Retek Retail passion. Innovative solutions.

Retek® Allocation™ 11.0

Operations Guide

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA

888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46

Fax:
+44 (0)20 7563 46 10

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction

Welcome to the Retek Allocation Operations Guide. The guide is designed so that you can view and understand key system administered functions, the flow of data into and out of the application, and the application's behind-the-scenes processing of data.

A retailer that acquires Retek Allocation gains the ability to achieve more accurate allocations on a stable product. Having the right product in the right stores allows for service levels to be raised, sales to be increased, and inventory costs to be lowered. By accurately determining which stores should get which product, retailers can meet their turnover goals and have increased profitability.

The Retek Allocation retailer benefits from the following capabilities:

- A Java HTML/JSP technology stack allows straightforward development and facile deployment. Debugging can be performed more rapidly; maintenance and alteration costs are kept low.
- Drivers map to different foundation data, enabling the application to be a flexible, stand-alone allocation system, able to integrate with RMS and retailer legacy systems.
- The application's interface takes advantage of Java database connectivity (JDBC), minimizing the number of interface points that need to be maintained.
- The application's robust algorithm executes rapidly.
- For retailers with other Retek products, integration with the Retek product suite means that item, purchase order, supplier, sales, and other data are accessed directly from the RMS tables, with no need for batch modules. Purchase order, item, location, and allocation information is passed from RMS to a warehouse management system, such as the Retek Warehouse Management System (RWMS).

What does an allocation system do?

A good allocation application enables retailers to make important decisions as close as possible to the time the product must be sent to the stores. A critical link in the supply chain process, the allocation process presents the final chance to distribute products successfully.

Retek Allocation enables retailers to take advantage of the most current, up-to-date sales and inventory information. Yet, the application also has the flexibility to allow allocations to be calculated months in advance for vendor commitment purposes.

Retek Allocation was designed to address the following challenges (among others) related to the correct allocation of product:

- How to put many a variety of merchandise plans into action.
- How to allocate product to support diverse marketing efforts and selling profiles.
- How to effectively and accurately allocate product without increasing head count while continuing to grow the business.
- How to streamline the training process for allocators, due to the position's high level of turnover.

If these challenges are not met, the wrong product can be sent to the wrong store in the incorrect quantity at the wrong time. The net result is higher markdowns, lower profits, and unhappy customers.

An overview of how need is determined

Retek Allocation determines the needs of each individual store at the SKU-location level through the following capabilities:

- The application sorts through copious quantities of data, such as sales history, current on-hands, and store volume groups.
- The application applies user-established rules, rule modifiers, and optional quantity limits.
- The application performs complex algorithms that can determine gross need for large volumes of stores and products, using real time data.
- The application applies constraints to the data, such as on-hands and on order, and determines the net need.

Who this guide is written for

Anyone who has an interest in better understanding the inner workings of the Retek allocation system can find valuable information in this guide. There are three audiences in general for whom this guide is written:

- Business analysts:
  - Those who are looking for information about the processes that enable the interface between Retek Allocation and a merchandising system such as RMS.
  - Those who are interested in how allocation data is calculated within Retek Allocation.
- System analysts and database administrators:
  - Those who are looking for information about Retek Allocation processes singly or in relation to the merchandising system.
  - Those who need to operate Retek Allocation on a regular basis.
- Integrators and implementation staff: Those who have the overall responsibility for implementing Retek Allocation.

What is not in this guide

This guide does not show you how to use the front-end of Retek Allocation. Rather, the focus here is on how data is managed, how it flows, and how it is processed.

This guide does not explain, except at a high level, the allocation-related data flow and processing that occurs among other applications across an enterprise (for example, the predictive planning system, the merchandising system, the price management system, the distribution management system, and so on). If you wish to find further information about how other Retek products handle allocation-related data, a list of applicable Retek documents is provided later in this chapter.

