
picoWMS Request for Comments 2.0

Released 1 August 2002

Note: *Table of Contents is at the end of the document.*

The purpose of this document is to provide the initial design guidelines for the picoWMS project in preparation for the generation of the specification and design documents.

This document is being provided in advance of the actual specifications in order to solicit comments from the community of users, consultants, developers and other interested parties.

picoWMS – What and Why

picoWMS is a highly-configurable warehouse management and control software package which will be available at no cost, and with full source code. The first complete installation-ready release is scheduled for 2Q2003, with updates and enhancements to continue thereafter.

While a completely off-the-shelf WMS that will appeal to everyone is nearly impossible to create, picoWMS plans to provide enough of the infrastructure, operational modules, and examples to allow consultants and computer/web-savvy warehouse IT groups to install their own systems.

Designed with growth in mind, picoWMS will allow the same software to operate a simple two person, paper-based system, and yet be able to grow into a high-end, optimized warehouse management system complete with bar codes, directed picking and optimized putaway with interleaved operations, cross-docking, ASN receiving, resource management, and much more. All of this will be done through a series of configuration changes, and addition of other open source modules as needed.

The initial configuration of the system is also simplified: a number of different user Case Studies (templates) will exist with pre-configured working systems available to test. Thus the warehouse manager could select one of the provided templates and immediately begin testing to see if the system meets his needs. It is expected that through the open source development path the code will continue to expand to meet the needs of most users.

Visit <http://openwms.sourceforge.net> for more information. You may also contact John Ribar, the picoWMS project manager, at john@picasso-software.com if you would like to help in the development, testing, documentation, support, or financial backing of the project. All of these opportunities are also discussed on the website.

Note: *We have acquired the picoWMS.com and picoWMS.org site names, and we will put them into use as soon as possible to reduce confusion ;-)*

Platform Requirements

There are two sets of platform requirements provided. The initial set is for the development environment, while the second set is for the operational installation.

NOTE: *I have made the following recommendations based on my experiences, research, and the responses I received from interested parties after the first revision of the RFC. These are not yet finalized. I still have a great desire to use Zope and Perl, but I think that the proper way (i.e., “right” for the long term) to build this is so that it can expand to the enterprise, and be able*

to interface with many other systems; Java will give us a better path, in my opinion, to this goal. As always, I am open to a good argument either way. I am also open to suggestions for the user interfaces – at this point I really like the VCL packages from Zaval (see website info below) – there is a Light-Weight one (LwVCL) and a “loaded” one (JVCL). Make your suggestions!

Development Environment

One goal of the picoWMS project is to develop and install the applications in an open-source environment. At this time, the recommended development environment is as follows:

- ❑ Development Language: Java 1.4 (J2EE) with Jikes Compiler for app generation
- ❑ Application server: Tomcat 4.0.4
jakarta.apache.org/tomcat
- ❑ Testing facilities: JUnit 3.7, JTestCase 1.2
junit.org, jtestcase.sourceforge.net
- ❑ Make facility: Ant 1.5
jakarta.apache.org/ant
- ❑ DBMS Interface: Torque or Castor [undetermined]
- ❑ Other technologies:
Enterprise Java Beans, Soap -- *jakarta.apache.org/soap*
User Interfaces – *Visual Components Library (JVCL or LwVCL) www.zaval.org/products/*
Internationalization Tools – *Zaval Java Resource Editor www.zaval.org/products/jrc-editor/*
- ❑ Recommended development environment: NetBeans (supplies access to Java, Ant, JUnit, Soap, Corba, and other emerging and standard technologies).
- ❑ Operating Systems: Linux, Microsoft Windows 2000™, Max OS/X

Operational Environment

Another goal of the picoWMS project is to keep the cost of the installation to a minimum for the end user, while providing the flexibility required in selecting a platform the will be “politically correct” for their organization. Therefore, the recommended installation environment is Linux. However, due to expected political resistance and support requirements, picoWMS will be designed to allow installation on Microsoft Windows 2000/XP and Max OS/X. Finally, any other platform that supports the Java development environment should be able to support a picoWMS installation. It is hoped that some of our partners will take this “port” upon themselves for the group.

Load balancing and failover will be handled by Tomcat, Apache, JBoss, or similar in a future revision of picoWMS, and will be discussed further in the specifications.

Database Platform

To keep initial costs down, and still provide a stable platform for installation, the open source installation will be based on MySQL and Postgres (that is, as two different types of installations, not as two databases used together).

