malcolms WMS in the form of a message file.

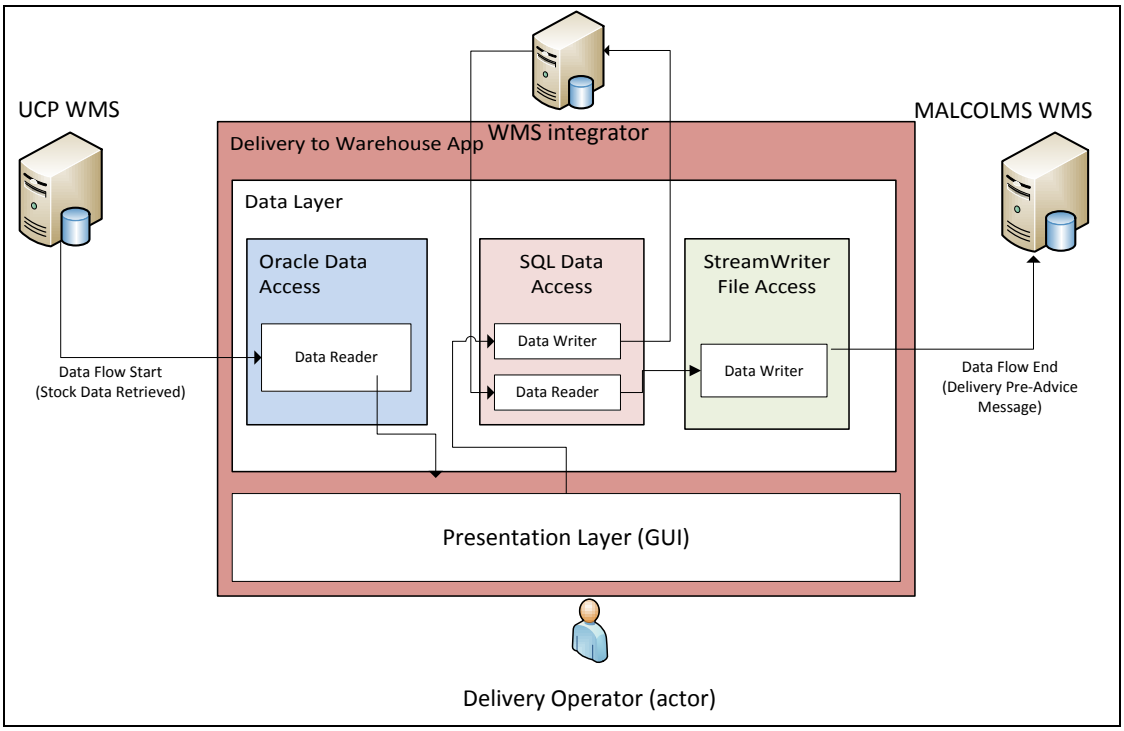


Figure 11: Design Overview (User Story 1)

The Delivery to Warehouse application (Figure 11, red box) contains the following components designed to replaces the manual activities that previously linked each WMS during a delivery. The figure shows the plan to retrieve delivery data automatically (Figure 11 start, marked "UCP WMS") and pass to a delivery message file for import into Malcolms WMS (Figure 11 end – marked "Malcolms WMS"). This satisfies our ultimate objective of a seamless data exchange from UCP to Malcolms WMS. The data flow route is shown by the linear arrow starting from the UCP WMS passing through the layers described and their individual components. A Presentation Layer (Figure 11, "Presentation Layer") contains the GUI that is needed for an operator to provide the critical validation checks demanded in the requirements.

A series of data connections within the Data Layer (Figure 11, marked "Data Layer") handle the retrieval, parsing, and processing of the data prior to the stock records being packaged as a message file for output the Malcolms WMS, which completes the delivery use case.

The Oracle Data Access component (described further in 0 below) is responsible for retrieving the relevant stock records into memory. This requirement satisfies the data capture need previously manually input into a spreadsheet and copied from stock labels. The Oracle Data Access components available for Visual Studio 2015 C#.NET are a critical library used.

The Presentation Layer (described further in 4.1.5 below) visualises the stock records retrieved from the native UCP WMS. In the event an operator notices an error; the process must be aborted and resolved. This provides a critical component of the requirements – user validation of delivery data.

Upon positive confirmation, the Sql Data Access component (described further in 4.1.2 below) writes the delivery data into a new WMS Integrator database. This provides a data store of deliveries performed for reporting, and the source repository for message files passed to Malcolms. The intermediary database underpins the entire data exchange. An SQL server license will be used as instructed by the company and implemented using virtual server storage space provided.

Finally, the StreamWriter File Access component (described further in 4.1.4 below) packages the stock records in the WMS Integrator and outputs in the message format specified by the Malcolms application support team. The workflow steps associated to this process are described further in the subsequent section 4.1.10.

This concept described has been visualised in

Figure 12 below, with the addition of descriptive route steps; (1) *user inputting lorry reference*, (2) *retrieving stock from UCP WMS*, (3) *writing delivery to Integrator Database*, and (4) *passing a message to Malcolms*.

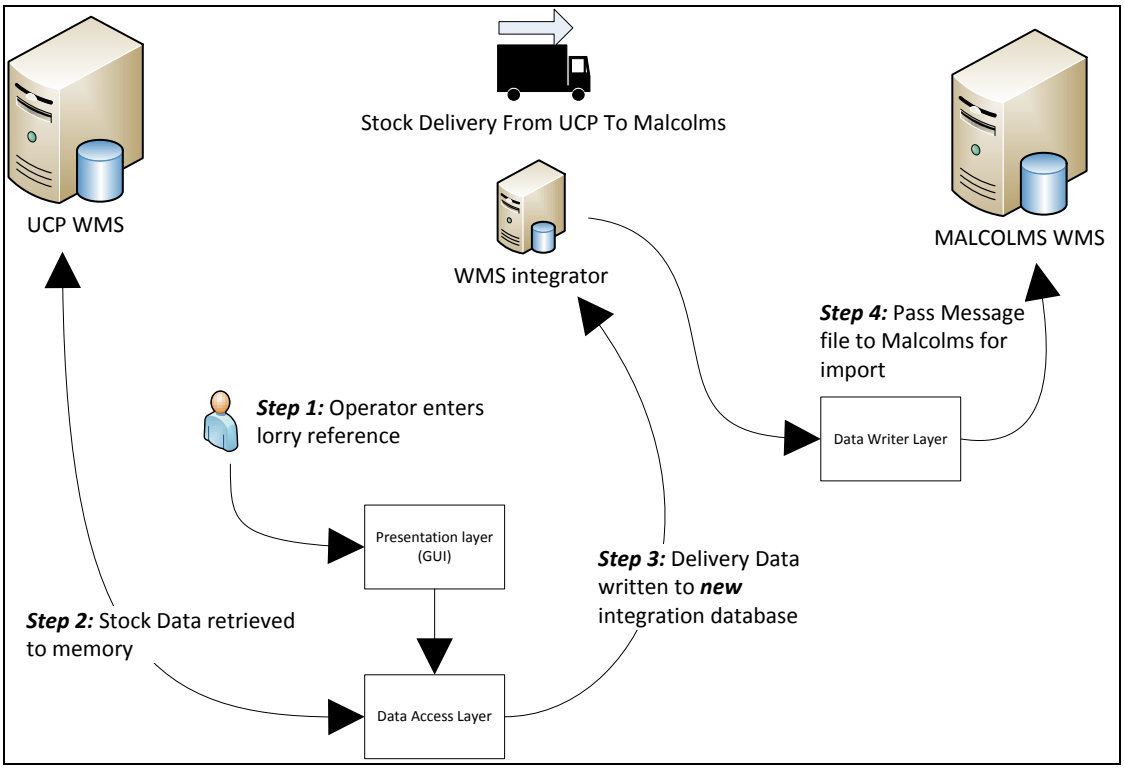


Figure 12: Design Overview (User Story 1) – Use Case Scenario

4.1.2 Design Components: SQL Data Access and Database Table Structure

The *WMS integrator database* introduced in 4.1.1 will be created to underpin the new delivery to warehouse function. This is required to provide historical reporting capabilities on delivery records and provide a repository for Malcolms delivery messages. From an abstract perspective, this database provides the link from a "standing data" perspective between the two WMS implementations.

The SQL Data Access component in the application utilises the Microsoft *System.Data.SqlClient* library to provide data exchange functions between the application and the new database.

This data table structure designed is quite a simple design concept, as it only needs to represent each delivery to the warehouse (a vehicle and its contents). To achieve third normal form, the delivery records table is designed over two tables (a vehicle table, and a vehicle contents table). This has been reflected in the ER diagram in Figure 13.

This shows a "Shuttle Header" and a "Shuttle Detail" table. The Shuttle Header represents the Lorry delivering the stock. The Shuttle Detail represents the contents of the vehicle, the critical stock data that it contains.

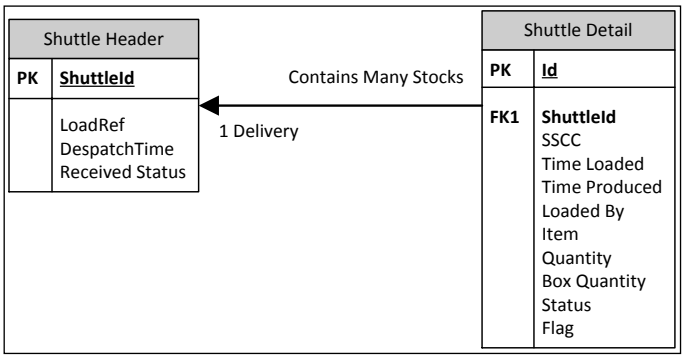


Figure 13: ER Diagram: WMS Integrator Database

Note: The term "shuttle" is a reference used by UCP to describe the process of a vehicle performing a delivery from the Factory to the Warehouse, thus: one vehicle + delivery of one or more stocks = shuttle. This is the reasoning behind the naming convention used.

A summary of columns in the ER diagram is provided below, Table 15 describing the columns and commentary on the Shuttle Header table, and Table 16 and Table 17 for the Shuttle Detail table.

Table 15 - Shuttle Header Table

A record for every delivery performed.

FIELD	PURPOSE
ShuttleId	Primary Key: A unique delivery identifier for every delivery made.
LoadRef	This identifies the physical lorry that the stock was delivered on. Taken from the native UCP WMS software.
Received Status	Determines if the delivery has been received by Malcolms. True = delivery complete, false = in transit. Not specified as a business requirement, but added in the design to support the need for "historical reporting". A manufactured column maintained in the SQL Integrator database only, to indicate the status of a delivery, which does not exist elsewhere.

Table 16 - Shuttle detail table

A record for every stock on a delivery.

FIELD	PURPOSE
ID	Primary Key: Uniquely identifies an individual stock on a specific delivery. A stock-delivery relationship. Database generated ID.
ShuttleID	The foreign key relation represented in the Shuttle Header table above.
SSCC	Serial Shipping Container Code. The stock reference used for all businesses in the supply chain (UCP and Malcolms). This field is required for both UCP reporting purposes, and the Malcolms delivery message output file. Taken from the native UCP WMS data.
Loaded By	The UCP user who loaded the stock onto the pallet. A requirement outlined by the business. Taken from the native UCP WMS data.

Table 17 Shuttle detail table (continued)

FIELD	PURPOSE
Item	Determines the type of stock. Required in the Malcolms delivery message output file. Taken from the native UCP WMS data.
Quantity	The stock record quantity, in the Unit of Measure "Each". UCP use a Unit of Measure by individual component. Taken from the raw UCP WMS data.
Box Quantity	The stock record quantity in the Unit of Measure "boxes" – will be either one or more. Malcolms use this Unit of Measure for the delivery message. Calculated value. Does not exist in the raw UCP WMS stock record. Using a calculation that incorporates another field value from the UCP WMS, "Default Box Quantity", this can be calculated simply: Quantity (see above) / Default Box Quantity = Quantity in boxes.
Status	The stock status, a critical property used by the warehouse and also required in the Malcolms delivery message.

The above tables gives a summary of each field that will store the relevant delivery information retrieved during delivery process. The data will be populated from a several sources; RDBMS Id's (primary keys), user inputs (the lorry identifier being delivered) and for the most part the stock data and properties retrieved from the native UCP WMS at the time the delivery is confirmed.

4.1.3 Design Components: Oracle Data Access

The application includes an Oracle Data Access component responsible for retrieving delivery records that need to be passed from the UCP to Malcolms WMS. Utilising the dot Net Oracle.DataAccess.Client library, part of the Oracle developer tools for Visual Studio 2015, this component provides methods and functions for querying the UCP WMS stock records.

With regards the stock record retrieval introduced in 4.1.1, the ER diagram in Figure 14 shows the source tables that will be queried in order to capture the necessary stock record details. These details will be subsequently passed to the integration database' Shuttle Header and Shuttle Detail tables described in 4.1.2 to populate automatically.

Figure relates to the UCP WMS source database, used to retrieve required stock data

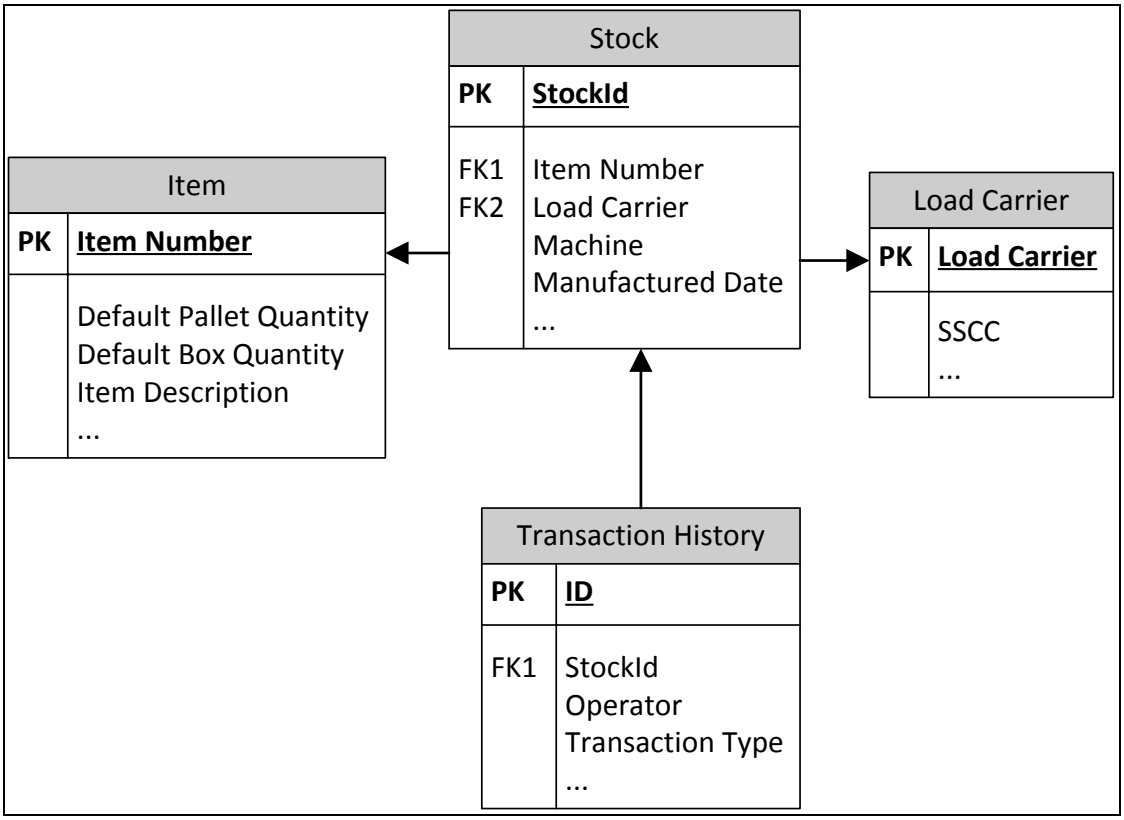


Figure 14: UCP WMS Oracle ER Diagram

Note: The UCP WMS Database uses German field references, in accordance with the home country of the vendor. The table and column names have been translated for the purposes of this document, but any implementation in code will use SQL that refers to the original German language variation.

Referring to the Malcolms delivery message specification output requirements (see Table 8 and Table 9 on page 32), most of the fields required can be retrieved directly from the UCP source data shown in Figure 14. The table below shows how this will be achieved and maps the fields from the UCP WMS Database (Figure 14) to the delivery message specification fields in 3.4.5:

Table 18 – Stock data retrieved from UCP WMS

MALCOLMS MESSAGE FIELD (SEE 3.4.5)	RETRIEVED FROM (SEE Figure 14)
SSCC	Load Carrier Table
Manufactured Time	Stock Table
Loaded Time	History Table
Operator Id	History Table

Item Number	Stock Table
Quantity	Stock Table
Machine	Stock Table
Batch	Stock Table

The stock data properties shown in Table 18, retrieved automatically from the UCP WMS, replace what was previously manually input into a spreadsheet form. With the data retrieval performed and collected in the application's database, this can now be passed to a data writer for output to the Malcolms import message file.

This message structure specification is detailed below, covering a header message (relating to the specific delivery) and a detail message (related to the delivery contents).

4.1.4 Design Components: StreamWriter File Access

The Malcolms delivery message is a file generation requirement that allows Malcolms to automatically import the delivery data. Therefore, a StreamWriter component has been included in the design. Part of the .NET System.IO library, this can be leveraged to package and generate output files to shared network locations. With the delivery data held in the SQL Integrator database, the StreamWriter can populate the message file output from the database as the source.

The message specification provided by Malcolms comprises a header and detail message file, and Table 19 (Header message) and Table 20 (Detail Message) describe how this message will be populated utilising the records stored in the SQL Integrator database. The column "Data" represents the output required by Malcolms in the message file, and the "SQL Source" column represents the source field in the applications SQL Integration database (see 4.1.2) which provides the data content for output. The fields that populate the delivery message header (shown in Table 19) are sourced from the SQL Integrator "Shuttle Header" table, and the delivery message detail (shown in Table 20) populated from the SQL Integrator "Shuttle Detail" table.