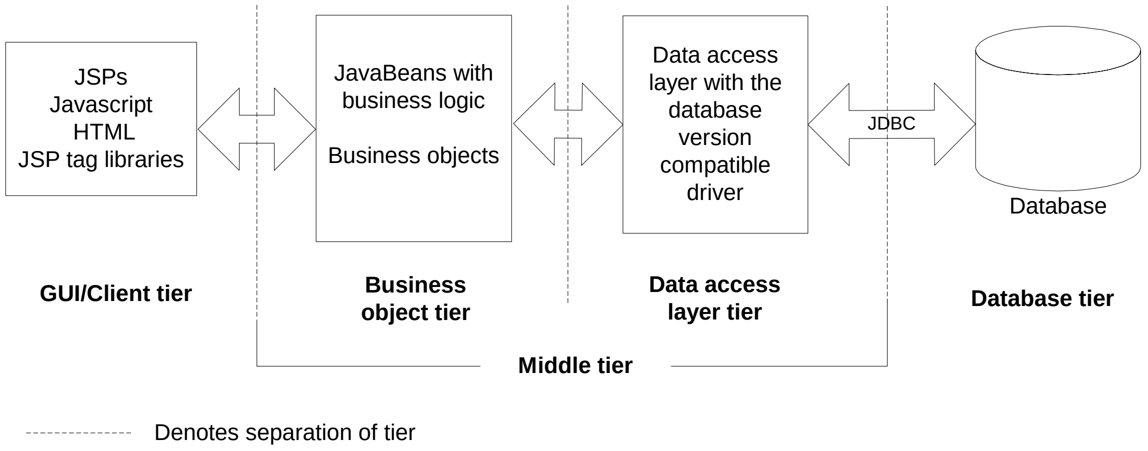
N-tier technical architecture overview

The following diagram provides a high-level overview of the general structure of the system, including the various layers of Java code.

The graphical user interface (GUI) is comprised of lightweight Java server pages (JSPs), enabling the GUI to adhere to today's 'thin-client' standard. JSP tag libraries are used for utility purposes.

The business object tier consists of JavaBeans, which contain all the business logic. The data access layer tier communicates with the database using a Java Database Connectivity (JDBC) protocol.

For more information concerning this diagram and Retek Allocation's technical architecture, see "Chapter 3 – Technical Architecture".



Retek Allocation's n-tier architecture

Where you can find more information

- Retek Allocation front-end documentation (for example, the Retek Allocation User Guide)
- Retek Allocation Installation Guide
- Retek Predictive Applications product documentation
- Retek Merchandising System product documentation
- Retek Warehouse Management System product documentation
- Retek Price Management product documentation
- RETL Programmer's Guide

Chapter 2 – Backend system administration and assumptions

This chapter of the operations guide is intended for administrators who provide support and monitor the running system.

The content in this chapter is not procedural, but is meant to provide descriptive overviews of the key system parameters.

Supported environments

This version of Retek Allocation has been certified on the following platform, with the following components:

- Operating system
  - AIX 5.2
  - HP-UX 11.11
  - Solaris 9
- Database version
  - Oracle 9.2.x
- Middle tier
  - Oracle 10G AS
  - OC4J 9.0.4
- Compiler
  - Java 1.4.x
- Browser
  - Internet Explorer 5.5 or higher

Supported version of RMS

This version of Retek Allocation is compatible with the following:

- RMS 11.0

Exception handling

Retek Allocation-related exceptions are handled through AllocException. AllocException is located in the following package:

- com.retek.alloc.utils

The following types of exceptions are wrapped by AllocException:

- SQLException
- Other checked exceptions

Errors are logged to the error log file, error_messages.log. For information about error logging, see the section, ‘Logging’, later in this chapter.

Allocation.properties file

A system administrator defines configurations for Retek Allocation in the Allocation.properties file. The key system parameters contained in this file are described in this chapter.

Minimum and maximum pool size to maintain

The pool size pertains to the number of available database connections that the retailer intends to keep available in the pool. A system administrator is encouraged to adjust these values per configuration to match the retailer’s anticipated number of users. The default values are intended to be a mere starting point. For more information, see the passage, ‘Pooling’, in “Chapter 3 – Technical Architecture”.