In addition, again due to political and practical concerns, support will be provided for Oracle, Sybase, and Microsoft database platforms. The goal is to utilize a generalized database

interface (Torque and/or Castor) to allow simplified selection of a database based on the users needs.

Licensing

A thorough review of licensing options is not yet complete. The goals of the picoWMS project require a license that allows us to provide a package that remains free, allows for direction and control to be maintained by a project “board”, but which still allows the software be enhanced by anyone. Enhancements can be charged to a customer as hours (time), but the resulting new functionality should be returned to the code pool.

The only problems occur with proprietary processes... the desire is for all (most?) proprietary items to be stored in data, not code, so that nothing will be given away by sharing the code. For items that really need to remain proprietary, the source/functionality will need to remain with the client.

The current (preliminary) selection is the Artistic License (see www.opensource.org for details - it's not too long).

Interfaces

Real-time Equipment

- Automated Storage and Retrieval Machines
- Automated Guided Vehicles
- Programmable Logic Controllers (PLCs): Conveyors, Production Equipment, etc.
- Automatic Picking systems (A-Frames, V-Frames, etc.).
- Pick-to-Light, Pack-to-Light, Pick-to-Belt systems
- Scanners
- Weigh Scales

Other Systems

- RF Terminals
- Operator Interface units
- Shipping Vendors (Airborne, UPS, FedEx, etc.)

Provided Functionality

Receiving

- Receive Unexpected Items at Dock
- Receipt of ASN information from supplier
- Enter list of expected items when ordered and track receipts
- Track receipts against ASN

- ❑ Returns, expected and not
- ❑ Credit to order
- ❑ Return info to host
- ❑ Return to stock, damage (recovery), trash

Putaway

- ❑ Operator specifies destination
- ❑ No rules applied
- ❑ Rules prevent disallowed locations
- ❑ System gives destination
- ❑ No rules - first available
- ❑ Rules for zoning
- ❑ Rules for grouping and zoning
- ❑ Allow cross-docking
- ❑ Allow operator override

Inventory Control

- ❑ Simple reporting
- ❑ Maintain FIFO
- ❑ Maintain QC
- ❑ Maintain Lot Control
- ❑ Recovery of damaged items
- ❑ Trashed items
- ❑ Ordering of supplies and raw materials
 - Based on manual data entry
 - Based on forecasts
 - Based on actual usage

Order Entry

- ❑ Manual Order Entry
- ❑ Order downloads, specified format

Order Processing

- ❑ Allow priorities
- ❑ Ship based on time of receipt
- ❑ Ship based on ship date
- ❑ Partial shipment hold

- Partial shipments
- Cancel partials

Picking and Fulfillment

- Employee management
- Picking zone assignments
- Pick-only, replen-only, or both
- Pick-to-belt in zone
- Pick-to-light
- Pick-to-cart, case, or pallet
- Paper, batch term, RF, voice
- Pre-staging
- Pre-packs (kitting)