Table 19 – Populating the delivery message file (Header)

COLUMN	DATA	SQL SOURCE
1	ID – A unique identifier for each despatch sent	ShuttleId
2	Load Reference – The Shuttle Number	LoadRef
3	Despatch Time – The Date and Time the load was despatched	DespatchTime

Table 20 – Populating the delivery message file (Detail)

COLUMN	DATA	SQL SOURCE
1	EntryID – A unique identifier for each despatch line	Id
2	ShuttleHeaderId – The Load ID (unique header Id) that the stock belongs to	ShuttleId
3	SSCC – The unique serial shipping container code	SSCC
4	Manufactured Time_- The time the stock was manufactured	TimeProduced
5	Loaded Time – The time the stock was loaded onto the shuttle	TimeLoaded
6	Operator Id – The ID of the user who loaded the pallet onto the shuttle	LoadedBy
7	Item Number – The JDE Stock Item number	Item
8	Quantity – Quantity in Units	Quantity
9	Box Quantity – Quantity in Boxes (1 if in bulk pack)	QuantityBox
10	Machine – The Machine the stock was made on	Machine

4.1.5 Design Components: Presentation (Graphical User Interface)

Although the objectives are to automate as much of the data exchange as possible, the delivery to warehouse function requires two user inputs which are unavoidable. The first input enables the application to retrieve all stock data being delivered from the WMS database. The vehicle location reference must be input to retrieve this stock data as a critical search clause (see 0). The second input is a management requirement, a validation of the stock data being delivered. The operator must perform this validation at the point of delivery.

Discussed earlier, the presentation layer uses the Windows Presentation Foundation API, with each menu / window consisting of a XAML mark-up component and an associated code file. This allows the presentation components to be de-coupled from underlying code, with changes in one not affecting the other. In addition to the above input needs, the following considerations have been addressed:

- ✓ The migration from a paper based delivery process to a software solution will require the provision of a PC at the delivery workstation. A touch screen monitor running Windows 7 has been provided for this purpose to ensure inputs are intuitive for staff members that are unfamiliar with using PC's.
- ✓ The GUI layout is therefore designed to be as simple to follow as possible, with buttons, text, and instructions presented in large font and format to ensure selection is easy through the use of a touch display.
- ✓ The current requirement is to provide one function only, however a main navigation window has been designed at this stage as the business have suggested there may be future requirements to perform different types of deliveries. Additional functions can be added to the navigation in the future.

A description of the GUI component design is provided below. Microsoft Visio has been used to mock up window layouts, buttons, text and images.

4.1.6 Main Navigation Window

The Main Navigation Menu is redundant at this stage as the business is only operating one function, and therefore one navigation route is provided. However, as per the business requirements discussed there is a strong possibility this may be expanded in the future, following successful proof of concept, to capture additional data. Therefore, a Grid Layout navigation menu has been incorporated anyway so additional functions can be added later.

Figure 15 shows the design concept for the navigation window, a Grid Layout providing a Delivery to Warehouse button, with space for additional functions.

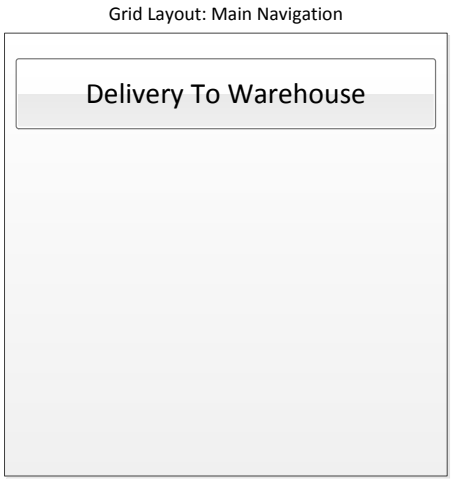


Figure 15: Main navigation Menu (Design)

4.1.7 User Input Window

The User Input window (Figure 16) allows the delivery operator to provide some input data. The current input requirement is for the lorry location reference, needed to retrieve stock data from the UCP WMS Database. The input template is designed to be populated at run time, so similar user input needs can be re-used for additional functions implemented later. Components populated at run-time are as follows, with the design mock up in the adjacent figure:

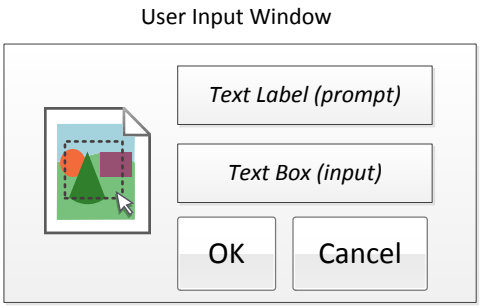


Figure 16: User input window (Design)

- ✓ Image place-holder for an icon relating to the input. In the case of the delivery function this could be an image of a lorry.
- ✓ Text Label providing an instruction prompt. In the case of a delivery this will instruct the operator to "enter the lorry reference".
- ✓ A text box field captured for the data being entered

4.1.8 Data Display Window

The Data Display menu (Figure 17) allows the delivery driver to check the stock data about to be delivered. It lists the stock data returned from the UCP WMS Database, and if there is an error (such as a stock is missing) he can abort the delivery, rectify the problem, and try again. This addresses a process management request in this user story and provide a simple and quick way for a delivery operator to check the stock data he has prepared.

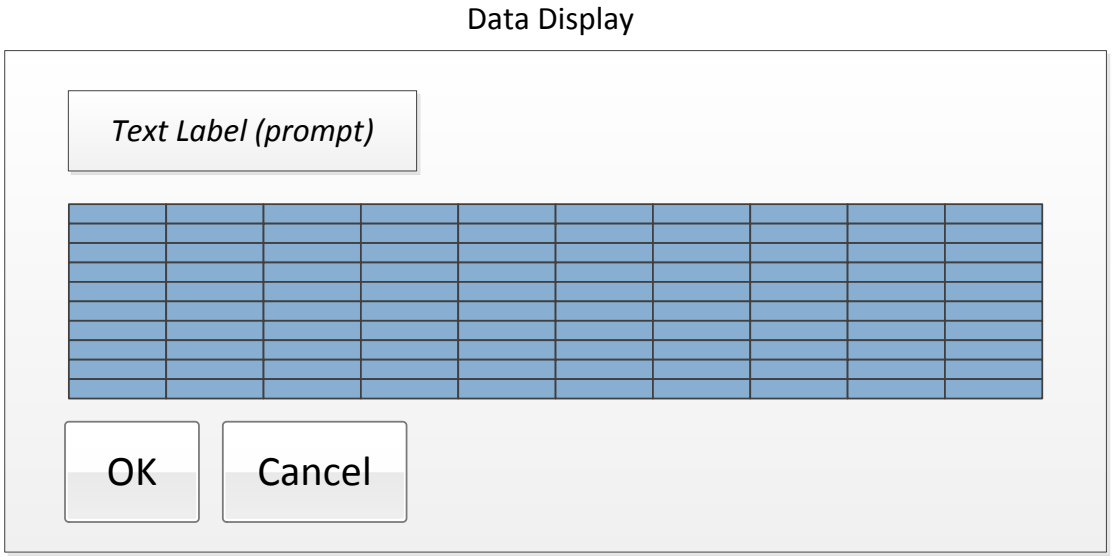


Figure 17: Data display window (Design)

Figure 17 contains two dynamically populated elements:

✓ Text Label which is populated with an instruction prompt for the operator. In this case it will be related to "checking the stock data on the delivery, press OK to continue, cancel to abort".

✓ DataGrid component which will be filled from the oracle data reader result set.

Upon positive validation the rest of the workflow will be executed, else delivery process will be aborted (See 4.1.10 below).

4.1.9 Class Diagram

The UML class diagram is shown in Figure 18 below. XAML Presentation and Data Access layers described above are shown with the addition of business logic classes that make up the final class diagram.

Two additional classes represent the business logic component of the application. These are a ***Load*** class, and a ***LoadCarrier*** class. These two classes represent the stock records that are being retrieved and exchanged in the application, with the Load class representing a delivery vehicle, and the LoadCarrier class representing a stock record.

This design follows the traditional Object Oriented class design relationship known as either an "Is A" relationship or a "Has A" class relationship, where one class is a type or member of another. In this case, stock records represented by LoadCarrier objects, are members of a delivery vehicle, represented by a Load object.

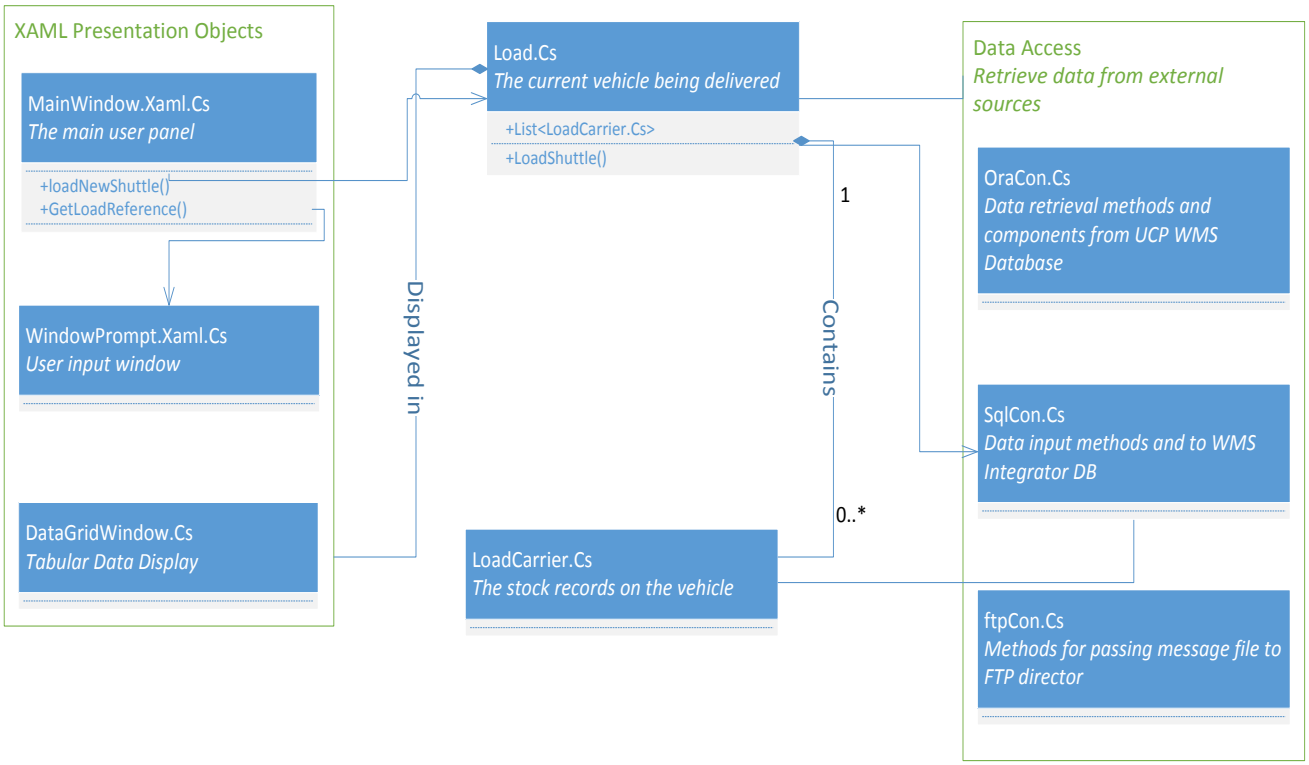


Figure 18: Class Diagram (User Story 1)

Summary of class diagram components:

- ✓ ***Load*** object classes represent a delivery vehicle, and they contain member list variable of ***LoadCarrier*** objects for each stock record being delivered. These two classes reflect the data records loaded on a delivery vehicle in memory, and used for subsequent data handling.
- ✓ ***Data Access*** components (OraCon, SqlCon, and ftpCon) make up the classes that allow the retrieval of data to and from the Load and LoadCarrier classes with external data sources.
- ✓ ***XAML Presentation*** classes represent the user interface described in 4.1.5 above that allow the presentation of stock records for validation and facilitate user input.

4.1.10 Design Workflow

With an understanding of the application and class structure, the associated workflow is discussed below. The starting point to this process is that some stock has been prepared for delivery by UCP. This stock is prepared by updating all stock records with a location record associated to the delivery vehicle.

At this point, an operator needs to send a delivery to the warehouse. The workflow to achieve the data exchange form the UCP to Malcolms WMS is achieved as follows (One alternate path is provided). Workflow is visualised in Figure 19 and references at each stage.

Primary Path:

I. **Delivery operator initiates a new delivery from the main menu** (Figure 19, Step 1)

To be executed from the GUI, a single function is required at present; delivery to warehouse.

II. **Operator inputs the vehicle location** (Figure 19, Step 2)

To retrieve the stock records from the WMS database, the operator must input the vehicle identifier as search criteria. This allows the data access layer to retrieve the records to memory. There is no systematic way to determine which vehicle location is being loaded at time of delivery, so this is an unavoidable input. In future, mechanical solutions could be engineered to present a vehicle location electronically, however this is out of the scope of this development and would require significant capital investment to automate.

III. **Retrieve stock records from UCP WMS Database.** (Figure 19, Step 3)

Using the input in the previous step, the database can be queried with this input as the clause.

IV. **Display Stock Records** (Figure 19, validation branch)

Pass stock records to presentation layer, and wait for operator view and validate.

V. **Pending positive validation, stock records written to SQL Integrator Database.** (Figure 19, Step 5)

A database will be used to store delivery history, containing the vehicle, date, time, and stock records on the vehicle. In addition, a flag will be set that indicates the specific delivery is in transit.

VI. **Stock records exported from database to message file for import into Malcolms WMS** (Figure 19, Step 6)

The message is passed to the Malcolms implementation for direct import, removing the need to manually enter data from a paper form. Upon delivery of the message, the data exchange is complete.

Alternate Path:

IV. **Display Stock Records** (Figure 19, validation branch - failed)

If the operator notices an issue when presented with the stock records, a negative confirmation is provided and the process must be restarted and the UCP WMS stock records checked.

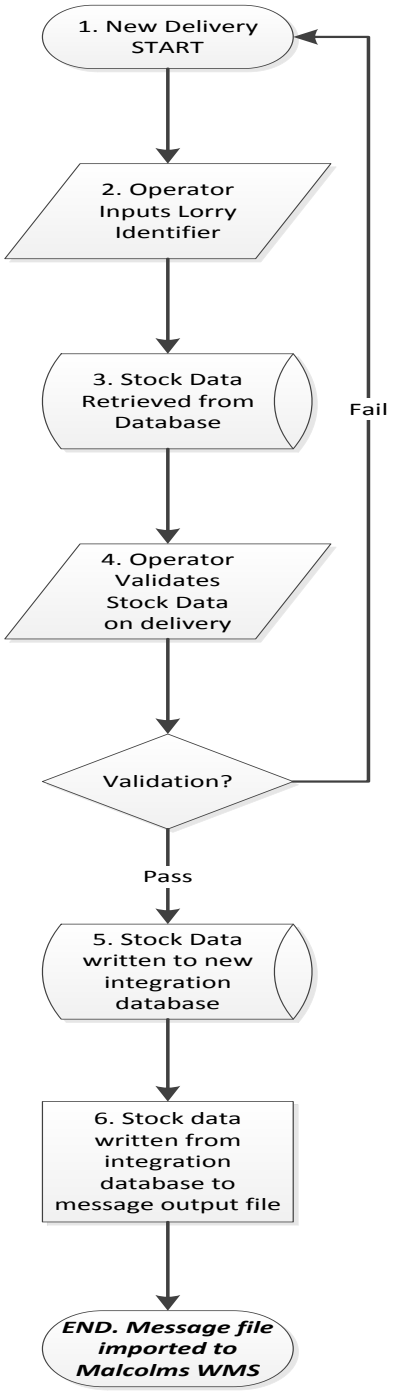


Figure 19: Deliver to Warehouse Design Workflow

4.2 User Story 2: Stock Status Update

Design structure for this user story is based on the business-critical need to automate the updating of a stock status record in the Malcolms WMS, triggered by a stock status change in the UCP WMS. Requirements are outlined in 3.5.

4.2.1 Design Components: Overview

In contrast to the design described in User Story 1, the stock status update function requires no user input and acts as a UCP to Malcolms message exchange solution, so designed as a console application that will run as a background process.

Figure 20 shows the status update application in the context previously used, with the UCP WMS to the left, and the Malcolms to the right. The application is represented by the red container and made up of a series of components described further in this section.

The components are made of two distinct layers; A Data Layer (FileReader, FileWriter and SQL components) combined with the Business Logic layer (shown in Figure 20 "Data Layer" and "Business Logic" containers). The linear arrow describes the data flow through these components.

The SQL Integration Database introduced in User Story 1 will be extended with additional tables to underpin the specific Stock Status Update requirement (discussed further in 4.2.5).