Logging

Logging files should be set up to valid directories, so that the retailer can generate logs regarding errors and messages. (For information about logging associated with RETL, see the section, ‘Message logging’ in “Chapter 6 – RETL batch processing”.) The example below shows the default Retek settings:

Note: For more information about the connection pool, see the passage, ‘Minimum and maximum pool size to maintain’, earlier in this chapter and see the passage, ‘Pooling’, in “Chapter 3 – Technical Architecture”.

```
# Log file for connection pool: one for Windows, and another for
UNIX

windows.pool.log=c:\\develop\\Alloc11\\oc4j\\j2ee\\home\\log\\connec
tion_pool.log

unix.pool.log=/files0/alloc11/oc4j/logs/connection_pool.log
```

```
# Log file for Error messages: one for Windows, and another for UNIX

windows.error.log=c:\\develop\\Alloc11\\oc4j\\j2ee\\home\\log\\error
_messages.log

unix.error.log=/files0/alloc11/oc4j/logs/error_messages.log
```

DEBUG mode on off switch

In a production environment, this setting should be set to false.

Start ship date for a purchase order (PO)

In a 'what if' scenario, the result can be a purchase order created in Retek Allocation (as opposed to a merchandising system such as RMS). Retek Allocation does not know the start ship date. Thus, this value has been added behind the scenes. The start ship date is derived from x days before the release date set in Retek Allocation. This number is pre-set to '3' days before the release date and can be changed by the system administrator.

Set the Automatic Update switch

Internally, Retek Allocation updates its location groups data based on the most current definitions. This update plays an important role when many months pass between initial and final allocations. The system administrator establishes this 'Yes' or 'No' value to instruct the system whether to automatically update location groups or not. Note that if a front-end user selects the 'Never Update' box, automatic updates do *not* occur even if the system administrator has established a 'Yes' value for this switch in the Allocation.properties file.

Date formats for specific locales

To provide a user-friendly date format that is understood by users, the system administrator may select one of four date formats that are available. The formats include the following:

- dd/mm/yyyy
- dd-mm-yyyy
- mm-dd-yyyy
- mm/dd/yyyy

Retek Allocation is sold worldwide and has been modified to meet internationalization and localization requirements. See the section, 'Internationalization and localization', later in this chapter.

Set the end of week day for the system

The system administrator establishes this value to inform the system what that end of the weekday is. Sunday is equal to 1, and Saturday is equal to 7. Note that this day must be identical to that set up in the merchandising system (such as RMS).

Minutes until a system unlocks from inactivity

This security feature is to prevent a user from walking away from the application and leaving behind an allocation in progress. After five minutes of inactivity, Retek Allocation returns the user to the Home page and unlocks the specific allocation that he or she was working on. The system is then available for the use of anyone with security access.

Bulk warehouse setting

When a user creates a bulk purchase order (PO) during a ‘what if’ scenario, the PO is cut to this designated warehouse. The retailer should make sure that this bulk warehouse is associated with a valid warehouse in the merchandising system (such as RMS). The default value is intended to be a starting point.

Flexible column definition

The system administrator initially establishes the default order and settings of the application’s columns. When a user customizes his or her windows, the result is saved and will continue to appear in that configuration until changed again. The default user ID (for flexible columns) is 1, which means that the user sees the column arrangement that Retek has designed.

Display future unit retail price values

The system allows the retailer to choose whether it wants the future retail price displayed on a specific screen. If a retailer uses future retail pricing to determine the future value of an allocation, this parameter should be set to ‘`true`’.

If a retailer creates or approves allocations without factoring in future retail pricing, the retailer should set this parameter to ‘`false`’. Future unit retail prices will not be available to be displayed on any screens.

Parameter:

```
# Indicate if allow user view future unit retail for item/store/release date in
# the allocation. When it is set to true, user can choose whether she/she
# wants the future retail price displayed on the allocation summary screen
# and allocation detail screen
enable_future_retail=true
```

Crossing legal entities

Using this parameter, retailers have the option to *disallow* allocations that cross legal entities. Legal entities are the locations in an organization grouped together due to legal requirements. Legal entities can be defined by brand, geography, country or some other grouping defined by the retailer. Issues in crossing legal entities can arise related to changes in cost and reail pricing, ownership, bookkeeping, and so on.

If the retailer selects ‘`Y`’, allocations *cannot* cross legal entities. Retek Allocation validates whether a warehouse/location combination is valid before processing. If a warehouse/location combination is *not* part of the same legal entity, the combination is skipped for processing. The system moves to the next combination.