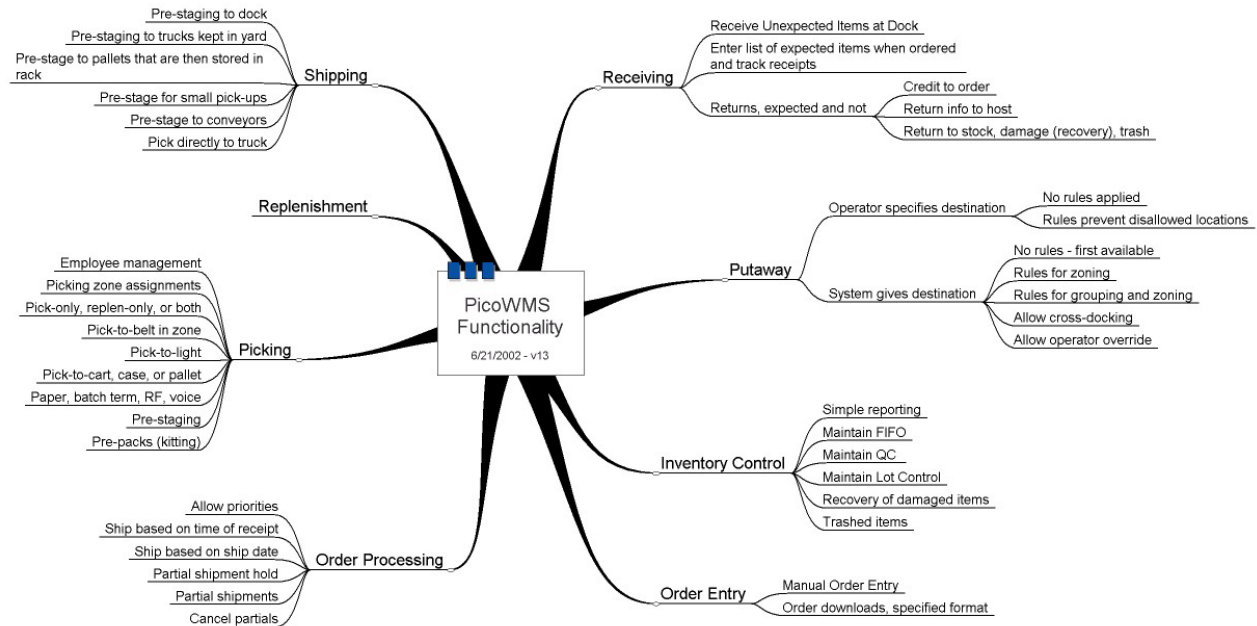
Replenishment

- Paper-based
- RF Directed
- Interleaved
- By Expectation

Shipping

- Pre-staging to dock
- Pre-staging to trucks kept in yard
- Pre-stage to pallets that are then stored in rack
- Pre-stage for small pick-ups
- Pre-stage to conveyors
- Pick directly to truck
- Internal manifesting system
- Interface to external manifesting systems
- ASN notification sent to recipient

Functional Drawing



Case Studies

The initial installation of this WMS is based on the concept of Case Studies. In our situation, a Case Study is the design of a warehouse with specific functionality. The modules of picoWMS can be combined in many thousands of combinations, but when someone first tries to install the software, this can be far too confusing. Thus, the case studies are pre-configured combinations available for an initial installation or test drive.

These are the initial ideas for Case Studies. Note that these are referred to as letters, rather than numbers, so that people don't infer a lesser system from a smaller number. Also, the titles are meant only as guidelines.

Case Study A – Single Zone Warehouse

This very basic level is a simple inventory tracking system. The operator tells the system where things are put, and can later look them up for retrieval. It has also been suggested that this could be used to track the simple putaway and retrieval of individual items in collections – stamps in multiple albums, portraits in a gallery, books in a library (maybe a little stretch here...), etc.

Case Study B – Single Purpose Warehouse

This is a typical starting level for a smaller warehouse. The system tells me where to put things away, based on matching products, FIFO, and/or groups (zones). The system has a manual order entry system for the entry of orders. Upon the release of the order by the operator, the system then selects items for the order, based on FIFO, LIFO, or other simple rules. The system tells the operator the location(s) and quality to pick for each item, and then prints a manifest list.

Case Study C – Standard Warehouse w/o Automation

ASN receiving. Forward and reserve pick locations. Directed movement of items from reserve to forward locations (replenishment) and to dock (pallet picks). Picking waves. Dock scheduling and management. Simple Cycle Counting under operator direction. Multiple warehouses/zones.

Case Study D – Standard Warehouse with Automation

Allow case/pallet picks from reserve area. Attribute based picking (country of origin, packaging, fancy picking methods (pick and pass, pick-to-belt, pick-to-light, pack-to-light, auto pick machine integration, picks waiting for replenishments, staged picks, cross dock picks) Returns processing. QA support. Integration with shipping companies – label printing (Airborne, UPS, Fed Ex, US Postal Service, etc.). “Continuous” picking strategies (waveless, dynamic, round-trip picking and replenishment, etc.). Cycle Counting integration with replenishment (zero counts).

Case Study E – Warehouse with Manufacturing Integration

Kit building, light manufacturing, QA integration, shipment building -- pack up that intermodal container just right! Work reporting (by task, employee, department, etc.). Scheduled cycle counts inserted into work queues.

Case Study F – High-Speed Optimizing Warehouse

Advanced attribute-based order processing. Integration with equipment. Vendor/supplier ASN shipping/receiving. Warehouse operations optimizations (work flow improvement, route planning).

Note: *Order processing* (getting the orders out of the warehouse) should not be confused with end-user *order entry* (getting the orders into the warehouse for fulfillment).