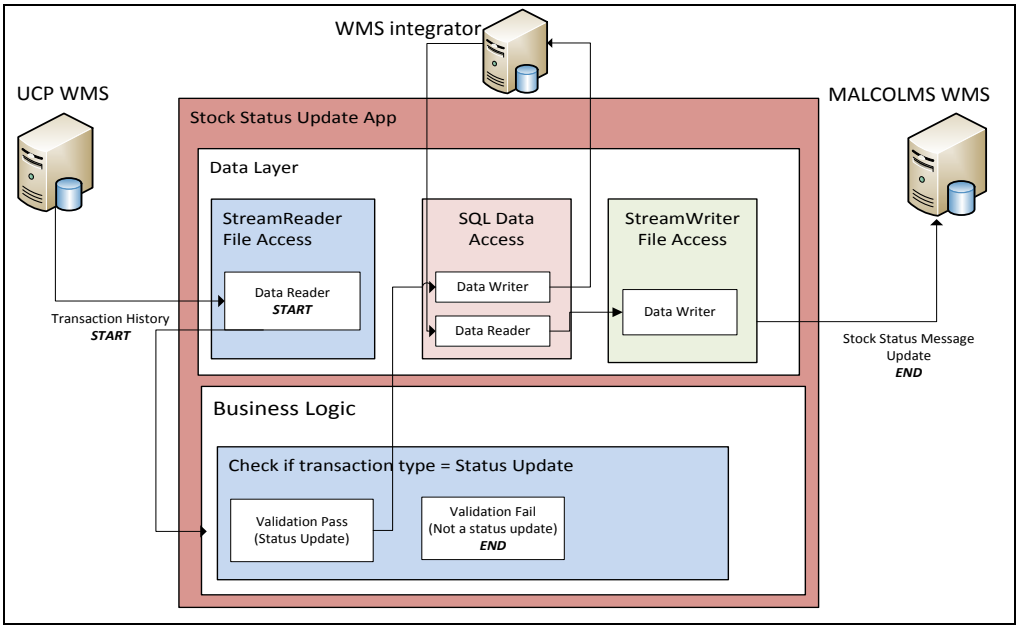


Figure 20: Design Overview (User Story 2) - Application Structure

The StreamReader component (in Figure 20 "StreamReader File Access") reads a WMS output stream (discussed further in 4.2.2) containing a history of all transactions performed in the UCP

WMS. This WMS Output stream provides the trigger mechanism for passing a Status Update message to Malcolms, and forms a crucial component of the design.

Transactions are passed to a Business Logic layer which parses the transaction history to identify scope and treat data accordingly. If this check passes, a stock status update record is written to the *SQL Integrator Database* and an associated Status Update message file is sent to Malcolms using the StreamWriter file access component.

This completes the automated data exchange, and the component parts are discussed further below, following an introduction to the WMS Output Stream. See following sections for expanded discussion on each of these components.

4.2.2 UCP WMS Output Stream

The output stream contains a history of all native WMS transactions performed in the UCP WMS, including the Stock Status Updates that must be read. Thus, this output stream can be monitored providing the trigger mechanism required to initiate a stock status update.

A complication with this approach is that the output stream is used and purged by a separate manufacturing system used by UCP, their ERP system (Enterprise Resource Planning). Therefore, to use this output stream, the solution must intercept, process, and then pass the file back and in-tact to avoid disrupting this process. This must be handled with care, as any interruption to the ERP service is mission critical.

As this output stream provides the key inputs into this function, research has been performed on this subject to understand its structure, and which messages constitute a status update. See the following sample retrieved from the UCP WMS in Figure 21.

```
1  AG00045845 W024416246          000000001008640635      0000020900.00Each     24416254
2  UL00045845                     000000000008384985      0000007560.00Each     24405475
3  EG00045845 W024416246          000000001008640627      0000020900.00Each     24416246
4  UL00045845                     000000000008512730      0000015960.00Each     24264473
5  UL00045845                     000000000008512730      0000015960.00Each     24264473
```

Figure 21: WMS Output Stream

Figure 21 shows a fixed length file, containing several fields. The following fields shown in Table 21 are worthy of note. The column “Name” defines the field in the output stream, with the “Character Position” and “Field Length” column indicating its point in the file.

Table 21 - WMS Output Stream - File Structure Specification

NAME	CHARACTER POSITION	FIELD LENGTH	VALUE
Transaction Type	1	2	"AG" = Picking Transaction "EG" = Receipt Transaction "UL" = Shift Transaction ***"BL"*** = Block (***Stock Status transaction***) ***"UB"*** = Unblock (***Stock Status transaction***)
Transaction Time	162	14	Format = "YYYYMMDDHHMMSS"
StockId	348	9	The unique stock record Id

shows the column "Transaction Type" (highlighted) which is of importance. The values contain a series of codes determining the type of WMS transaction performed. The following are of interest in the context of this discussion:

- UB – Stock status from "Bad" to "Good" – also known as an "unblock".
- BL – Stock status from "Good" to "Bad" – also known as a "block".

The transaction type is what identifies a ***Stock Status Update*** and those which are not. Reading this column will provide the starting point of a trigger mechanism in the application.

4.2.3 Design Components: StreamReader File Access

This component is responsible for monitoring a WMS output file stream (see 4.2.2 above). The output stream contains a history of all native WMS transactions performed in the UCP WMS, including the Status Updates that must be read. Once transactions are read into memory, they are passed to a Business Logic layer further processing.

4.2.4 Design Components: Business Logic

A series of business logic tests are performed on the WMS Output Stream. This file stream contains a history of all native transactions performed in the WMS so checks must be made to identify the specific type, a Status Update. Validated transactions are forwarded for further processing.

For a pass, the transaction must be of a "type" equal to "UB" or "BL" (see 4.2.2) and relate to a stock record that has a Warehouse location record associated to it.

4.2.5 Design Components: SQL Data Access and Database Table Structure

The SQL Integrator Database introduced in the previous user story will be extended with additional tables to support the Status Update function. The SQL data access component within the application will also leverage the dot net System.Client.SQL data access libraries to write the associated transaction history of stock status updates.

Two new tables will be added, "Stock Status Update Header" and "Stock Status Update". The header table will contain a record of each WMS Output Stream file that is imported by the application. The "stock status update" table will include a history of all Status Update messages sent to the Malcolms WMS. Figure 22 shows an ER diagram of this design, where a "WMS Output stream file" (Stock status update header table) may contain zero or more Stock Status Update records (Stock status update table). The status update records stored over two tables as such ensures design achieves third normal form.

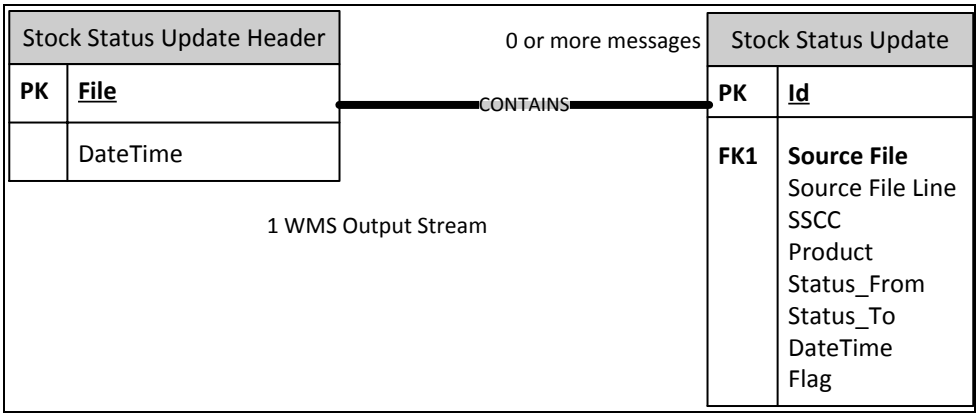


Figure 22: Stock Status Update ER Diagram

Figure 22 is expanded upon with a supporting commentary for the fields and descriptions in Table 22 (Stock status update header table) and Table 23 (stock status update table).

Table 22 Status update: Header table commentary

FIELD	PURPOSE
File	Primary Key: The WMS Output Stream file. One record for every output stream imported
DateTime	The date and time the output stream was imported

Table 23 Status update: detail table Commentary

FIELD	PURPOSE
ID	Primary Key: Database Id that uniquely identifies the stock status update message
SourceFile	Foreign Key: Joined to the *File* column on the header table. Represents the source WMS Output Stream file.
SSCC	Serial Shipping Container Code. Synonymous with the WMS Load Carrier property. Required in Malcolms Status Update message specification.
Product	Stock record property that relates to the type of parts contained in the stock record. Required in Malcolms Status Update message specification.
Status_From	The previous value of the stock status property. Required in Malcolms Status Update message specification.
Status_To	The new value of the stock status property. Required in Malcolms Status Update message specification.
DateTime	The time of the stock status change transaction. Required in Malcolms Status Update message specification.
Flag	A binary value, 0 or 1. Default value is 0, to be changed to 1 when the message is accepted by Malcolms (requires a confirmation message provided by Malcolms). This will allow us to trace which messages have been accepted, and investigate any messages that have failed. A future development consideration

4.2.6 Class Diagram

The UML class diagram is shown in Figure 23 and described in the following paragraphs.

A console application, the class structure and associated components are linked through the Main() console application entry point (Figure 23, “Main” Class).

The WmsStream class (Figure 23, “WmsStream” class item) implements the System.IO StreamReader libraries (described in 4.2.3 above) providing methods for importing the WMS Output Stream transaction file contents (described in 4.2.2 above) and parsing the data. Status Update transactions are retained from the stream, with the rest disposed, and passed to the Update Message class (Figure 23, “UpdateMessage” class object). This class is responsible for packaging the transactions into the message file required in 3.5.5 above. Both the message and message source file details are

inserted to the SQL Integrator Database via the SQLConnect data connection class (Figure 23, "Sql-Connect" class object) providing traceability and message exchange record retention.

Finally, the FtpConnect class (Figure 23, "FtpConnect" class item) implements System.Net and System.IO libraries which allow the message file to be passed to the Malcolms import directory using the file transfer protocol, the final stage of the data exchange. This is due to the Malcolms implementation being inaccessible from the UCP private network, so a shared FTP directory accessible from the internet provides the intermediary message exchange platform.

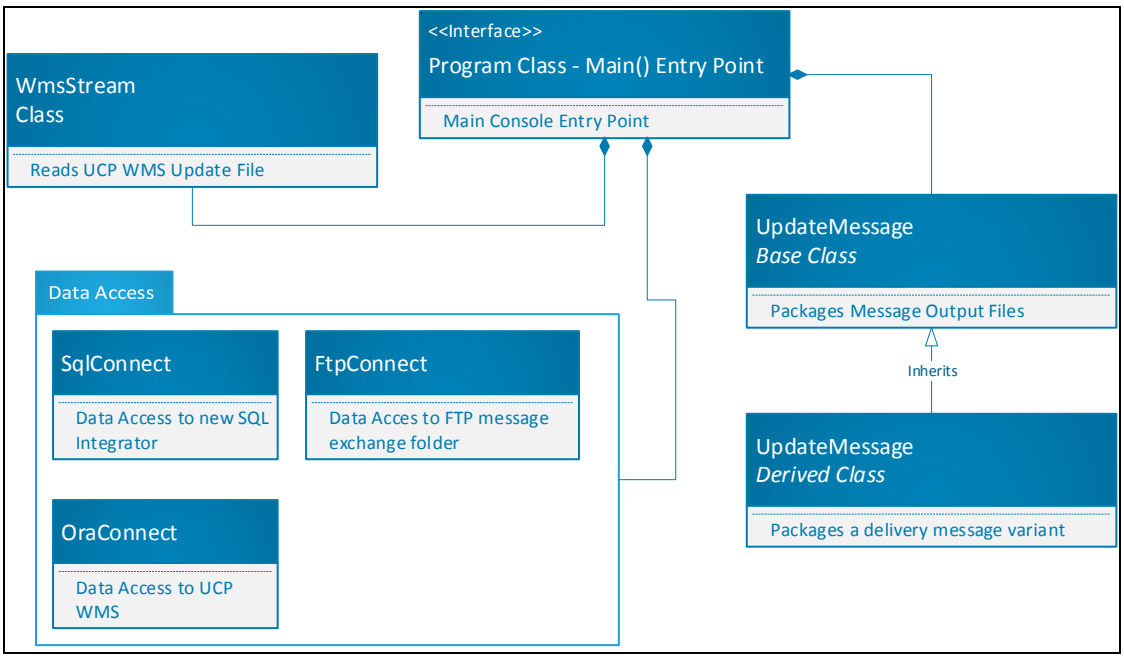


Figure 23: Stock Status Update - Class Diagram

4.2.7 Design Workflow

With an understanding of the application and class structure, the associated workflow is discussed below.

The starting point to this process is that some stock has been blocked in the UCP WMS software. The same status update must now be replicated in the Malcolms WMS, described as follows with references to the visual in Figure 24.

Primary Path:

I. **Check if WMS Output Stream Exists** (Figure 24, Step 1)

The critical input, if there are no transactions performed in the WMS, the application has no work to perform. This check will be performed on a frequent schedule until some transactions exist for processing.

II. **Read lines from WMS Output Stream** (Figure 24, Step 2)

The WMS output stream is read into memory for further processing. Contains record of all recent WMS transactions.

III. **Branch 1: Check if transaction is a status update** (Figure 24, Step 3)

Using the logic described in 4.2.2, gather status update transactions only. The “type” field is checked for either a “BL” or “UB” value (see Figure 21).

IV. **Branch 2: Check if stock record is at Warehouse location** (Figure 24, Step 4)

As described in detail previously, the scope of the exchange is applicable to stock records located at the Malcolms warehouse only. Therefore, the stock record’s location must be checked before processing further.

V. **Export Status Update message** (Figure 24, Step 5)

If the transaction satisfies the check in Branch 1 and Branch 2 above, the Status Update message is packaged and exported to file for Malcolms import.

Alternate Path (Branch 1):

III. **Branch 1: Check if transaction is a status update** (Figure 24, Step 3)

If the transaction read from the UCP WMS is not a Status Update, then no further action is required.

IV. **Branch 2: Check if stock record is at Warehouse location** (Figure 24, Step 4)

The second Branch checks the stock record location if the first branch passes – if the stock record has a location not equal to the Warehouse then no further action is required.

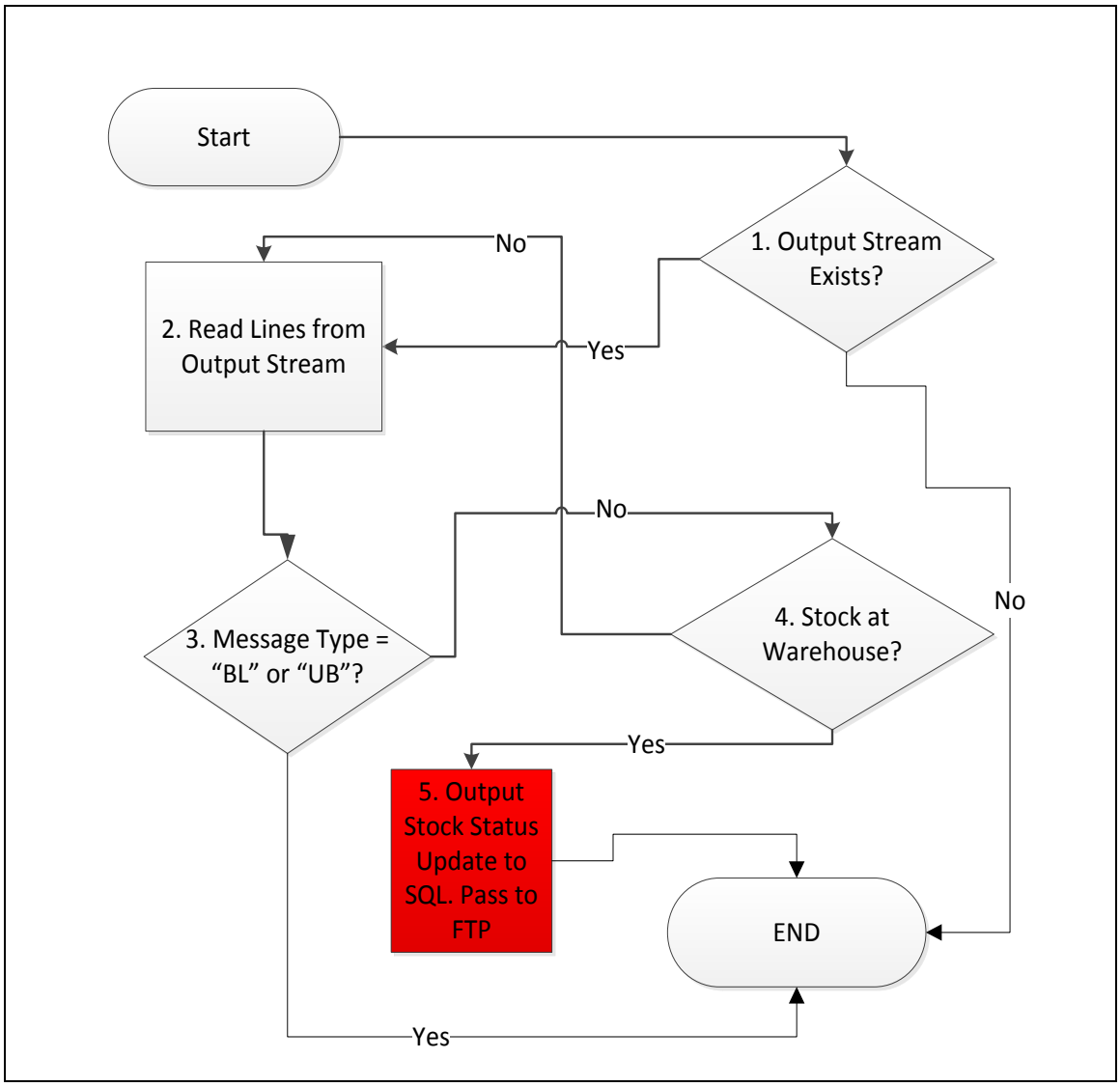


Figure 24: Status Update - Design Workflow

4.3 User Story 3: Warehouse Update

Design logic reflects the Stock Status Update (4.2) but reversed; the trigger is generated from the Malcolms WMS, and the output is the UCP WMS.

A StreamReader component (Figure 25 – "StreamReader File Access") is used to read a message file containing stock records that have been delivered from the warehouse.

These are written to the SQL Data Access component (Figure 25, "SQL Data Access") which passes the transaction data to the new SQL Integrator Database introduced.