Note: When a retailer selects ‘`Y`’, the system allows the user to select invalid combinations. The system does *not* process these when calculating need.

If the retailer selects ‘`N`’, allocations *can* cross legal entities.

Parameter:

```
# Indicates whether or not the user can cross legal entities
# 'Y' indicates Allocations can not cross legal entities
# 'N' indicates Allocation can cross legal entities
enforce_MLE=N
```

Bayesian sensitivity factor

Retek Allocation utilizes Bayesian forecasting in its Plan Re-project Rule. The sensitivity factor is described below and is pre-set at .3. A system administrator can change the setting anywhere from zero and one.

A higher sensitivity setting makes the forecast more reactive to actual sales, and a lower setting makes the forecast less reactive to sales.

A description of Bayesian sensitivity

Retek Allocation’s Plan Re-project Rule utilizes a Bayesian method to reproject the future dates of the plan. The rule takes sales history and compares it with the plan to create a forecast. This forecasting algorithm thus merges a retailer’s sales plans with any available historical sales in a Bayesian fashion (that is, the algorithm *uses new information* to update or revise an existing set of probabilities). A retailer would use this rule mid-season to allocate products based on actual sales results to date and planned sales.

Bayesian forecasting assumes that the shape sales will take is known, but the scale is uncertain. In Bayesian forecasting, when no sales history is available, the sales forecast figures are equal to the sales plan figures because there is no reason to mistrust the sales plan. As point of sale data becomes available, the forecast is adjusted and the scale becomes a weighted average between the initial plan's scale and the scale reflected by known sales history. Confidence in the sales plan is controlled by the amount of sales data on hand and a Bayesian sensitivity constant (which, as mentioned earlier, the system administrator can set from zero to one). Unlike standard time series forecasting, which requires only sales history to produce a forecast, Bayesian forecasting requires a sales plan and sales history (if available). As sales information arrives during the first few weeks of the season, the model generates a forecast by merging the information contained in the sales plan with the information contained in the initial sales data. These forecast updates can be critical to a company’s success.

For more information, see the section ‘Plan re-project algorithm’ in “Chapter 5 – Allocation calculations”.

Internationalization and localization

The technical infrastructure of Retek Allocation supports languages other than English. Retek Allocation has been subject to the modifications associated with ‘internationalization’, also known as I18N. (The I18N name stems from the fact that eighteen letters exist between the first ‘i’ and the last ‘n’ in the word ‘internationalization.’) Internationalization is the process of preparing software in order to ensure that it can efficiently handle multiple languages. In other words, the software is created so that it can be released into international markets.

Localization, also known as L10N, is the process of adapting software that has been internationalized so that it can be released into a local market with its own language. (The L10N name stems from the fact that ten letters separate the letter ‘l’ from the letter ‘n’ in the word ‘localization’.) Software is only internationalized once. However, software must undergo the localization process for every new language or location into which it is released.

This section describes configuration settings and features of the software that ensure that the base application can handle multiple languages.

Multibyte coding

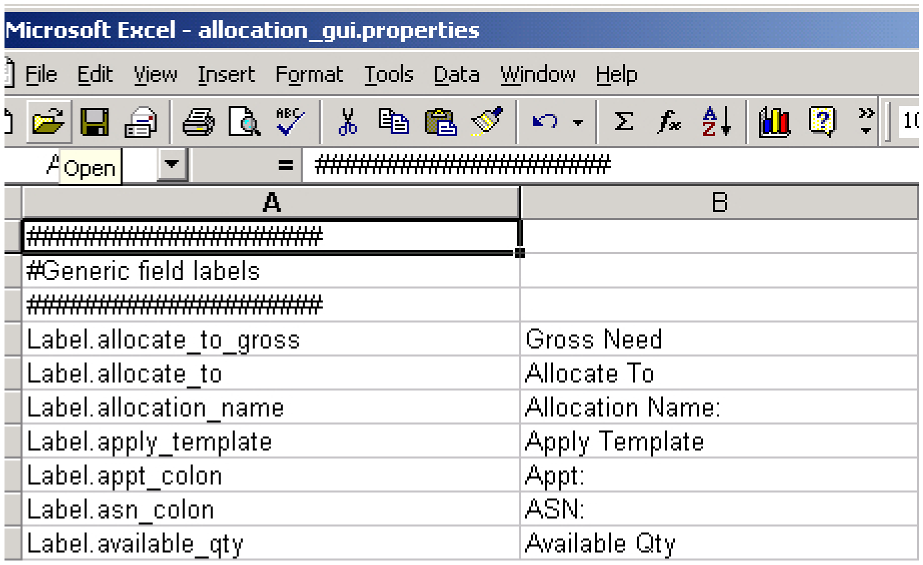
Retek Allocation has been developed to be compatible with multibyte languages (such as Japanese). In multibyte representation, a character may occupy more than one byte.