Database Design

This is very PRELIMINARY, and is included here only for completeness. There are several major things to be changed in this database layout, and they will be included in the specifications, so please just comment on things you want to be included, and not at how inadequate this version is... ;-)

Also, I KNOW this is not Java code! It is left to your imagination for now, since the language is not yet official. The next revision will include a "complete" SQL definition for the database.

```
/* -----
PRELIMINARY v0.1 Database Definition for picoWMS

This is the basic information about the pallet itself, not the
contents of the pallet. Those are maintained in the zContents
records.

Note that all database tables begin with 'z', and that a modified
Hungarian notation is utilized for variable/member names.

[LOOKUP] = foreign key reference
-----*/

struct zPallet
{
    char sPalletID[7];          // internal pallet number
    int iItemCount;           // number of ItemIDs on pallet
    char sLocationID[12];     // current location
    int iStatus;              // lockout, profile errors [LOOKUP]
    char sDestination[12];    // destination planned
    char sOrderID[7];         // batch/order assignment
    int iQCStatus;           // # of bad contents [LOOKUP]
    TimeStamp oPrevCountDate; // last cycle count
    TimeStamp oPrevChangeDate; // last change date
    char sPrevChangeUser[11]; // who made last change
    int iType;                // pallet type [LOOKUP]
};

// -----
// This is the information about the actual product being stored.
// -----

struct zContents
{
    char sPalletID[7];        // which pallet this is on,
                             // *<loc> if not on pallet
    int iSeq;                 // sequence on the pallet
    char sOrderID[7];        // batch/order assignment
    int iLineNumber;         // and associated line number
    char sItemID[21];        // which item this contains
    char sLotID[16];         // lot number for this item
    int iContainerCount;     // number of cartons/containers
    int iOrderCount;         // number containers assigned
    double dNetQty;          // total quantity/weight
    double dOrderQty;        // quantity/weight assigned
    int iQCStatus;           // QC status (stop work, critical) [LOOKUP]
    TimeStamp oStoreDate;    // when item was stored
    TimeStamp oProdDate;     // production date
    TimeStamp oExpirationDate; // when this one expires
};
```

```

    char sSerial[25];          // serial number
};

// -----
// This is the information about a specific lot.  This is different
// for each project.
// -----

struct zLot
{
    char sLotID[16];          // Lot number for this item.
    TimeStamp oChanged;       // Date/time of last change
    char sUserID[11];         // User (QC?) who made last change
};

// -----
// This is the item master file.
// -----

struct zItem
{
    char sItemID[21];         // item identification code
    char sScanCode[21];       // scan code for UPCs, etc.
    char sDescription[26];     // item description
    int iUnits;               // unit of measure          [LOOKUP]
    int iShelfLife;           // days til expiration
    char ucQCDefault;         // default QC status
    TimeStamp oLastMove;      // lst time item was delivered
    ULONG ulTotalCount;       // total count of containers/cartons
    double dTotalQty;         // total quantity/weight
    int iDefPalletCount;      // default container count per pallet
    double dDefContainerQty;  // default weight/quantity per container
    int iZoneID;              // default zone for storage    [LOOKUP]
    int iGroup;               // default storage group      [LOOKUP]
};

// -----
// Location file information
// -----

struct zLocation
{
    char sLocationID[12];     // location identification
    int iStatus;              // current status              [LOOKUP]
    char sPalletID[7];        // pallet currently stored here
    int iGroupID;             // group to which this loc is assigned
    int iZoneID;              // zone to which this loc is assigned
    char sItemReserve[21];    // item which must be stored here, if any
};

// -----
// Order header file, for both internal batches and external
// shipping orders.
// -----

struct zOrderHeader
{
    char sOrderID[7];         // order identification
    char sCreatedBy[11];      // who created the order
    TimeStamp zCreation;      // when order was created
    int iPriority;             // order priority
    char sReleasedBy[11];     // who released/activated order
    TimeStamp zReleased;      // when order was released
    TimeStamp zStarted;       // when order was started (delivery)
};