Following the processing of any business logic against the Malcolms WMS data (Figure 25, "Business Logic"), a new component is introduced as part of the Oracle Data Access layer; a direct SQL UPDATE function (Figure 25, "SQL UPDATE"). This will directly update the UCP WMS database records accordingly.

Due to the close technical similarity, the class structure of User Story 2 should be extended, allowing use of common code across applications through the use of base classes (see 4.3.2).

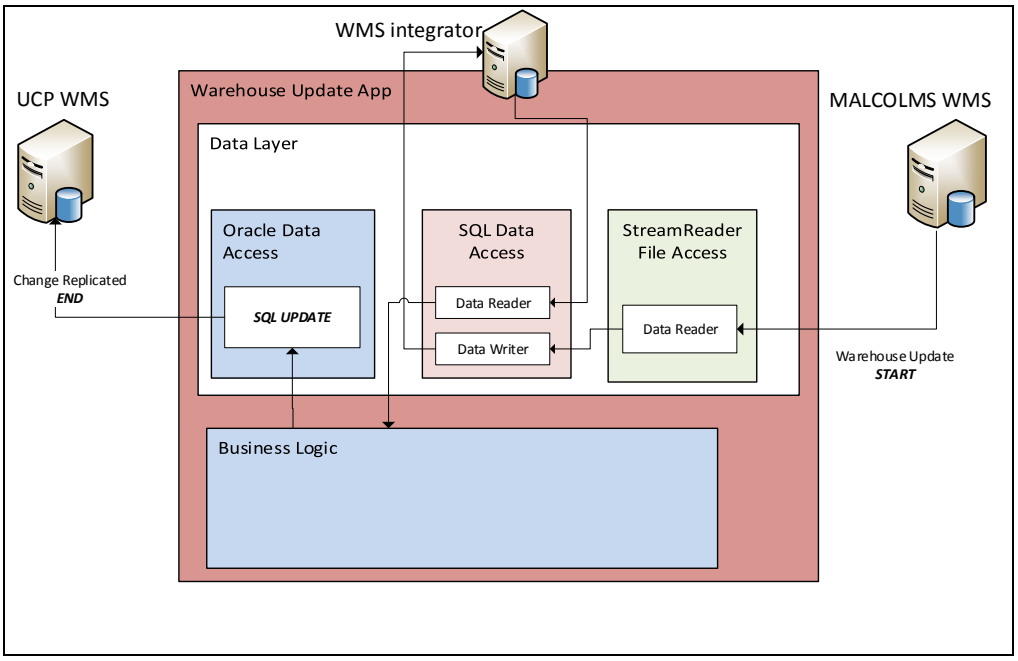


Figure 25: Design Overview (User Story 3) - Application Structure

4.3.1 Malcolms WMS Output Stream

The Malcolms WMS output stream provides both the trigger and stock record data for an exchange being passed from the Malcolms to UCP WMS. User Story 2 describes the reverse of this scenario, but the same principles apply. A System.IO library StreamReader will be implemented to retrieve and parse the stream, composed of the delivery and receipt messages described in 3.6.5 and 3.6.6. There-

fore the existing class from User Story 2 has been extended to accommodate this, discussed further in 4.3.2.

The receipts message file will contain confirmation of the stock records that have been previously sent in user story 1 (delivery to warehouse). The pallets delivered file will contain records of all stock that have been sent to customer or returned back to the factory.

These two file types provide all the necessary inputs to facilitate the UCP side data updates previously performed manually. The extension of User Story 2 "WmsStream" class is described below.

4.3.2 Extending the WmsStream class

The WmsStream class introduced in 4.2.2 has been split into two derived classes, a UCP and Malcolms WMS Output stream (Figure 26, highlighted green). For the Warehouse Update requirement, the stream is the Malcolm's message files containing the warehouse stock record changes (deliveries or receipts). With both design concepts covering technically similar functions, this structure should ensure code duplication is minimised, with common components and methods can be re-used across derived classes.

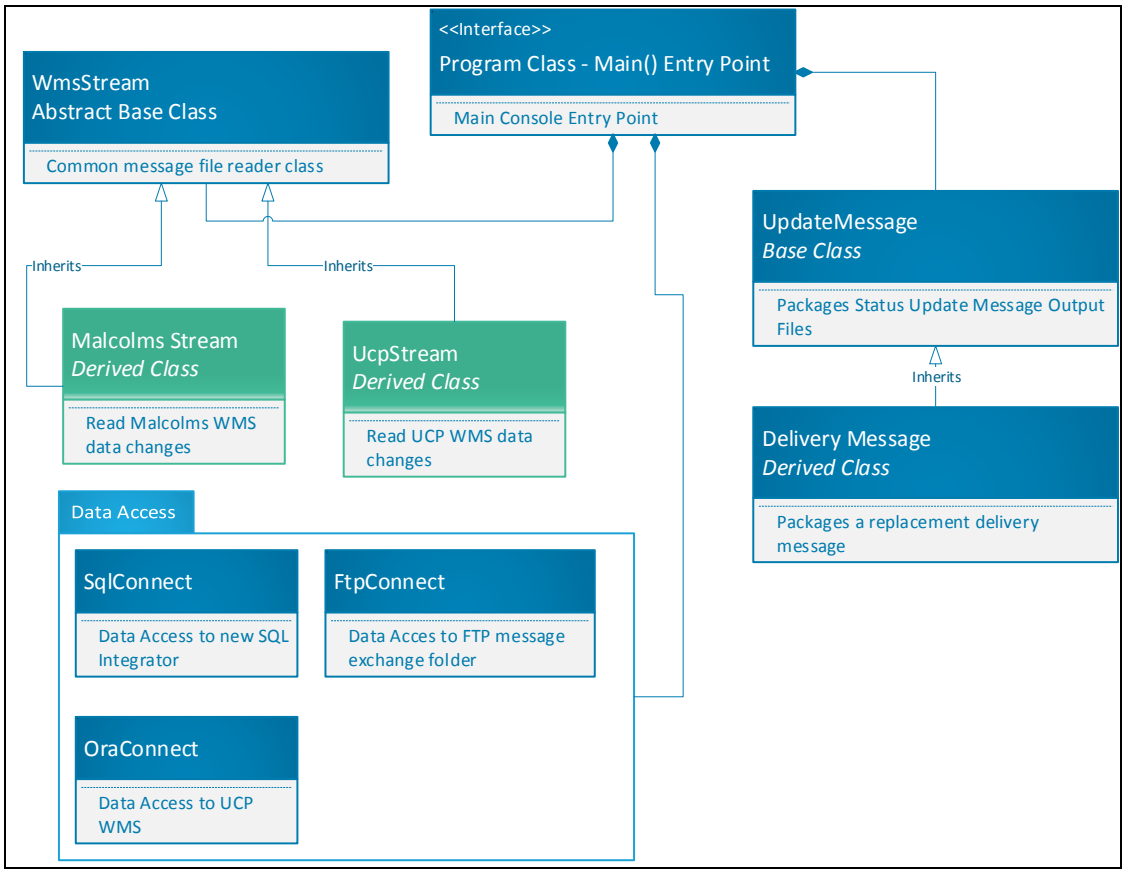


Figure 26: User Story 2 Class extension

4.3.3 Design Components: Business Logic

A series of business logic tests are incorporated into the Warehouse Update design.

The first relates to the "Unit of Measure" used in the Malcolms stock record messages. Referring to the stock record structure in 2.1.2, there is a conflict between the UCP and Malcolms definition. Malcolms use "number of boxes" as the stock record's base quantity, whereas UCP use the number of products in all boxes. For example, if a stock record consists of 35 boxes, with 1,000 products in each, the UCP stock record would have a quantity of 35,000 and the equivalent Malcolms stock record would show 35. Therefore, a conversion will be implemented to address this need.

Another check is the type of stock record contained in the message. During the design phase, it appeared that Malcolms WMS contains a small number of packaging items in their own WMS that UCP do not control via their WMS. Although infrequent, these will also be contained in the Malcolms WMS Output Stream messages, so must be filtered out.

To pass both checks, each message must therefore be able to be converted to the default UCP quantity UOM, and relate to a non-packaging product type.

Finally, the Malcolms updates messages contain a limited number of delivery locations. These locations have different references in the UCP WMS so must be translated at run time from Malcolms to UCP location relationship mapping.

4.3.4 Design Components: SQL Data Access and Database Table Structure

The SQL Integrator Database will be extended further to accommodate the message reversal of the message exchange already discussed. The same SqlConnect class will be used that contains the system.Data.SqlClient library components for this interaction. Furthermore, the table structure concept mirrors the design exactly as per the User Story 2 design – the addition of tables to accommodate the Malcolms WMS Output Stream instead of the UCP WMS Output Stream.

4.3.5 Design Components: Oracle Data Access

The Oracle Data Access component forms the final stage of the data exchange use case. Methods will be added to this layer that update the necessary UCP WMS Stock records accordingly. Each permutation of a stock record update will require two table modifications as a single stock record transaction:

✓ Updating the stock record table with a new location

The critical output of a data exchange in this user story. An SQL UPDATE statement must be passed to the BESTAND (Stock) table to adjust the location record property.

✓ Adding a location change transaction to the journal

As a secondary but critical process in the transaction, a journal table, used for WMS history reporting and traceability, must be updated with an entry relating to the delivery. An SQL INSERT statement must be passed to the TABAKT (History) table to facilitate this need.

5 Implementation

5.1 Delivery to Warehouse App (User Story 1)

This module provides a walkthrough and commentary on the implementation of the new Deliver to Warehouse application. Context is provided in relation to the original problem activities that are replaced by the new process.

In contrast to the lengthy processes this application replaces, the execution of the new delivery application takes less than 60 seconds from start to finish. The delivery operator is the key staff member responsible for the preparation and execution of the delivery in its entirety. This use case directly replaces the activities they perform in the manual system, outlined in 2.1.8 (Table 1).

To execute the new delivery function, the delivery operator loads stock records onto a vehicle in the native WMS as a critical pre-requisite. This critical pre-requisite is summarised in 5.1.1 to provide context.

Ready to confirm delivery of the vehicle, the new process is executed and facilitated by upgrades to supporting infrastructure and connectivity in the delivery area, described further in 5.1.2.

Initiating a new delivery through the application interface, the application is executed and subsequently manages all data logging, retrieval, and exchange functions previously described in the requirements and design chapters. A walkthrough of this in practice is described with supporting screenshots and commentary starting at 5.1.3 through to the end.

5.1.1 Preparation

Delivery preparation remains unchanged in the new process but adds context to the subsequent steps of the implementation. The stock records being delivered are loaded onto a delivery vehicle in the UCP WMS, and this results in the stock records being associated to a vehicle location.

Figure 27 shows two stock records in the UCP WMS (marked “Stock Record 1” and “Stock Record 2”) that have been prepared for delivery as described. The record marked “Stock Record 1” shows one stock of Johnny Walker Black Label with a status “blocked” (column marked “Status Property”). The record marked “Stock Record 2” shows one stock of Morgans Spiced with a status available (column marked “Status property”). Both have a location record LOR-05-06, the vehicle being delivered (column marked “Location”).

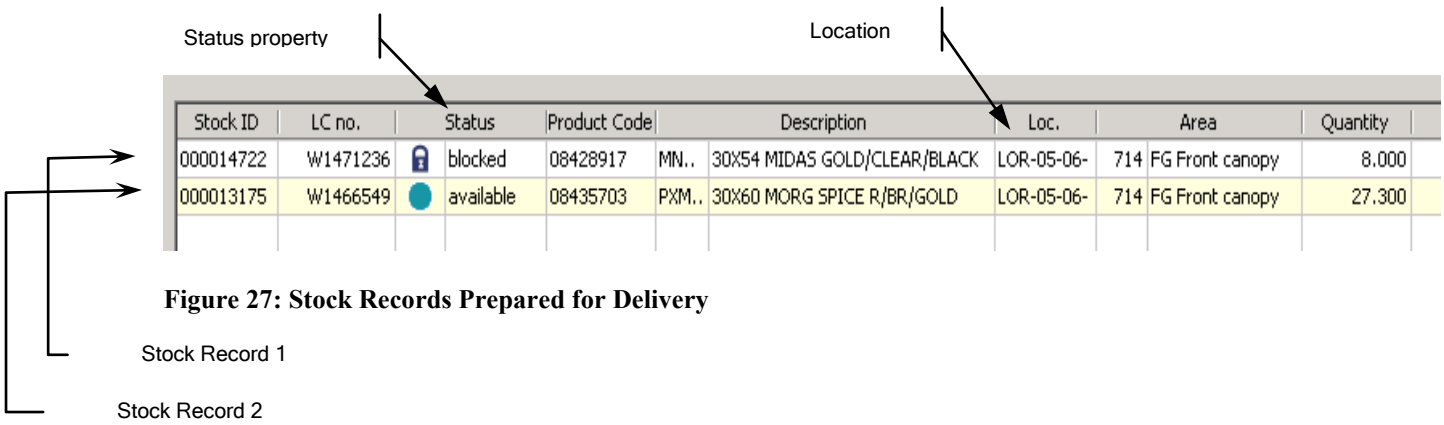


Figure 27: Stock Records Prepared for Delivery

The records presented in Figure 27 are now ready for delivery.

5.1.2 Supporting Infrastructure

The processes discussed are physically performed in an area of the factory called the "Front Canopy" – this is the area where UCP WMS delivery operators physically load stock onto vehicles destined for Malcolms.

Keeping the area of work confined to the one area, a Windows 7 Professional desktop has been mounted with a touch screen display in this area, near the previous workstation where manual administration was performed. This provides the applications point of use.

An Active Directory group policy object has been created and applied to this machine to secure the desktop operating system allowing only specified functions to be performed on this terminal. This control had been demanded by management, as this new network attached device is positioned in an area of the plant that is used 24 hours per day, accessible by all shift staff members. Another risk, a critical business function was now reliant on a PC running in a sound state, creating a potential bottleneck. Any issues during the nightshift could impact night shift deliveries without the availability of super users or IT support. A counter measure to mitigate this risk was to replace the company standard domain user logon with a generic user account, and an automatic logon facility introduced using a registry update. This means in the event of a problem, the user only needs to reboot the machine, and it will log on and take them back to where they need to be quickly and easily without causing an interruption to operational functions.

Finally, the company firewall has been opened to allow the machine's IP address access to a shared FTP server, which will be used by UCP and Malcolms for message file exchanges.

5.1.3 Global Application Setting Adjustments

The entry point in the WPF application is the App.xaml file. This XAML file contains an *Application Object* that provides some global application settings. A minor adjustment has been added, the *Startup* attribute. This allows an associated startup script to run prior to the applications main window loading. The attribute's property, "Application_Startup" refers to the method called in the code. This method is limited to a splash screen at present (Figure 28), showing version number and contact details to support troubleshooting. However, this startup method is now in place and can be leveraged for future needs, with addition of code added accordingly.

Figure 28: Splash Screen – Cosmetic addition with version details

5.1.4 Main Navigation

The main navigation window is loaded as the application's primary entry point and provides one function at present (Delivery to Warehouse). The *StartupUri* element in the global App.Xaml (see 5.1.3) defines this behaviour instantiating a new MainWindow.xaml object.

This main navigation has been constructed using the WPF StackPanel component of the XAML mark-up. This has been used as it allows a basic structured composition of a window and associated objects (Buttons, Windows, Text, Labels) using a simple tag structure with associated attributes.

The StackPanel contains a grid layout consisting of two columns and one row, which can be extended in the future. The first column containing the button (Figure 29, marked "Despatch to Malcolms") and the second containing the associated label instruction (Figure 29, marked "Press to delivery vehicle"). Additional rows can be added in the same format if new functions are needed in the future.

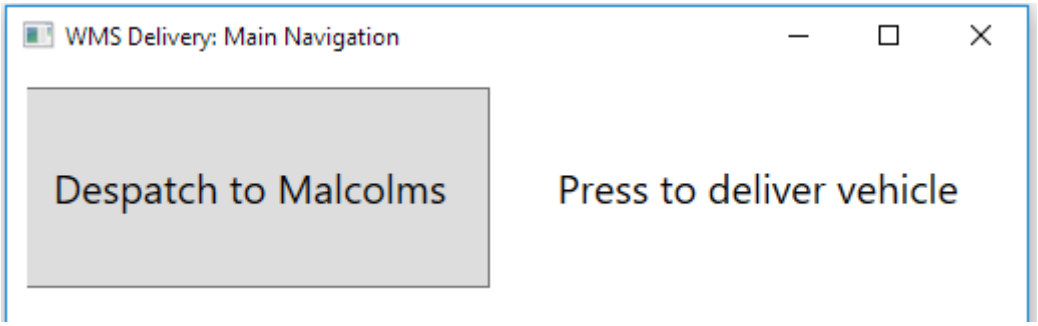


Figure 29: Main Navigation - Instance of MainWindow.Xaml Object

5.1.5 Delivery to Warehouse Function

The Button element in the Main Navigation (Figure 29) has an event handler mapped to it in the XAML mark-up, this is responsible for executing the delivery workflow.

The first phase of the process requires the implementation of the WindowPrompt.Xaml component which enables the operator to input the vehicle location reference for the vehicle being delivered, e.g. "LOR-05-06" has been used in 5.1.1 (Preparation).