Interface text that is separated from executable code

An application that can run in various languages must be transformed into somewhat of a ‘generic’ product. That is, the features of the application that could be specific to just one language or locale (such as text, date formatting, and so on) must not be hard-coded into the software. Instead, locale-specific information is intentionally placed in files external to the application.

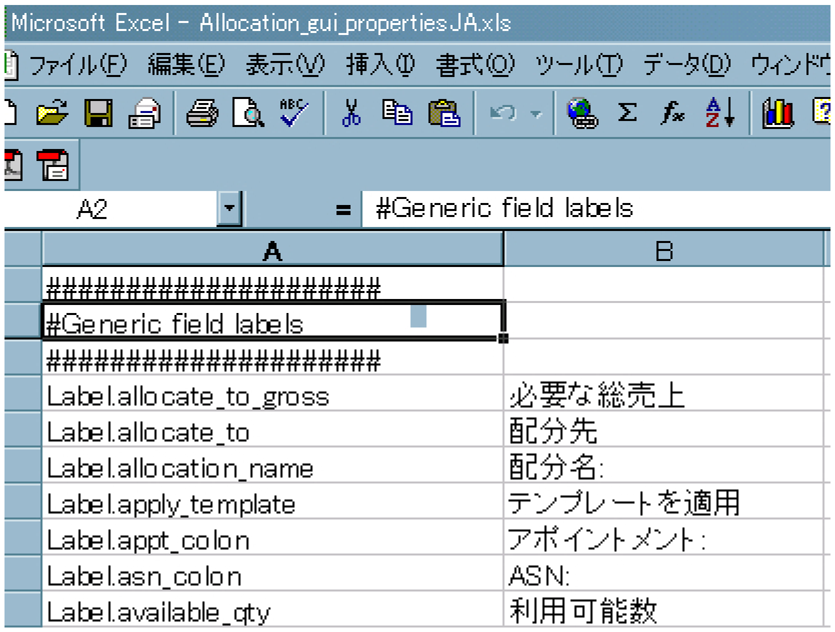
Much of what is locale specific in Retek Allocation has been pulled out of the code and placed into files. The content of these files is interface related, as distinct from executable code. The text in multiple allocation.properties files is translated so that the interface functions in local settings. These files comprise the interface layer. The allocation_gui.properties file contains the text within the GUI strings (for example, button names, menu names, title bars, and so on) that is translated.

The two images below, for example, display a small portion of two allocation_gui.properties files. One has been prepared for the English version of Retek Allocation and one for the Japanese.


Microsoft Excel - allocation_gui.properties
File Edit View Insert Format Tools Data Window Help
Open

A	B
########################	
#Generic field labels	
########################	
Label.allocate_to_gross	Gross Need
Label.allocate_to	Allocate To
Label.allocation_name	Allocation Name:
Label.apply_template	Apply Template
Label.appt_colon	Appt:
Label.asn_colon	ASN:
Label.available_qty	Available Qty

English version of the allocation_gui.properties file


Microsoft Excel - Allocation_gui_propertiesJA.xls
ファイル(F) 編集(E) 表示(V) 挿入(I) 書式(O) ツール(T) データ(D) ウィンドウ
A2
#Generic field labels

A	B
#####################	
#Generic field labels	
#####################	
Label.allocate_to_gross	必要な総売上
Label.allocate_to	配分先
Label.allocation_name	配分名:
Label.apply_template	テンプレートを適用
Label.appt_colon	アポイントメント:
Label.asn_colon	ASN:
Label.available_qty	利用可能数

Japanese version of the allocation_gui.properties file

Translation

Translation is the process of interpreting and adapting text from one language into another. Although the code itself is not translated, components of the application that are translated include the following, among others:

- Graphical user interface (GUI)
- Online help
- Some print documentation
- Error messages

Single executable

Because a single executable can handle multiple languages, the application can ship with multiple languages. Users can choose their preferred language 'on the fly' and can even switch languages when necessary or convenient.