```

```

    TimeStamp zCompleted;    // when last item was delivered
    DATE zCompleteByDate;   // last date order can be completed
    int iStatus;            // current order status          [LOOKUP]
    int iType;              // order type                [LOOKUP]
    int iDestination;       // final delivery location    [LOOKUP]
};

// -----
// Order detail file. This is the actual line items for the order.
// -----

struct zOrderDetails
{
    char sOrderID;          // order identification
    int iLineNumber;        // line number on the order
    char sItemID[21];       // item that was ordered
    char sLotID[16];        // optional lot requirement
    int iCount;             // number of cases/containers
    double dQty;            // weight/qty required
    DATE zOldestDate;       // oldest date that can be used
    int iStatus;            // status of this item      [LOOKUP]
};

// -----
// User information - for each person that can log in to the system
// -----

struct zUser
{
    char sUserID[11];       // short name for logging in
    char sLastName[16];     // last name of the person
    char sFirstName[11];   // first name
    char sPassword[11];    // password, encrypted
    ULONG ulAccess;         // access rights bitmap
    DATE zExpiration;      // if needed, when does this expire
    int iStatus;           // user status              [LOOKUP]
    int iMaxLogins;        // max terminals that this can be
                           // logged into simultaneously
    int iCurrLogins;       // current simultaneous logins
    int iSuperUser;        // super user flag
};

// -----
// Information about specific terminals
// -----

struct zTerminal
{
    int iTermID;           // terminal ID number
    int iStatus;           // terminal status          [LOOKUP]
    ULONG ulAccess;        // terminal rights bitmap
    char sNetID[15];       // network address
    char sUserID[11];      // who is currently logged in
    int iAutoLogout;       // number of minutes of no activity
                           // before auto-logout
};

// -----
// Log file, maintains all the things that have happened.
// -----

struct zLog
{
    TimeStamp zLogStamp;    // when the logging occurred
};

```

```

        UINT uLogID;           // type of logging message
        char sUserID[11];      // person/process that generated log
        int iTermID;          // terminal where log occurred
        char sDatum[81];       // actual log info, based on uLogID
};

// -----
// Log ID types are defined in this file
// -----

struct zLogType
{
    UINT uLogID;           // type of logging message
    char sDescription[26]; // description of the message
    char sFormat[81];     // format of sDatum for this type
    char sHeadings[81];   // mini-report headings
    char sFieldList[81];  // list of fields in sDatum
    int iPriority;        // priority of this message 1=hi 9=lo
    int iSaveFlag;       // 1=save to database, 0=ignore it
    int iPrintFlag;      // 1=print to log printer, 0=don't
};

// -----
// Display information for log messages, based on priority
// -----

struct zLogDisplay
{
    int iPriority;        // priority of message
    int iPrinterID;     // which printer to use
    int iTermID;        // which terminal to use
};

// -----
// Process information, on the programs that are available
// -----

struct zProcess
{
    int iProcessID;      // process ID number
    char sProcessDescription[11]; // short description
    int iStatus;        // current status of the process
    int iTermID;        // where the process should be run
    TimeStamp oLastStart; // last time this process started
    TimeStamp oLastStop; // last time the process exited
    TIME oStartTime;    // auto start at this time daily
    TIME oStopTime;     // auto shutdown at this time daily
};

// -----
// Login sessions are tracked here
// -----

struct zSession
{
    int iTermID;        // where they logged in
    char sUserID[11];   // who logged into a terminal
    TimeStamp oLoginAt; // when they logged in
};

// -----
// Lookup tables - all with the same basic structure, but differing
// information
// -----

```

```

struct zZone
{
    int iZoneID;           // zone ID number
    char sDescription[26]; // description of zone
};

struct zGroup
{
    int iGroupID;         // group ID number
    char sDescription[26]; // description of group
};

struct zStatus
{
    int iStatus;         // status ID number
    char sDescription[26]; // description of status
};

struct zQCStatus
{
    int iQCStatus;       // QC status ID number
    char sDescription[26]; // description of QC status
};

struct zType
{
    int iType;           // type ID number
    char sDescription[26]; // description of type
};

struct zDestination
{
    int iDestination;    // destination ID number
    char sDescription[26]; // description of destination
};

```

Design Document (Preliminary Notes)

These are not ready for publication, but ready for comment. These are comments and ideas from various people who donated info for RFC 1.0, and which was not already included in the above portions of the document.

Document Content

Main chapters of design document:

- Receiving: Take possession of the goods
- Putaway: Determine where to put the goods and direct movement.
- Inventory control: Directed counts, adjustments, reporting, historical
- Order Processing: Get order into system (Get External order from sales), Determine when to process, allocate goods to order, Schedule picking
- Picking
- Shipping
- Replenishment

Appendix:

- System Case Studies
- Play/Test system
- Database model
- System Design: Task Processing Design
- System Configuration
- Simulation system
- Conversion from existing system

Applications – System Architecture

Servers

Application Server

The application server(s) will serve Enterprise Java Beans to all other applications, providing the required functionality in all areas where needed. Also connection to Database server?.