WindowPrompt.Xaml has been designed with a constructor implemented that accepts four parameters passed at run time: An image file, window title, an instruction message, and a default value for the data entry field. This ensures the WindowPrompt.Xaml can be re-used for similar user input needs in the future, with different graphic, instructions, and window titles as needed. The call to the MainWindow.XAML object and its associated representation when loaded is shown in Figure 30 (The call) and Figure 31 (Presentation).

```
public string getLoadReferenceFromUser()
{
    /* Create new instance of user input class with text instruction */
    WindowPrompt confirmLoad = new WindowPrompt("pack://application:,,,/images/img_Lorry1.png",
        "Scan Load Reference", "Please scan the load reference to be despatched:", "<scan or type>");
```

Figure 30: WindowPrompt.XAML constructor

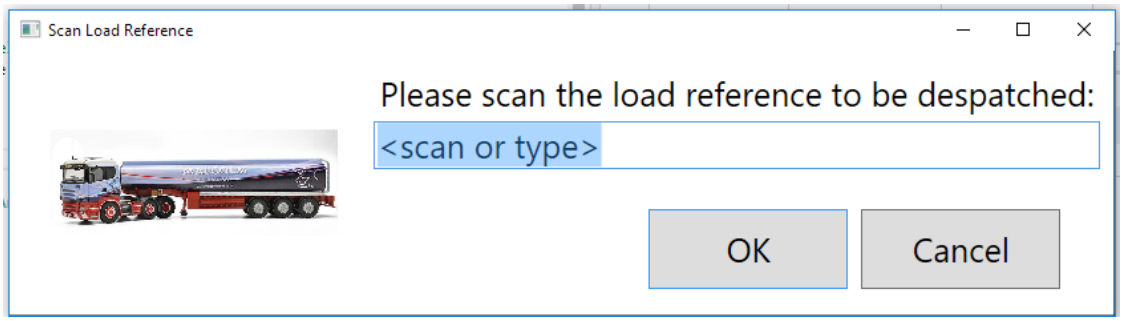


Figure 31: Vehicle input window - Instance of WindowPrompt.Xaml object

The text “scan or type” is based on the fact that a delivery operator has a barcoded location record available at delivery time, used for performing preparation activities. This saves time entering the vehicle location manually, but this can be performed if needed.

5.1.6 Load Class (The delivery Vehicle)

A new instance of the Load class is instantiated and the vehicle reference stored in the “loadReference” property. The class represents the active “Load” (the vehicle) being delivered. The new object also contains a member variable *pallets;* this is a List of LoadCarrier class objects, a placeholder for the stock records loaded on the vehicle. This Load and LoadCarrier relationship is represented in Figure 32 showing the associated *pallets*, and *loadReference* properties in the Load class. The Load class implements a List<> of LoadCarrier objects, rather than an Array[] of objects, due to the greater flexiflexibility provided, primarily being able to dynamically increase the list size at run time; a delivery will always contain a variable number of stock records. In contrast, an array required a fixed array size definition.

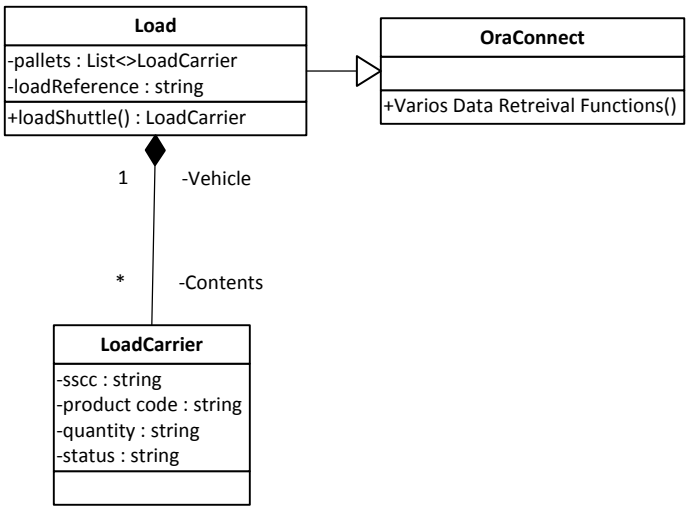


Figure 32 Load class implementation

Included in the Load class is the LoadShuttle() method (Figure 32) which is responsible for two main function:

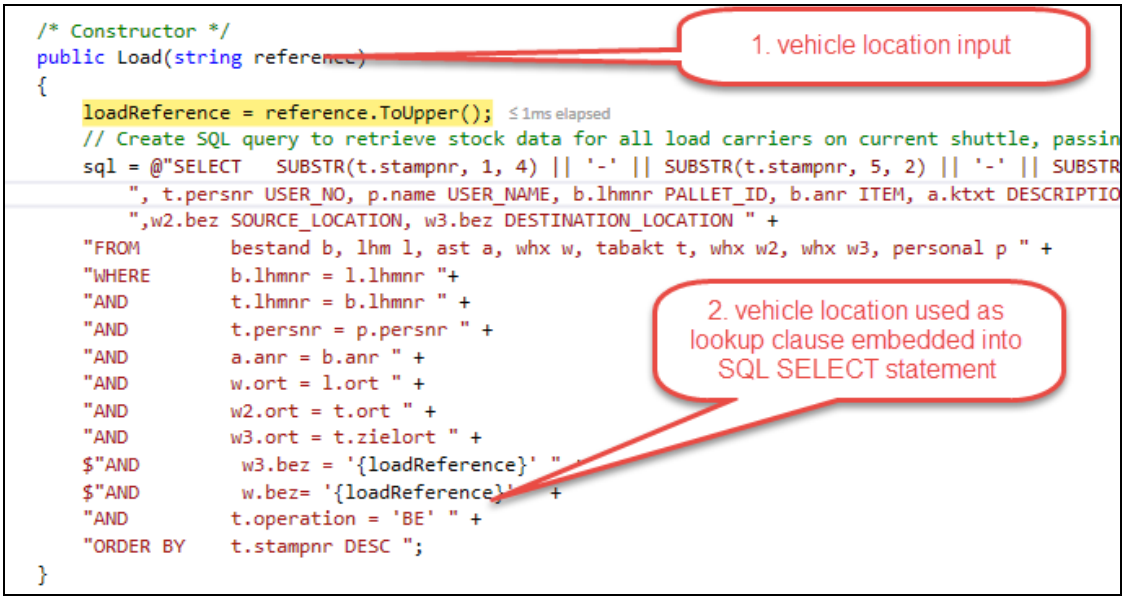
- ✓ Instantiating a new instance of the OraConnect class (Figure 32, OraConnect)
  - o Essential to retrieve stock record details from the UCP WMS database
- ✓ Populating the *pallets* List array with LoadCarrier objects for each stock record on the vehicle.
  - o This results in the Load object (the vehicle) containing members (LoadCarrier objects) in the *pallets* variable. The LoadCarrier class is an object representation of a stock record, designed to be a member of the Load class. The LoadCarrier object and associations are shown in Figure 32, with stock record properties such as *sscc*, *product code*, *quantity*, *status* and so forth (list is not exhaustive) class properties pertaining to stock records.

5.1.7 Records Query from Oracle (UCP WMS)

When the Load object implements the OraConnect class, an SQL statement is constructed embedding the vehicle input within the WHERE clause (Figure 33). This allows the query to lookup records that pertain to the current delivery vehicle.

A GetData() method is used, a member of the OraConnect. This method accepts an SQL statement as the argument and retrieves the SQL resultset into an Oracle DataReader within the OraConnect class. These results were transferred to a DataTable object, part of the System.Data C# libraries, which supported a critical functionalal need. It was discovered later that the use of a DataTable resultset can be seamlessly passed to a DataGrid gui object for presentation to a user (a critical component of the delivery validation process, see 5.1.9). Therefore, the DataTable object provided link between the Oracle DataReader resultset, it's calling object (the Load class) and the subsequent presentation of stock records in a DataGrid component in the GUI. This is discussed in the fllowing section 5.1.9.

Figure 33 describes the vehicle input reference (marked "1") utilised in the OraConnect class constructor to retrieve the necessary stock records pertaining to the delivery being performed (marked 2, utilised in the SQL SELECT statement's WHERE clause).




```
/* Constructor */
public Load(string reference)
{
    loadReference = reference.ToUpper();  ≤ 1ms elapsed
    // Create SQL query to retrieve stock data for all load carriers on current shuttle, passin
    sql = @"SELECT   SUBSTR(t.stampnr, 1, 4) || '-' || SUBSTR(t.stampnr, 5, 2) || '-' || SUBSTR
        ", t.persnr USER_NO, p.name USER_NAME, b.lhmnr PALLET_ID, b.anr ITEM, a.ktxt DESCRIPTIO
        ",w2.bez SOURCE_LOCATION, w3.bez DESTINATION_LOCATION " +
    "FROM        bestand b, lhm l, ast a, whx w, tabakt t, whx w2, whx w3, personal p " +
    "WHERE       b.lhmnr = l.lhmnr "+
    "AND         t.lhmnr = b.lhmnr " +
    "AND         t.persnr = p.persnr " +
    "AND         a.anr = b.anr " +
    "AND         w.ort = l.ort " +
    "AND         w2.ort = t.ort " +
    "AND         w3.ort = t.zielort " +
    $"AND         w3.bez = '{loadReference}' " +
    $"AND         w.bez= '{loadReference}' " +
    "AND         t.operation = 'BE' " +
    "ORDER BY    t.stampnr DESC ";
}
```

Figure 33: embedding the vehicle input to retrieve stock records

5.1.8 Add Stock Record Objects to Load class

The LoadCarrier class (see Figure 32) represents the stock records and associated properties. The Load object introduced previously fills the *pallets* List variable with new LoadCarrier objects for each record returned from the Oracle DataTable (see 5.1.7). Each DataRow member of the DataTable row is looped over, and for each record a new LoadCarrier objects is added to the Load class.

5.1.9 Delivery Validation

The Load class at this stage contains a List of LoadCarrier objects representing all stock records.

The Load class will now implement the DataGridWindow.XAML GUI component (. This is designed to present a view of the records being delivered. This class has been designed to accept several arguments. For future re-use of a DataGridWindow.XAML object, three of these arguments populate the window title, user prompt, and description elements of the window providing layout flexibility at run time. In addition, a DataTable object is also passed, the contents of which are used to populate the DataGrid GUI component. Finally, a set of param string[] arguments can be passed that are used to filter the DataGrid result set presented to the user. This "filter" functionality provides flexibility by enabling the developer to change the data grid output with ease – simply adding or removing a column reference as needed or requirements change. This was implemented as a result of user testing (see 5.4.5). Additional benefit is for efficiency in use of code. This means the DataTable object retrieved can be used for both populating the SQL Integrator Database (see 5.1.10) and due to the filter mechanism, the same DataTable can be used to load the DataGrid gui component, without worrying about showing unnecessary or private record data.

The resulting DataGridWindow is displayed to the operator, who must validate the prompt as described. The Cancel button returns to the main menu, with OK continuing delivery execution. Figure 34 provides an example of the filtered result set shown in one use case awaiting validation. These present a view of the records prepared in 5.1.1.

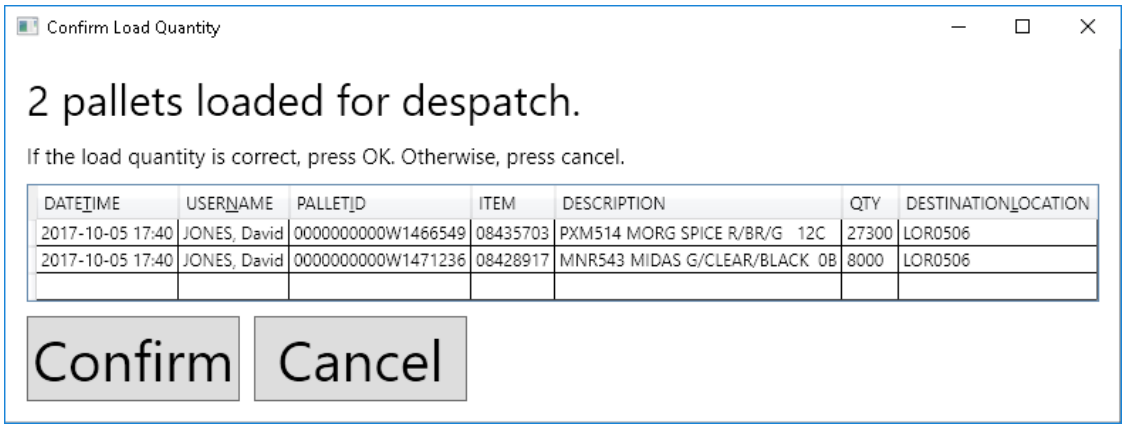


Figure 34: DataGridWindow for user validation

5.1.10 SQL Integrator Database

Following a successful validation, the rest of the data exchange is automated. First of all, delivery records are written to the SQL Integrator Database. Using the active Load object, the SqlConnect class is instantiated which provides these functions. The Load object properties are written to the

ShuttleHeader table (delivery vehicle information) and the Load object's *pallets* object (stock record contents held in LoadCarrier objects) are written to the ShuttleDetail table.

As the ShuttleHeader table record is written first, the subsequent ShuttleDetail records require the header ID to populate the detail table's foreign key (see 4.1.2). Problems were encountered during this process using complex SQL insert then subsequent select statements to try to retrieve the auto generated header ID. Thus, the use of SQL server Stored Procedures that can incorporate a dynamic return value (such as an ID column) have been implemented (statement shown in Figure 35). The use of the "SCOPE_IDENTITY" function in Figure 35 returns the database Id value generated from the previous INSERT statement. This is referenced from the application code to both insert the header record (result shown in Figure 36) and simultaneously ascertain the header ID value before populating the ShuttleDetail table in the same transaction (shown in Figure 37, with header ID). The resulting figures display the stock data presented and validated in 5.1.9 successfully loaded in the SQL Integrator Database.

```
INSERT INTO ShuttleHeader(LoadRef, DespatchTime,OperatorID,[Received Status]) VALUES
    (@LoadRef, @LoadTime, @OpId, @RecStat);
SELECT @id = SCOPE_IDENTITY();
```

Figure 35: SQL Integrator stored procedure with return value

The SqlConnect object provides the functionality for executing this stored procedure in Figure 35, using the System.Sql.Client SqlCommand object to reference the stored procedure and the return value requested. This is then used for writing the delivery record detail.

ShuttleId	LoadRef	DespatchTime	OperatorID	Received Status
105	LOR0506	2017-10-06 21:58:28.877	1	0

Figure 36: ShuttleHeader table insert results

Id	ShuttleId	SSCC	Time Loaded	Time Produced	Loaded By	Item	Quantity	Box Quantity	Status	Flag
58	105	0000000000W1466549	2017-10-05 17:40:00.000	2017-10-06 21:58:00.000	798058	08435703	27300	35	AVAILABLE	0
59	105	0000000000W1471236	2017-10-05 17:40:00.000	2017-10-06 21:58:00.000	798058	08428917	8000	1	BLOCKED	0

Figure 37: ShuttleDetail table insert results

5.1.11 Message Delivery

The final stage of a use case requires the packaging of a data message file for the stock records being sent to Malcolms. A new DataOutput class has been implemented to handle this specific function. The design assumptions were to extract directly from the SQL Integrator Database, but flexibility in code was required following discussion with Malcolms on their message file requirements. Their message requirements were extended form the original scope, to include additional data record properties. These could be null, but must be populated if they exist (see Table 13, machine and batch field). Due

to some uncertainty about the real need for these fields, and to avoid re-factoring code for somewhat fluid requirements in the future, a “writeFromSqlProcedure” method has been implemented that accepts a number of bool arguments. These arguments dictate if an optional field should be written to the message file. This method calls an SQL Stored Procedure, which returns a “full” result set, with the method code providing flexibility to include or filter specific fields as needed.

The FtpCon class (see 4.1.9) is used for passing this message file to an FTP server, the agreed message exchange platform between UCP and Malcolms businesses. The delivery message file relating to the records prepared in 5.1.1 is shown in Figure 38.

Figure 38: Delivery Message Output

5.2 WMS Integration Monitor (User Story 2 and 3)

User Story 2 and 3 have been implemented within the same application namespace, to be executed as a combined function. To avoid re-writing similar code this provides a more efficient approach to deployment. Both user stories represent many similar features (read message output streams from one WMS, perform business logic, write to a new integrator database, and output a message or a database update to the alternate WMS). As discussed in the design phase, many classes have been extended to keep common code in base classes.

Designed to run as a background process in its entirety, this application does not contain any discussion for a front end. Therefore, its results are based on purely inputs, business logic, and outputs.

In contrast to the manual data updates performed by administrators, a single use case of the new application is transferred from one system to the other in a matter of seconds. A small delay of up to 15 minutes may be observed between each use case due to the import / export schedule being limited to this frequency at the Malcolms business. (*Note: At the time of writing, the scheduler described is not yet in a production ready state. However, this was the most frequent schedule allowed and as advised by Malcolms at the time of the UCP application implementation*). Ultimately, this means there should be no more than 15 minutes between an update being replicated across WMS databases. Reducing the frequency of the scheduler at both Malcolms and UCP could reduce this delay if deemed necessary in the future. The rest of this section describes the following components of the implementation:

Section 5.2.1 introduces the execution of the process, the server it runs on, and trigger mechanisms.

Section 5.2.2 discusses the UCP WMS Output stream file handling setup. Measures have been implemented using an intermediary file handling process, to avoid breaking another interface that uses this output stream.

Section 5.2.3 describes how the WmsStream class needed further work in order to accommodate a solution that encompasses the requirements of User Story 1 and 2 (original class design in 4.3.2).

Section 5.2.4 and 5.2.5 provide a commentary and walkthrough on the data exchanges in both directions. Without a GUI front end, the walkthrough will be limited in terms of visual representations.

5.2.1 Application Execution and Trigger

The application is run directly on the UCP WMS cluster server, beneficial for two reasons: network overhead is minimised and the existing server implementation provides failover services. Running across two physical servers, this ensures failover redundancy is provided for software and database services, in the event of critical failure.

With the need to run as a fully automated background process, two options were considered with regards providing a trigger mechanism for any data exchange: poll the WMS output streams and execute on change, or run on a schedule. A schedule was chosen in accordance with some basic understandings; The Malcolms message file import / export tasks run on a schedule, every 15 minutes. Therefore, a live poll running at the UCP side would be redundant so far as the process is only as efficient as the sum of its parts. For example, any time gained by identifying "real time" UCP updates would be lost when the associated "write" operation was passed to the Malcolms WMS – the process would still stall for up to 15 minutes. Furthermore, running on a similar schedule ensures implementation can be easily achieved using the server's task scheduler service, with schedule timing changed easily changed by any systems administrator.