Because only a single executable exists, maintenance efforts are minimized. The retailer does not have to recompile when switching from language to language. When patches are released, they only have to be applied once to the code and to the interface.

Date format preferences

To provide a user-friendly date format that is understood by the users, four date formats are available. See the section, 'Date formats for specific locales', earlier in this chapter.

RMS dependencies and assumptions

RMS differentiator setup

The RMS item structure allows multiple item levels and multiple differentiators. To structure item setup for use with Retek Allocation, the retailer must understand the implications of the Item Aggregate Indicator and the Aggregate Indicators that exist at the differentiator level.

The following section describes how an item family must be structured to enable the Retek Allocation product to differentiate the items among fashion, staple and pack items.

Fashion Item

RMS allows for the potential of three item levels. For a customer who allocates based on the concept of style/color, the 'style' can be translated to RMS item setup as being the level one item in the item family. The 'SKU' can be translated to RMS item setup as being the transaction level item (this could be level one, two or three). There is no requirement within RMS that forces a 'fashion' item to be multi-level.

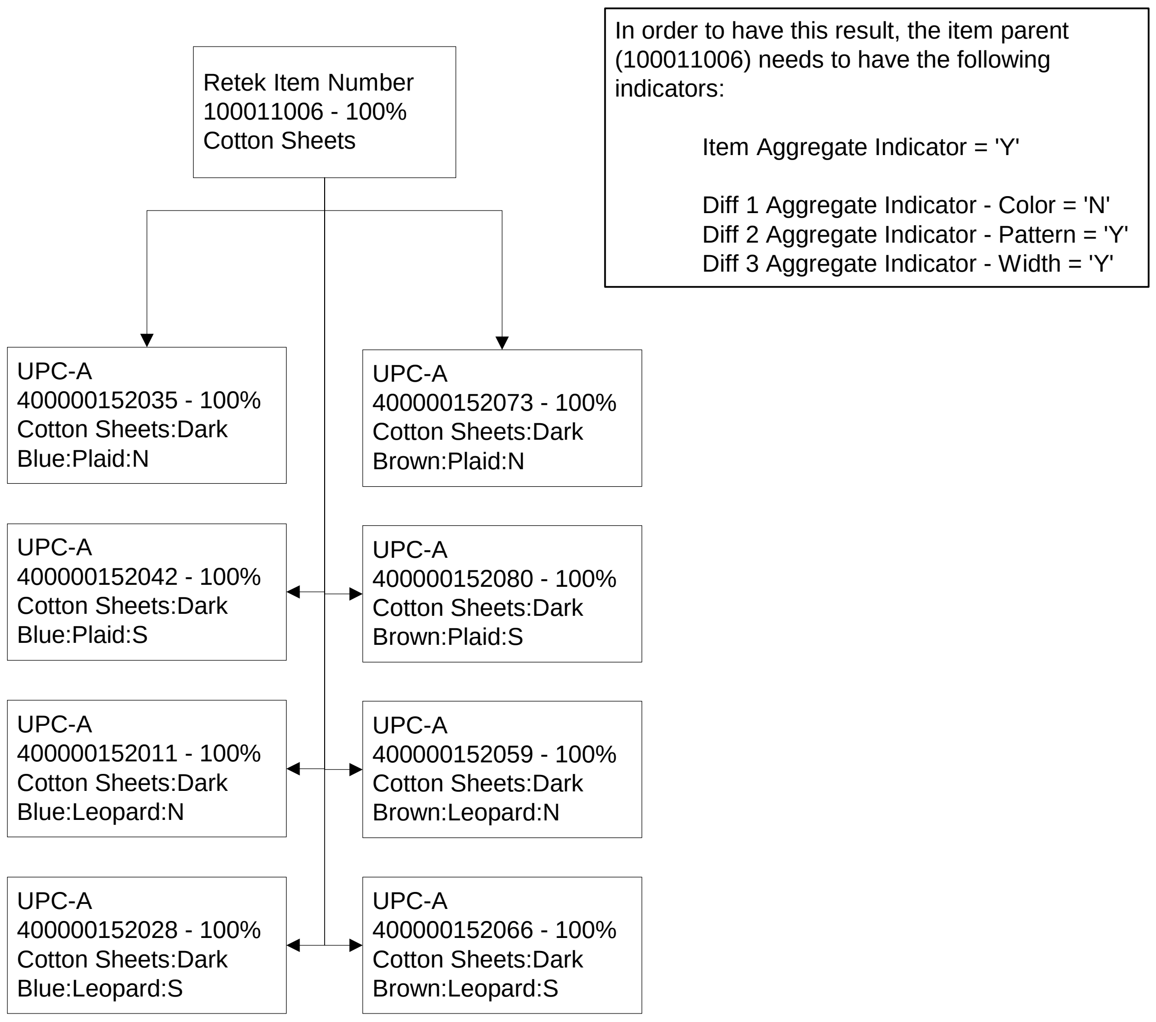
An item is viewed as a fashion item only if the Item Aggregate Indicator in the Attributes section of the Item Master Window is selected for the style (level one item) in the item family.