RF Manager

The RF Manager is a web server based on the GoAhead embedded open source web server package. The RF manager allows RF units to perform necessary processes, and allows for a minimal amount of program loading on the client. A DOS-based client (browser) is to be provided as well, so any DOS-based handheld unit will be able to use the system. **Note:** GoAhead is a good platform to keep the DOS browser simple. If we obtain/build a more robust DOS browser, I would prefer to manage the RF system through the normal web server. And in fact, we could probably do that with an Apache module or Java Bean, or even PHP or Perl, if someone is up to the task...

Operator Applications

User Program

Typical operator interactions

RF Clients

Receiving, putaway, retrievals (replenishment, QA), picking, queries.

Utility Programs

Paging Server and Clients

To page IT staff if problems arise.

Database Archiving

Store old data to CD/DVD for later retrieval in case of billing errors, recalls, etc.

Notes and Conversations

Picking, Configuration

[LVR>]When picking where does the pick path field go? The first answer is that a pick path is related to the location and should be thus in the location table. Well what if you pick in both directions? ok add two fields: "up pick path" "down pick path". better. Now what is the pick path relies upon the method or more likely the equipment you are using? Now the pick path is best put into the location_zone record where the zone is defined/configured as a picking zone functionality.

One of the ideas that I have kicked around (and haven't seen from major vendors) is a fully functional working demo on the net. Some will send out CD's etc.

It's also possible to run small operations on remote servers. This could also be a source for funding.

I have even seen "two man operations" inside large aerospace organizations.

One idea I have kicked around is to store the spec in a configuration application. The user goes through a number of question and answers and the system can state if it fits their requirements (or what version of the system!) and it will have also gathered information for the configuration process. [<LVR] **NOTE: Anyone done this? Or want to set it up for a trial? Or know of an Open Source app we can use like this?**

Service Side, Demos

[LJR>] Well, I hadn't wanted to get into the service end yet, but...

- 1) small systems run off a main server with VPN or direct websites with their own logins, etc.
- 2) automatic backups for people who still run their own servers
- 3) EDI-type connections between vendors?

I think a fully working demo is a great idea. I think the problem is that, despite everyone's attempts there is no way to create a fully off-the-shelf WMS. We've been trying for 12 years, and others have more time than that in. So a demo of something that can't be handed out is rather difficult. We can start with the base functions, which CAN be demonstrated. The customization is done after the install, if at all.

Also, I think the features should be marked as REQUIRED (rev 1), IMPORTANT (rev 2?), or COOL (rev 3, or whenever the fancy strikes), or something like that, so we can at least start with the absolutely necessary items first (i.e., product info, storage locations, etc.).

You will probably think I'm a little nuts until I get some of this on paper, but I've been working on the concept for a dozen years or more, and need to gather things into cohesiveness, and then I'd surely like more input. [<LJR]

Final Note

Taken from Dr. Frazelle's **World-Class Warehousing**.

These three operating decisions (slotting, batching, and sequencing) combined define the operating philosophy of any warehouse or distribution center and govern the overall performance of the operations. As a result, what should be three critical evaluation criteria for any warehouse management system? That's right - the way the system performs slotting, batching, and sequencing. These three decision operations applied to the main documents of the warehouse - purchase orders and customer orders - should be the foundation of any warehouse management system. From that core functionality should come the functionality for receiving, putaway, storage, picking, and shipping.

Contents

picoWMS Request for Comments 2.0	1
picoWMS – What and Why	1
Platform Requirements	1
Development Environment	2
Operational Environment	2
Database Platform	2
Licensing	3
Interfaces	3
Real-time Equipment	3
Other Systems	3
Provided Functionality	3
Receiving	3
Putaway	4
Inventory Control	4
Order Entry	4
Order Processing	4
Picking and Fulfillment	5
Replenishment	5
Shipping	5
Functional Drawing	6
Case Studies	6
Case Study A – Single Zone Warehouse	6
Case Study B – Single Purpose Warehouse	6
Case Study C – Standard Warehouse w/o Automation	7
Case Study D – Standard Warehouse with Automation	7
Case Study E – Warehouse with Manufacturing Integration	7
Case Study F – High-Speed Optimizing Warehouse	7
Database Design	8
Design Document (Preliminary Notes)	12
Document Content	12
Applications – System Architecture	13
Servers	13
Operator Applications	13
Utility Programs	13
Notes and Conversations	13
Picking, Configuration	13
Service Side, Demos	14
Final Note	14
Contents	15