5.2.2 UCP WMS Output Stream file handling

Due to the UCP WMS Output stream being read and purged by a separate UCP ERP system (see 4.2.2), the application has incorporated a file handling process to avoid interfering with this existing critical business interface.

Currently, a scheduled task on the WMS server moves the WMS output stream file to a network share using a windows batch file. This network share is being polled by the business ERP system, where the file is consumed and processed for ERP purposes.

As the output stream provides both the trigger and the raw data being used in the implementation of the new application, this file must be intercepted without affecting the ERP's function. Thus, the same file must still end up being passed to the network share used by the business ERP in any new use case scenarios.

Therefore, a file exchange mechanism has been incorporated into the stream import classes that includes a method for handling this process. Replacing the existing process, the file will be passed to the ERP network share as part of the new applications data import sequence. This allows the output stream to be used by both the existing ERP interface and the new WMS to SQL Integration implementation. Simple System.IO library objects are used for copying, pasting and checking for file locks before moving.

5.2.3 WMS Message Readers

Implementation of the UCP and Malcolms WMS message readers involved significant refactoring of the original design concept (4.3 and 4.2). With user story 2 and 3 being merged into a single development, and each message import template having a different file structure, consolidation into a single class would increase complexity in these definitions threefold. Therefore, message readers have been split into an abstract base class "MessageStream" providing common methods and properties, and derived child classes for message specific items (column definitions, file URL, etc). The newly refactored reader classes replace the already refactored WmsStream base class design shown in 4.3.2. Commentary on the restructure is as follows:

✓ ***MessageStream*** becomes the new abstract base class

The master message reader class. Provides shared functionality for all permutations of the shared methods and functions for all message reader types.

✓ ***WhDeliveryMessages*** derived / child class

The derived class used for the delivery message file definition generated from the Warehouse WMS. Implemented for all warehouse delivery messages received (3.6.5)

✓ ***WhReceiptMesssages*** derived / child class

The derived class used for the receipt message file generated from the Warehouse WMS. Implemented for all warehouse receipt messages received (3.6.6)

✓ ***UcpMessages*** derived / child class

The derived class relating to the UCP status update messages contained in the UCP WMS output stream. Implemented for all UCP output stream messages received (4.2.2)

5.2.4 Walkthrough: UCP to Malcolms data exchange (User Story 2)

The application functions on the basis of a Quality Auditor inspecting stock and using the UCP WMS to change a stock record's status to either "blocked" or "available" (Figure 39 shows two records being blocked in the UCP WMS, stock Id 11646 & 11647) .

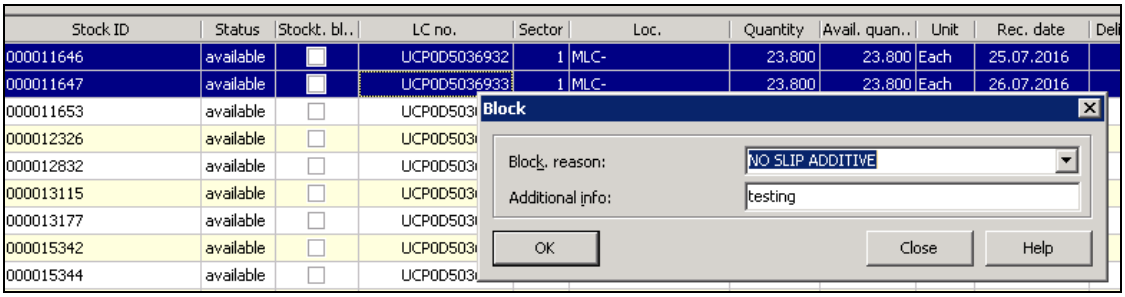


Figure 39: Native UCP WMS Blocking

At this stage, messages now exist ready to be read, processed, and passed to the Malcolms WMS.

Upon execution of the application, a new "UcpMessages" class is instantiated (see 5.2.3). This class has been implemented with two enum variables; characterPosition and fieldLengths. These are critical references that determine the stock record fields in terms of their position in the message file itself, and similar properties exist in all derived WMS message import classes. Each key field in the message file specifies its starting character position (Figure 40 "characterPosition" enum) and the field's length (Figure 40 "fieldLengths" enum) defined in each. This provides flexibility to incorporate any additional fields in the future, if a message file stream changes structure.

```
/* The following enums define the character position (field) start character,
and associated field length, in the WMS fixed data message file. */
enum characterPositions
{
    Type = 1,
    Year = 162,
    Month = 166,
    Day = 168,
    Hour = 170,
    Minute = 172,
    StockId = 348
};
enum fieldLengths
{
    Type = 2,
    Year = 4,
    Month = 2,
    Day = 2,
    Hour = 2,
    Minute = 2,
    StockId = 9
};
```

Figure 40: WMS Message Class File and Field Structure

In addition, a MessageStream base class method is called which moves the UCP message file into a "processing folder" where it will be stored until the file handling process is complete (5.2.2). As the UCP WMS output stream is always held in a file called "ConfirmStock.txt", to ensure that subsequent iterations of this process do not over write a *pending* file, a unique identifier is appended to the file name as it is imported. This ensures that a unique value at run time is added to the file as a suffix extension ensuring pending files are not overwritten by a subsequent data exchange, and therefore lost.

With the file in the procesing folder, a method getPendingBlockingMessages() in the UcpMessages object (Figure 41) iterates over the file contents, as each line in the file relates to one stock record update. For each stock record, a new StockData object is implemented for each stock record in the message file. This is represented in Figure 41. The StockData class is a new class that representes a stock record, and was added during the implementation process to improve the representation of

record updates in memory. Once all stock records are retrieved into memory, the source file is moved into an archive directory on the WMS server for future reference.

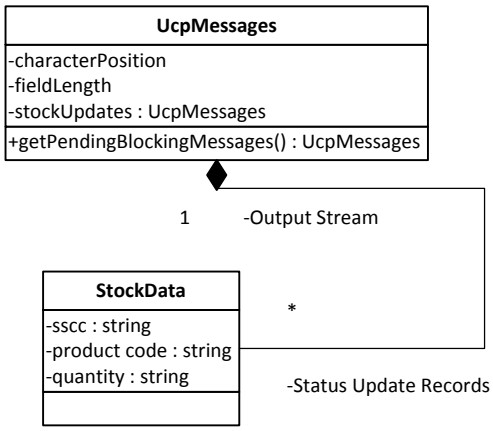


Figure 41 Message and Stock data objects

For each StockData object created (for every message / line of text in the message file, see representation in Figure 41) a connection to the UCP WMS database is opened using the Oracle Connection Layer. This is implemented in the StockData class constructor to populate record properties that do not exist in the WMS output stream, which only provides limited information. The Oracle Connection Layer contains a "getFieldFromStockId()" methods that accept two arguments – the stock record field to look up, and the stockId to perform the lookup against. Figure 42 (marked "Properties Assigned") shows an example of this StockData object constructor, with references to the method as oraCon.getFieldFromStockId(*string returnField, lookupvalue)* which return the requested stock record property to the object's stock record property in memory. This concept in the StockData object base class ensures that missing stock record properties in the WMS output stream are easily overcome by performing an oracle lookup as they are instantiated, using the Stock ID field which is included.

```
public class WmsStockData : StockData
{

    public WmsStockData(string stockId, string sourceFileName, int sourceFileLine)
    {
        OraConnect oraCon = new OraConnect();
        SqlConnect sqlCon = new SqlConnect();


        this.stockId = stockId;
        oraCon.connectionOpen();
        sscc = oraCon.getFieldFromStockId("b.lhmnr", stockId);
        location = oraCon.getFieldFromStockId("l.ort", stockId);
        locationDesc = oraCon.getFieldFromStockId("w.bez", stockId);
        item = oraCon.getFieldFromStockId("b.anr", stockId);
        oraCon.connectionClose();
        this.sourceFileName = sourceFileName;
        this.sourceFileLine = sourceFileLine;
    }
```

Properties Assigned

Figure 42: Stock Data Object Constructor

At this stage a list of stock records exist in memory in the form of StockData objects. Each StockData object contains properties of all the fields required to pass as an output to Malcolms. Referring to the workflow logic in 4.2.7, the message output file delivered to Malcolms is dependent on it's location. If the record is in transit, the output file must be sent as delivery message (3.4.5). If

the record has already been received and located at malcolm, s a status update message must be sent (see 3.5.5). Thus, each StockData object is iterated over via a loop, and using the Sql Data Access class a check is made to see if the stock record is in a pending delivery state. This check is therefore intrinsically linked to the new "delivery to warehouse" function in User Story 1 which enables such a check to be made. The SqlConnect class implements a method "checkPalletPendingReceipt()" which accepts the stock record's "load carrier" property as an argument. In turn this method calls an SQL stored procedure, "Get_Pallets_Sent_Not_Received". Figure 43 shows the definition of this procedure, accepting one dynamic input "@SSCC" (also known as Load Carrier reference). This dynamic input in the stored procedure query is populated by the method code. The procedure accepts this dynamic input and uses it to search the ShuttleDetail Table (4.1.2) for the stock record passed. If this is found, the delivery record's status flag is checked to see if it is still in transit. Thus, determining the type of message to output.

```
SELECT Id, ShuttleId, SSCC, [Time Loaded], [Time Produced], [Loaded By], Item, Quantity, [Box Quantity], Status FROM ShuttleDetail
WHERE Flag=0
AND SSCC = @SSCC;
SELECT @ShuttleId = ShuttleId
FROM ShuttleDetail
WHERE SSCC=@SSCC;
```

Figure 43: SQL Stored Procedure - Check Delivery Status

If the procedure (shown in Figure 43) returns true, the stock is still in transit, thus a delivery message is constructed (see Figure 38). Otherwise, this is packaged as a new stock status update message (message output shown in Figure 44). Finally, using the FtpCon class (4.3.2), this message is sent to the FTP message exchange server for import into the Malcolms WMS.

UCPSTOCKSTATUS20171008061621.csv

```
42,08/10/2017 17:39:00,000000UCP0D5036932,08569947,BL,AVAILABLE,BLOCKED
43,08/10/2017 17:39:00,000000UCP0D5036933,08569947,BL,AVAILABLE,BLOCKED
```

Figure 44: Status Update Message File

The associated records originally blocked in the UCP WMS (Figure 39) are now visible in the SQL Integrator Database table "Stock Status Detail" in Figure 45.

ID	Source_File	Source_File_Line	SSCC	Item	Status_From	Status_To	DateTime	Batch	Flag
42	confirmstock_20171008055509_636430821097762486.txt	1	000000UCP0D5036932	08569947	Available	Blocked	2017-10-08 17:39:00.000	08569947	0
43	confirmstock_20171008055509_636430821097762486.txt	2	000000UCP0D5036933	08569947	Available	Blocked	2017-10-08 17:39:00.000	08569947	0

Figure 45: Stock Status Update Records in SQL Integrator Database

5.2.5 Malcolms to UCP data exchange (User Story 3)

As a starting point, the FtpCon object must this time be used to download and then clear any pending messages that have been queued from the Malcolms WMS. This allows the message reader classes to access the message files for further processing.

Two classes derived from the base MessageReader class, WhReceiptMessages and WhDespatch-Messages are implemented with StockData member objects created for all stock records. This is the same process as described in 5.2.2 but using a variation of the MessageReader class that is aligned to the Malcolms message file format instead of the UCP WMS Output stream format.

In line with the requirements and design considerations, some stock record messages from the Malcolms WMS relate to stock records with a product type of "packaging". As these product types aren't managed by the UCP WMS, these must be filtered out, and considerable effort was involved in the implementation of this function (discussed further in 5.3.1).

To positively ignore these stock record updates, an additional table has been added to the SQL Integrator Database containing a master list of UCP's products (see 5.3.1). These products contain a product type, which can be used as a lookup to determine if the stock record object should be passed back to the UCP WMS. The population and implementation of this table is discussed below in 5.3.1 on page 83.

Finally, StockData objects that pass the product type validation are passed to a new method in the Oracle Data Access layer, which update the UCP source tables directly. The OraConnect class connects using an account that has been granted UPDATE and INSERT privileges to the History and Stock tables in the UCP WMS. This allows the necessary updates to take place.

5.3 SQL Integrator Database

The final SQL integrator database implementation underpins the above design and implementation function. The Shuttle Detail and Shuttle Header tables (Figure 46, highlighted green) support User Story 1 and provide the new delivery to warehouse data structures. The Shuttle Header table contains a log of all deliveries confirmed, with the associated delivery content in the Shuttle Detail table. As well as providing history reporting, these table structures also provide critical functionality to the User Story 2 implementation, where by a check needs to be made against "in transit" stock records (see 5.2.4). This is provided by maintaining a delivery status flag for each delivery record, only marking as complete when a Malcolms "Receipt" message is processed (part of the design and functionality described in 5.2.5).

The Status Update Header and Status Update Detail tables (Figure 46, highlighted blue) underpin the User Story 2 function and facilitate the data exchange and reporting needs for passing a stock status update to the Warehouse electronically.

The Malcolms Messages table (Figure 46, highlighted turquoise) contains a record of all messages received and processed from the Malcolms WMS system (see 5.2.5).

A Products table has been introduced (Figure 46, white highlight) which joins tables on their "Product" field as a foreign key. The Products table introduces extended properties relating to a stock record's product code, and allows the identification of "product types. User Story 3 relies on this relationship to lookup a product type to filter out updates that relate to packaging items (see 5.2.5).

Further discussion on these components are provided below.

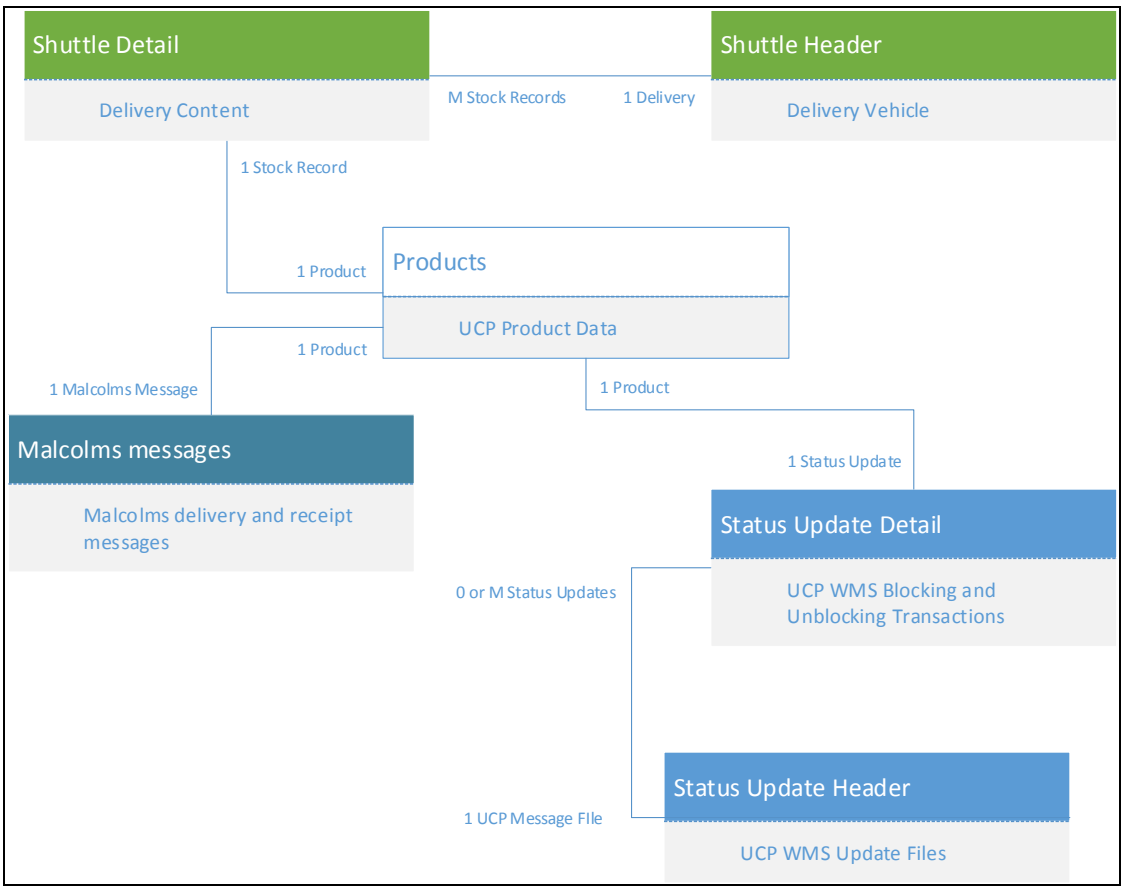


Figure 46: SQL Integrator Database - ER Diagram

5.3.1 Products Table

The products table, introduced above and shown in Figure 46 ("products"), provides functionality within User Story 3 to filter out Malcolms messages that relate to a specific product type of "Packaging". As a positive side effect, extended reporting can be performed directly within the SQL Integrator

Database due to additional stock record properties that can be retrieved from the product table. These properties do not exist in either WMS.

The product table contains records that relate to the "Product Code" (see 2.1.2 for background). The problem faced was that there are many thousands of product codes in the UCP ERP system that need to be input into the new database. Although these products exist in the WMS, they do not contain the properties required (such as "type"). Secondly, as new product codes are continuously being added to the companies JDE ERP system, someone would need to keep adding them as they are updated in JDE. Considering the project is about reducing manual data inputs, this measure would be counter-productive. Therefore, a second interface, from the business ERP system (JD Edwards) directly into the WMS Integrator Products table has been implemented to address this.

ERP side, following research, a simple JDE WorldWriter report has been created containing a list of all active product codes and properties. Using existing data export functions that are tied to the WorldWriter module, this report has been scheduled to run daily to a CSV output file, extracting all active product codes to a network path.

On the SQL Integrator Database, an SSIS (SQL Server Integration Services) package source has been created using the SQL Server Management Studio tool. This allows the construction of an import package that can import table data from a source file on a schedule, referencing the JDE report described above. Following the creation of the SSIS import package, a schedule has been added to the SQL Server Agent (an SQL server job scheduler) and the new job is highlighted in Figure 47. The job runs daily and automates the import of the latest product data.

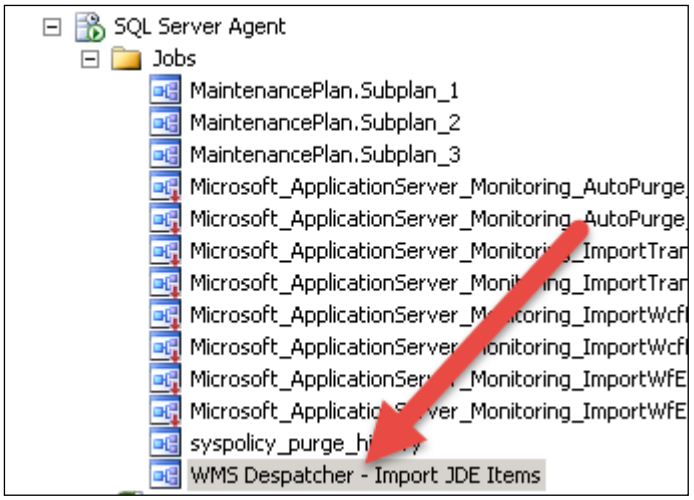


Figure 47 - SSIS Package Import - Products Table

To overcome a final complication, a secondary step was added to the import process to treat the product data imported from the ERP file extract. The WMS product code field includes a leading "0" (8 digit

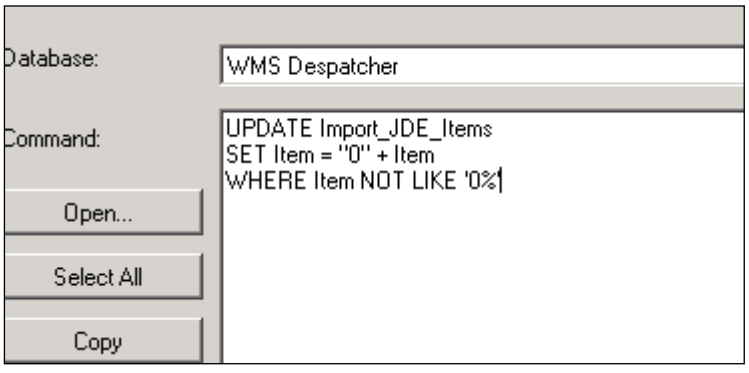


Figure 48 - Product Data Import Treatment

code) however the ERP extract removes this leading zero during the extract (7 digit code), and due to very limited controls in the ERP report writer, this could not be added at the extraction phase. Therefore, as part of the SQL Agent job, after the SSIS package is imported, a secondary step is processed which appends the leading zero, allowing the foreign key relationship to match the existing table structure (product code with a leading zero). This step in the SQL Agent Import process is shown in Figure 48, structured as an UPDATE statement against the import table, performed after the products table is refreshed from the JDE source file.

This results in an automatically maintained produce table, imported from the business JDE ERP system.

5.3.2 Reporting Capability

The implementation of the SQL Integration database has created a platform for delivery and message exchange reporting. Through the implementation of user story 1, delivery reports can be presented to end users involved in the logistics of stock management between UCP and Malcolms. This was previously kept in a paper filing system in the "Front Canopy" loading bay within the UCP factory.

Figure 49 shows just one example of the delivery reporting structure between the tables discussed and visualised in the ER diagram in Figure 46. The report shows a list of all stock records delivered on vehicle "LOR0101".

The figure shows the load carrier (SSCC column), and product code (Item column) which are both retrieved from the Shuttle Detail table (the stock records). The associated Product Description is retrieved from the Products table, and the Vehicle reference and Despatch Time columns from the Shuttle Header table. The SQL SELECT statement has been provided showing the use of the ShuttleHeader, ShuttleDetail, and Products table.

```
SELECT sd.SSCC, sd.Item, PRODUCTS.Item_Desc_1 [Product Description], sh.LoadRef [Vehicle], sh.DespatchTime
FROM ShuttleDetail SD
    INNER JOIN Import_JDE_Items PRODUCTS on SD.Item = PRODUCTS.Item
    INNER JOIN ShuttleHeader SH on SD.ShuttleId = sh.ShuttleId
WHERE sh.LoadRef = 'LOR1001'
```

100 %

Results | Messages

	SSCC	Item	Product Description	Vehicle	DespatchTime
1	350371330000001228	08512730	WNR543 JWRL R/G/ALLIANCE 0C	LOR1001	2017-09-03 10:07:58.177
2	350371330000001181	08512730	WNR543 JWRL R/G/ALLIANCE 0C	LOR1001	2017-09-03 10:07:58.177
3	000000UCP0D5039732	08557276	PXL372 BELLS RRP G/W/B/MAR 12C	LOR1001	2017-09-03 10:07:58.177

Figure 49: Delivery Reporting

Further permutations of delivery reports can be created as required using different fields or lookup criteria, such as delivery between dates, stock records delivered but not received, all deliveries loaded by a specific user, and so on.

The remaining message processing tables (Status Update, Malcolms Messages) can have similar reports configured for historical traceability purposes, although these will likely only be used by super users or administrators, as they relate to a background message exchange rather than a representation of a business function such as delivery reporting.

5.4 Testing

5.4.1 Functional Testing

Part of the "test driven development" concept introduced in 2.4.1, a series of functional unit tests have been maintained based on a set of business logic for each function. With each new function providing a workflow that processes inputs and creating an output, this logic was based on a set of standard workflow "permutations" (combinations of inputs and outputs).

5.4.2 Functional Testing – User Story 1 (Delivery to Warehouse)

User Story 1 maintained a set of functional tests that relate to the workflow discussed in 4.1.10. Four permutations exist, that result in either the workflow restarting with an error, restarting with a warning, or executing in full. These were driven by a combination of the user vehicle reference input, the Stock Records that relate to this vehicle input, and the result of the delivery being validated by the operator.

Table 24 describes each of these permutations. The column "Vehicle Input" relates to the vehicle reference input being valid reference (it is possible a mistake is made). The column "Stock Records" relate to the existence of stock records on the vehicle location, and the column "Validated" relates to the stock records being confirmed in the validation step of the workflow by the operator. The column "Expected result" describes the result that has been maintained throughout.

Table 24 - Functional Testing - User Story 1 (Deliver to Warehouse)

#	VEHICLE INPUT	STOCK RECORDS	VALIDATED	EXPECTED RESULT
1	Invalid	N/A	N/A	Restart with error - *user input error* - Vehicle reference does not exist in the UCP WMS
2	Valid	Not Exist	N/A	Restart with error – *user input error* – Vehicle reference exists but does not have any stock records.
3	Valid	Exist	False	Restart with warning – *WMS records error* – vehicle reference exists, with stock records on it, but operator aborts when prompted to confirm records

				are correct.
4	Valid	Exist	True	Execute Workflow in full

5.4.3 Functional Testing – User Story 2 (Status Update)

For User Story 2, the following business logic was maintained throughout. For any use case, five different permutations can occur. Table 25 lists each of these permutations, each numbered in the first column. The permutations are based on several factors, represented in the column values. The third column "Output Stream" relates to the existence of an output stream being present at run time. The fourth column "stock location" relates to the stock record being in transit, at the warehouse, or elsewhere. Finally, the "Expected result" column defines the output that must be maintained. Permutations of the above should result in either:

- ✓ Executing status update sequence in full, and passing a status update message file to Malcolms
- ✓ Executing in full, and passing the delivery message file to Maclolms
- ✓ Skip (stock record out of scope)

Table 25 - Functional Testing - User Story 2 (Status Update)

#	OUTPUT STREAM	STATUS UPDATE	STOCK LOCATION	EXPECTED RESULT
1	Not Exist	N/A	N/A	**Skip** – No output stream available.
2	Exists	Not Exist	N/A	**Skip** – Output stream exists, but does not contain status updates
3	Exists	Exists	In Transit	**Execute in full** – status update is on a record in transit. Output new delivery message file.
4	Exists	Exists	Warehouse	**Execute in full** – status update is on a record at the warehouse. Output stock status update message file.
5	Exists	Exists	Other	**Skip** – Status Update message exists but relates to a record out of the scope of shared UCP and Malcolms stock.

5.4.4 Functional Testing – User Story 3 (Warehouse Update)

For User Story 3, testing was maintained across each message type provided by the Warehouse WMS; Receipt messages and Delivery messages. For delivery messages, the permutations were simple; either the product type was in scope or out of scope (one permutation). However, for receipt messages, which also perform checks against the stock record's "transit" status in the SQL Integrator Database, combinations were defined and monitored.

Table 26 shows the three permutations applicable to the Receipt messages received. The permutations are based on the factors represented in the column values. The second column "product type" defines a receipt message being in or out of scope (5.2.5). The third column "in transit" relates to the stock record being marked as being on a delivery vehicle in the SQL Integrator database. Finally, the "Expected Output" column relates to the actions performed to either skip or update the UCP WMS database. These expected will be one or more of the following:

- ✓ Updating the record location in the UCP WMS.
- ✓ Updating the delivery status in the SQL Integrator Database.
- ✓ Skipping the record update.

#	PRODUCT TYPE	IN TRANSIT?	OUTPUT
1	In Scope	No	**UCP WMS Database Update:** Succeed. **SQL Integrator Database Update:** Skip.
2	In Scope	Yes	**UCP WMS Database Update:** Succeed. **SQL Integrator Database Update:** Succeed.
3	Out of Scope	N/A	**UCP WMS Database Update:** Skip. **SQL Integrator Database Update:** Skip.

Table 26 - Functional Testing - User Story 3 (Receipt Messages)

5.4.5 User Testing

The functional tests described in 5.4.1 are extended into a set of user test scenarios, which would ideally be attempted and managed by a test analyst in co-ordination with system users. Such resource is not available at UCP, therefore the existing software "Super User" (see Table 4 Critical Business Users: Roles and Responsibilities) has covered this role. With an understanding of the business operation, and an advanced user of the existing software tools, they have enough understanding to

perform the tests in context of all relevant factors. The business logic tests are formalised into a set of scenarios replicated by the end user from start to finish.

The idea to performing this stage as a secondary test phase, is that the functional testing should take care of the coding bugs, and the user testing can focus on exposing situations or scenarios that break the business logic already factored into the application.

In many circumstances, a significant proportion of a user testing phase would encompass application navigation use and feedback. Considering most of the application is executed as a background process, such checks are limited to operation of the Delivery to Warehouse application, which is designed to include minimal user interaction anyway. For the remaining "background process" tests, significant support had to be provided to the Super User in terms of articulating the scope of the test, e.g. "what and why" we are performing certain activities and checks in terms of the data exchanges tested.

User Story 1, Delivery to Warehouse, has been tested by the delivery operator as well as the UCP Super User, and feedback was gained on the ease of use, speed, and interface design. A relatively simple interface, the general feedback was extremely positive. The prospect of using the new tool to replace the existing admin procedures has been greeted with significant enthusiasm to progress. As a result of such tests, some changes during the implementation were made as a result. Cosmetic adjustments to the Main Navigation and Data Grid Layout window were achieved with simple amendments to the WPF XAML mark-up. In addition, user feedback drove more complex adjustments. The stock record properties shown in the DataGridWindow as part of the validation phase of a delivery were reduced by incorporating flexibility in to the method that populates the DataGrid element of the window. The implementation changes described in 5.1.9 were as a direct result of this feedback. Finally, an issue that still needs to be addressed, was the provision of a final "confirmation" that a delivery has been completed. At present, the delivery process executes and returns to the main navigation in seconds. This is almost two quick, and a simple extension to the application will provide such final confirmation that a delivery has been made.

User Story 2 and 3, the WMS to WMS message exchange application (5.2) was tested in accordance with the functional tests, but extended to a more "black box" approach where a test environment was provided on the Super User client PC, who could try to achieve expected results and "break" the functional logic built into the application. The WMS to WMS exchange mechanism was connected to the UCP WMS test database, and the WMS test database client installed on the super user's PC. Therefore, tests could be prepared and checked without impacting the production system. Limitations were encountered with regards message creation and import to and from the Malcolms WMS. These message files imported and exported are reliant on transactions performed in the Malcolms WMS. This functionality is not yet implemented, therefore dummy message files had to be manipulated to

perform tests, using samples provided by Malcolms as the template. Nevertheless, because of this limitation, the user testing exposed an unhandled scenario. When preparing dummy message files, the user noticed that the application accepted and processed the messages, but did not provide any feedback that the erroneous updates had been skipped entirely. Although erroneous data being generated is reduced when using an automated process, it is still possible. Therefore, it is deemed that such a scenario should be handled and written to the SQL Integrator Database, so an administrator can review such inconsistencies and follow them up. Other than this one exception, the update exchanges performed as expected.

5.4.6 Integration Testing

The WMS Delivery to Warehouse application has been deployed to both a virtual machine, and the PC used in the Front Canopy loading bay in the factory. The virtual machine has been used during functional testing, and the Front Canopy PC as the production platform being deployed to. Installation packages were created in the Visual Studio IDE, but following successful installation, errors were encountered upon execution of the deliver to warehouse function on both virtual and physical machines.

The GUI loaded OK, but at the point data was being retrieved from the UCP WMS, the application halted.

The first issue was related to Oracle data access components, part of the native UCP WMS application installer on PC clients. Debugging analysis was performed on the problem machines, and the issue was due to the components being missing completely. Following an install of the UCP WMS client, which includes these components, the original problem was resolved. However, a secondary error was observed in relation to user rights, when performing data retrieval on the UCP WMS. During development and functional tests, an administrator account had been used throughout. However, during the integration tests, a new user dedicated for the application interface had been implemented to avoid exposing the database from a “Shop floor” computer with admin access. Following data retrieval permissions being granted to the new user, stock records could be received on both the test and production machines.

Like the data retrieval issue, further security complications were encountered when trying to execute stored procedures for insert statements on the SQL Integrator Database. Significant research into SQL user roles and permissions were performed and with a more complete understanding of these functions, a dedicated WMS application group role has been created in SQL Server Management Studio, with the production accounts being members. This allows the “GRANT EXECUTE” privilege on the new account to those stored procedures referenced in the application code.

With regards to the WMS to WMS message exchange application (User Story 2 and 3) the integration tests were much more limited. The application runs as a background process on the WMS Server

itself. This runs under an administrative level account, so issues related to permissions were less obvious. Problems were experienced however, when trying to read messages from the FTP server (the platform where Malcolms messages are sent / received). Following many days of debugging and troubleshooting, the problem was eventually identified to a firewall restriction on the WMS server node. The code worked OK from the development machine, which runs with more relaxed network restrictions. However, a company policy implements strict access rules on Windows Server nodes. An exception has been made on the company firewall to accept traffic through port 21 from this server node, when the destination address is equal to the FTP server used by UCP and Maclolms for the message exchange purposes.

6 Conclusion

6.1 Summary

This project has encompassed the planning, design, and delivery of a bespoke WMS Integration solution designed to bridge the gap between UCP's factory to warehouse administration processes. The solution provides an automated solution to key business objectives; reducing non-value added time manually entering duplicate data into both systems; safeguarding customer satisfaction by delivering stock status updates before it is too late; and providing an extension of UCP's existing WMS functionality by integrating a dedicated WMS Deliver to Warehouse module, replacing paper records.

A full business process analysis has been performed involving cross functional departments in multiple companies. The problems have been highlighted, documented, and translated into demonstrable solutions. The problems that were unearthed exposed a situation that highlighted more issues than the operations teams originally expected, receiving positive feedback as a result.

The solution delivered has leveraged the C#.net framework to develop an "Inter-WMS" communication system, which runs seamlessly alongside the existing UCP WMS implementation. A new SQL Server Database instance has been implemented to underpin this resource and provide inter-WMS traceability and delivery reporting, something the business previously achieved through a delivery filing system. The solution has also been integrated with the business ERP application, to extend master data records that facilitate more advanced functionality.

The first module, the ***WMS Integration Monitor***, is a fully automated Console Application that monitors and exchanges in scope stock record updates across the UCP and Malcolms WMS implementations. The second module, the ***WMS Deliver to Warehouse*** application, provides new functionality to the UCP delivery operators to support the execution, reporting, and data exchanges associated to performing a delivery. Using a WPF interface the UCP delivery team can perform warehouse deliveries quickly and easily, replacing a manual process that took upwards of 15 minutes per delivery into seconds. In addition, management demands for data capture (delivery reporting) and delivery validation (process control) were incorporated into this module. Similar to the WMS to WMS Monitor, the application also provides a delivery message passed to the Maclolms WMS implementation. This message facilitates the creation of stock in the Warehouse WMS without the need for data entry on arrival. Following demonstrations to operational teams, excellent feedback has been received. There is an enthusiasm to press on with the project. However, it is expected that such a process change will provide cultural and training challenges, however simple, with some staff members used to a process that has been in place for many years previously.

Significant learning challenges have been overcome throughout the project and software development lifecycle. UCP's preference for Visual Studio C#.net development has meant significant research has been invested in expansion of existing skills, transferring introductory level JAVA OOP concepts to C#.NET, and the use of an advanced IDE for supporting the development has been challenging and rewarding. With the removal of "data duplication" being a critical deliverable to UCP, significant effort and time has been spent interrogating and interacting with both Oracle and SQL databases, with the SQL implementation also being built from the ground up providing the data structures for the new application. Such requirements have proven to be both challenging and enlightening, with the use of database programmability concepts such as stored procedures and functions being leveraged to overcome challenges in a way that was previously not understood. The Visual Studio IDE has proven to be an excellent tool at consolidating the key functions into a single environment – database manipulation, presentation layers, debugging and unit testing. Further investigation into the extended capabilities of the IDE will be a keen point of interest for future development in this area. The WPF presentation API, using XAML, have been leveraged to provide an interactive GUI for despatch operators, extending the functions of the existing WMS. The capabilities of this framework have only been discovered, but provide interesting possibilities for future projects.

6.2 Evaluation

The solutions delivered have by and large addressed the ultimate objectives of delivering automation in an area of the business that required significant manual efforts. However, a critical evaluation is provided which describes the areas that were addressed comprehensively as well as those that require further improvement or provide learning experiences for future projects.

As a critical pre-requisite, research into the existing business processes were performed with a view to supporting the requirements and design phases. The results are discussed in some detail in 2.1 and expanded into a set of problem categories. One of the most crucial stages of the project, it was also significantly challenging. The UCP business processes that were in place for handling communications and systems administration between themselves and their Warehouse supplier were varied and spanning several areas of the business. Many aspects were unclear without obvious roles and responsibilities for each stage. Gathering information raised many inconsistencies and contradictions which needed picking apart to disseminate useful information from noise. The business objective at UCP was to provide automation and this was clear, but it could be argued that had the manual processes been better understood by the people involved in executing them, and more closely monitored by those in charge of them, it would perhaps have caused fewer problems as a result. For this project, it highlights that operational procedures and training will be critical to successful and continued use of an automated tool.

Some interesting results were gleaned from industrial research in similar fields. A review of alternative solutions provided background on a range of business to business data translation software, some of which provided “out of the box” components that would address specific elements of the business problem at hand. However, with the problem spanning two distinct objectives to both extend WMS functionality and reporting of an existing system, combined with said data exchange mechanisms, a bespoke solution was delivered. This allowed the problem to be addressed directly that is made to measure without compromise and unnecessary “bloat”. For these reasons, a bespoke solution was the best fit for this problem. However, in the future it could be argued that business circumstances may change this position, with continuous evolution of requirements and business structure. Therefore, it must be noted that at some point, it is not beyond the realms of possibility that a commercial tool as such may be considered a feasible option if the business changes considerably. The implementation review in 2.3.1 discussed a complex data exchange process between a client and many internal and external systems, which use different message standards, schemas, and requirements. Using a BizTalk solution, the company leveraged built in API’s for cross protocol communications, providing core functionality out of the box. The cost and need for such an implementation at UCP is unnecessary, and also misaligned to the exacting requirement but if the business were to change significantly, incorporate many different warehouses, or suppliers and customers, such a tool may need to be considered and become a worthwhile consideration.

Reviewing the requirements definition, they have been well addressed, although not in their entirety. Those items that were deemed high or mission critical, have been addressed first and foremost. This may seem like an obvious approach, however, the associated functions for some of these priority items were also extremely time consuming to research and implement in comparison to some low priority items that could have been delivered quickly and easily. One such example is in relation to filtering Malcolms message updates based on product type (see 5.3.1). Many days were spent investigating a method to determine product type. With the lack of any built in lookup mechanism, a business ERP to application database interface was researched and implemented from the ground up. This involved researching new areas of ERP (associated report writing) to configure the extract, and SQL agent routines to schedule data refreshes. Although this succeeded, it played only a component part of a “high” priority requirement. Although the solution delivered is effective, considering the timeframe and scope this could have potentially waited until later, while other simpler more immediate requirements were delivered earlier. The one example that wasn’t delivered, is a requirement on the Delivery to Warehouse application in 3.4.4. This is to incorporate an email alert comprising the stock records sent on a delivery. As a lower priority item, this had been left until all research, design and development work on the high priority items had taken place, and has therefore been missed. On reflection, a more pragmatic approach to prioritisation could certainly be taken from any future project. Rather than evaluating a scope of work based purely on business priority alone, a second

consideration could probably have been made on effort required, which may have allowed more flexibility in fitting “smaller” jobs into the work schedule. With that being said, the objectives and requirements have largely been met comprehensively and provide a platform for future evolution.

From a technical perspective, much research was performed on the concept of “test driven development”, a component of Agile “Extreme Programming”. A brand new concept, the fundamental concepts were well understood but implementation proved difficult due to both lack of experience and how to actually implement the concepts described. The idea being to “test first” and “code second”, the plan was to create tests for a function, create the function, and make it pass, refactor, and so on. The idea is that Unit tests become a fundamental component of creating code, at the same time testing it. This proved difficult as the application expanded, with the utilisation of data layers and increasing object inheritance and membership. Revisiting this topic retrospectively, additional development concepts cleared up the confusion, clarifying errors in the approach taken. The use of mocking and class interfaces allows the decoupling of such objects and data layers, as the complexity of the application grows. This concept then exposes the real test driven development potential within the Visual Studio IDE. As a result, all unit tests were performed manually which meant business logic checks took longer than they could have. This will be an area that is revisited in the future, which should provide make development much more effective.

Finally, although the solution is demonstrable, critical final integration testing is still to be achieved in terms of ensuring messages are imported and exported in the Malcolms WMS database. With the project being undertaken on behalf of UCP, the context, scope, and focus of the work discussed is based on the UCP systems and processes, with Malcolms data handling needs based on providing or reading a message they provide. Therefore, in order to execute a ***full*** data exchange with the Malcolms warehouse, this message handling facility must be enabled. Due to project delays in the Malcolms organisation alone, their commitments to enable this message import / export facility has been missed, and still not yet available at the time of writing. So although the solution demonstrably delivers and reads these message formats following the syntax provided, the messages generated and read in the new UCP application cannot yet be imported and exported from the Malcolms WMS. Therefore, Malcolms WMS messages must be mocked in order to import into the UCP WMS. Thus, integration testing for the Malcolms WMS is not yet complete, pending actions out of UCP’s control.

Having visited the Malcolms warehouse with a UCP commercial manager to discuss the original requirements that were agreed, it was a surprise when UCP were advised of the delays half way through the project. Although the majority of the development work is on the UCP side, the Malcolms team have explained that the delay on their commitment is due to tight schedules and deadlines that are late, and for projects that pre-date the UCP request.

Following the most recent project update meeting to discuss this delay, UCP reminded the business that they had committed resources and delivered their actions as part of the agreement (the applications discussed in this research paper). UCP and Malcolms operational teams found the news disappointing, as this is considered to be one of the most important business improvement activities in progress. As discussed in 2.1.9, the implementation of this solution addresses business critical holes in the process related to maintaining accurate stock status, thus it is expected UCP will continue to keep the pressure on Malcolms in the coming weeks, in order to implement as soon as possible. With this in mind, a major learning can be taken from this experience. Should such a project present itself again, involving one or more external parties, careful consideration should be taken to understand how realistic all deliverables are, considering priorities and projects on going for all key players.

6.3 Future Work

Discussed in some detail in 6.2, the primary objective is to focus on progressing the go live date for the solution developed. This is entirely reliant on the Malcolms business enabling the message import / export facility as per their specification, and subsequently testing this successfully. It is still unknown at this stage if the Malcolms message import and export will function as expected, even though the application conforms to the specification provided; it is possible there may have been an oversight when Malcolms provided the specification format.

With the priority item being reliant on an external party to progress further, consideration can be given to the remaining work list. As a starting point, the delivery to warehouse application can be extended to include the incorporation of an email alert. As a lower priority request, it was the only item listed in the requirements file that was not delivered and demonstrable. .Net libraries exist that appear to achieve such functions easily, therefore should not pose too many issues.

In relation to functionality of the application developed, some concerns were raised during the user testing discussed in 5.4.5. Although the business logic performed as expected when passing Malcolms messages to the UCP WMS database, the super user managed to find a flaw when he filled the mock message file with erroneous data. The errors were handled but simply skipped the record, without any form of log. Although this situation is unlikely for a system generated file, it is possible that a Malcolms message file contains erroneous data of one form or another, and this will need to be followed up. Extensions of error handling and associated logging in the SQL Integrator Database will be added as a priority. This will also provide more advanced error reporting capabilities, which can be dealt with by exception.

UCP use a reports package, Crystal, which can be used to interrogate and report on many relational database management system. It is currently used to provide more advanced reports that fall outside of the capabilities of the built in WMS client reports. With the addition of the deliver to warehouse

function (5.1) a brand new set of reports are available. Users can not only view what deliveries have been made electronically, they can see which deliveries are still in transit pending arrival at the warehouse. Additional crystal reports will be added to the current reporting suite that provides this availability.

An additional “loosely” defined requirement was raised during design and development phases. Brought to the table by one of UCP’s key account directors, they discussed a request from one of their customers, Irish Distillers, who were starting to ask their suppliers to provide Advanced Shipping Notifications when they deliver stock. This is a similar concept to the new delivery message that is being sent to Malcolms when an operator confirms a delivery. However, this would be based on messages received from the Malcolms WMS, rather than initiated by the Delivery to Warehouse application. Further clarification will be required on this need but extensions to the existing logic will be reviewed and considered accordingly.

Finally, to address an inefficiency in the applications suitability for unit testing (introduced in 6.2), research will be performed on the refactoring of application code, strategically using interfaces to decoupled objects from one another and data access layers. Although the application functions well at present, such a strategy should prove beneficial in the medium to longer term by providing a solid foundation for further extensions, reducing the risk of interfering or disrupting existing functionality.

References

[1] GS1, *Serial Shipping Container Codes*, GS1 AISBL 2015. Available: https://www.gs1.org/docs/idkeys/GS1_SSCC_Executive_Summary.pdf [Accessed 03/08/2017]

[2] A. Troelson and P. Japikse, *C# 6.0 and the .NET 4.6 Framework,* Seventh Edition, Apress, New York, 2015.

[3] T. Toivonen and J. Siitonen, *Value stream analysis for complex processes and systems*. Elsevier B.V, 2016. Available: www.sciencedirect.com

[4] M. AlManei, K. Salonitis and Y Xu, *Lean implementation frameworks: the challenges for SMEs*. Elsevier, B.V, 2017. Manufacturing, Cranfield University. Available: www.sciencedirect.com

[5] J. Womack, D. Jones and D. Roos, *The Machine That Changed The World*. Free Press, 1990.

[6] D. Woods,. *Why Lean and Agile go together.* Forbes, 2010. Available: https://www.forbes.com/2010/01/11/software-lean-manufacturing-technology-cio-network-agile.html [Accessed 03/08/2017]

[7] J. Radhouane and D. Sundaram, *Inter-Organizational Information and Middleware System Projects: Success, Failure, Complexity, and Challenges*. Twenty-first Americas Conference on Information Systems, Puerto Rico, 2015. Available: https://pdfs.semanticscholar.org/85e0/d8bf3fccccaf7c5c9ad2a9a93472f9ccdaef.pdf [Accessed: 03/08/2017]

[8] Microsoft Azure, *What is Middleware?* Available: https://azure.microsoft.com/en-gb/overview/what-is-middleware/ [Accessed: 03/08/2017]

[9] Computer Weekly, *Choosing middleware software.* Available: http://www.computerweekly.com/feature/Choosing-middleware-software [Accessed: 04/08/2017]

[10] British Computer Society, *Bespoke vs. Off-The-Shelf software*. Available: http://www.bcs.org/content/conwebdoc/2767 [Accessed: 04/08/2017]

[11] S. Goel, H. Sharda and D. Taniar, Message-Oriented-Middleware in a Distributed Environment, In T. Bohme, G. Heyer and H. Unger editors, *Innovative Internet Community Systems*, pages 93-103, Third International Workshop, IICS 2003, Leipzig, Germany, June 2003 Revised Papers.

[12] A. Schill, *DCE-The OSF Distributed Computing Environment Client/Server Model and Beyond*. Intl. DCE Workshop, Germany, Springer-Verlag, Oct.1993.

[13] EDI Basics, *What is EDI (Electronic Data Interchange)?* Available: https://www.edibasics.com/what-is-edi/ [Accessed: 06/08/2017]

[14] G. Banavar, T. Chanra, R. Strom and D. Sturman. *A Case for Message Oriented Middleware.* IBM T.J. Watson Research Center, Hawthorne, New York. Available: https://pdfs.semanticscholar.org/c1c3/5adc71f20794be5598a476c54b2c1419bde7.pdf [Accessed: 06/08/2017]

[15] C.M. Robert and M. Micah, *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, July 2006.

[16] The CRM Team, *5 reasons why agile is better than waterfall.* Available: http://thecrmteam.com/5-reasons-agile-better-waterfall/ [Accessed: 18/08/2017]

[17] C.M. Robert and M. Micah, Chapter 3. Planning. Pages 49 to 60 In *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, July 2006.

[18] C. Schwaber, *Enterprise Agile Adoption*. 2007, 2008, Forrester Research.

[19] A. Begel and N. Nagappan, *Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study*, Proceedings of the 1st International Symposium on Empirical Software Engineering and Metrics, 2007.

[20] D. West and T. Grant, *Agile Development: Mainstream Adoption Has Changed Agility*. 2010, Forrester Research.

[21] G. Benefield, *Rolling out agile in a large enterprise.* Proceedings of the Hawaii International Conference on Software Systems (HICSS'41), 2008, pp.461-470.

[22] D. Lee and H.S. Yong, *Agile Software Development Framework in a Small Project Environment.* Journal of Information Processing Systems. 9. 10.3745/JIPS.2013.9.1.069, 2013.

[23] H. Kniberg, *Scrum and XP from the trenches. How we do Scrum.* C4 Media, 2015.

[24] M. Rapaport, *Messaging, Middleware, and EDI.* Sonic.Net. Available: http://sonic.net/~quine/middlewareEDI.html [Accessed: 18/08/2017]

[25] *EDI.* OpenText Business. Products and Services. Available: https://www.gxs.co.uk/products/edi [Accessed: 18/08/2017]

[26] OpenText B2B Managed Services. *Semiconductor Supplier Sharpens Competitive Edge*. Available: https://www.gxs.co.uk/resources/case_studies [Accessed: 18/08/2017]

[27] M. MacDonald, *Pro WPF in C# 2008: Windows Presentation Foundation with.NET 3.5*. 2nd edn. Apress, Berkeley (2008)

[28] A. Sohi, *Understanding the Basics of UI Design Pattern in MVC MVP*. 2011. Available: https://www.codeproject.com/Articles/228214/Understanding-Basics-of-UI-Design-Pattern-MVC-MVP [Accessed: 18/08/2017]

[29] I. Markovic, S. Pesic and D. Jankovic, Using XAML in Representation of Dental Charts in Electronic Health Record. In D. Davcev, and M.J Gomez, editors, *ICT Innovations 2009.* Pages 247-249, Springer-Verlag, Berlin, 2010.

[30] *Introducing Windows Presentation Foundation and XAML*. In A. Troelson and J. Japikse, editors, *C# 6.0 and the .NET 4.6 Framework,* Seventh Edition. Chapter 26, Apress, New York, 2015.

[31] S.D. Bart, *C# 4.0 Unleashed,* Chapter 5. Sams Publishing, 2011.

[32] I. Griffiths and A. Matthew, *.NET Windows Forms in a Nutshell.* Pp. 27-53, O'Reilly Media, 2003.

[33] Microsoft.Com. *XAML Syntax In Detail,* 2017. Available: https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-syntax-in-detail [Accessed: 18/08/2017]

[34] C. Jones, *Software Quality: Analysis and Guidelines for Success*. Thomson Learning, 1997.

[35] NASA. *Mars Climate Orbiter Mishap Board Report*, Phase 1, 1999.

[36] P. Palermo, Microsoft Developer Network. *Guidelines for Test-Driven Development* Available: https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx [Accessed: 18/08/2017]

[37] Microsoft Developer Network. *Unit Test Basics.* 2016. Available: https://msdn.microsoft.com/en-us/library/hh694602.aspx [Accessed: 05/10/2017]

[38] A.T. Majchrzak, *Improving the Technical Aspects of Software Testing in Enterprises.* International Journal of Advanced Computer Science and Applications, Vol. 1, No. 4, October 2010. Available: https://thesai.org/Downloads/Volume1No4/Paper%201-%20Improving%20the%20Technical%20Aspects%20of%20Software%20Testing%20in%20Enterprises.pdf [Accessed: 18/08/2017]

[39] E. Taymor, *An Agile Overview.* In The Agile Handbook, pp 5-6. Philosophie.Is. Available: http://agilehandbook.com/agile-handbook.pdf [Accessed: 18/08/2017]

[40] G. Boer, *Scale Agile to Large Teams*. Visu alStudio.com. Available: https://www.visualstudio.com/learn/scale-agile-large-teams/ [Accessed: 09/09/2017]

[41] D. Hellem *What is Agile Development?* VisualStudio.com. Available: https://www.visualstudio.com/learn/what-is-agile-development/ [Accessed: 09/09/2017]

[42] C.M. Robert and M. Micah, Chapter 2. Overview of Extreme Programming. In *Agile Principles, Patterns, and Practices in C#.* Prentice Hall, July 2006.

[43] C.M. Robert and M. Micah, Chapter 4. Testing. In *Agile Principles, Patterns, and Practices in C#.* Prentice Hall, July 2006.

[44] Microsoft, *C# 6.0 draft Language Specification*, Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/index [Accessed: 05/10/2017]

[45] Afdtek.com. *Case Study: Systems Integration*, Johnson Controls. [Accessed: 05/10/2017]

[46] Aspire Systems, *Professional Services – Enterprise Product Integration using Biztalk, ,* Available: http://www.aspiresys.com/casestudies/Case%20Study%20-%20Product%20Integration%20using%20Biztalk%20Server.pdf [Accessed: 05/10/2017]

[47] E. Ciurana, *Mule: A Case Study.* TheServerSide.com. Available: http://www.theserverside.com/news/1365047/Mule-A-Case-Study [Accessed: 05/10/2017]