Once the item aggregate indicator has been selected, the user needs to indicate which differentiator should be curved by allocations. Each item may contain up to four differentiators. The Aggregate check box is enabled when more than one differentiator is being created for an item where the Item Aggregate Indicator has been selected. The differentiator that the customer wants to be curved by Retek Allocation must be the only differentiator that is *not* indicated on the Item Master Window.

Below is an example of a fashion item, its indicators within RMS, and what is visible.

Item 100011006 has three differentiators associated.

Color/pattern/width


Retek Item Number
100011006 - 100%
Cotton Sheets
In order to have this result, the item parent (100011006) needs to have the following indicators:
Item Aggregate Indicator = 'Y'
Diff 1 Aggregate Indicator - Color = 'N'
Diff 2 Aggregate Indicator - Pattern = 'Y'
Diff 3 Aggregate Indicator - Width = 'Y'
UPC-A
400000152035 - 100%
Cotton Sheets:Dark Blue:Plaid:N
UPC-A
400000152073 - 100%
Cotton Sheets:Dark Brown:Plaid:N
UPC-A
400000152042 - 100%
Cotton Sheets:Dark Blue:Plaid:S
UPC-A
400000152080 - 100%
Cotton Sheets:Dark Brown:Plaid:S
UPC-A
400000152011 - 100%
Cotton Sheets:Dark Blue:Leopard:N
UPC-A
400000152059 - 100%
Cotton Sheets:Dark Brown:Leopard:N
UPC-A
400000152028 - 100%
Cotton Sheets:Dark Blue:Leopard:S
UPC-A
400000152066 - 100%
Cotton Sheets:Dark Brown:Leopard:S

The retailer wants to have Retek Allocation apply the curve to Color. Therefore, it sees information within the Retek Allocation screens based upon the pattern and width differentiators.

All of the transaction level children will have their item and differentiator aggregate indicators = 'N'. These values are only maintained for the level one item. All other items in the system (including packs) have those indicators defaulted to 'N'.

In this scenario, if the retailer is creating an allocation for the parent item (100011006), it has visibility to four different levels of the 'style'.

100011006 - 100% Cotton Sheets Plaid:N

100011006 - 100% Cotton Sheets Plaid:S

100011006 - 100% Cotton Sheets Leopard:N

100011006 - 100% Cotton Sheets Leopard:S

Staple item

A staple item is every item in the system where the level one item in the item family does not have the Item Aggregate Indicator selected. In this scenario, the Retek Allocation retailer has visibility to the transaction level item only. There will be no roll up of item information as there is behind the scenes when the retailer is looking at fashion items at the style/differentiator level. The retailer also has visibility to the non-sellable packs that contain the component staple item and is able to include or exclude those packs from the allocation.

Pack Item

There are multiple types of packs that may be set up within RMS. The key criteria for Retek Allocation is whether the pack is sellable or non-sellable, whether the pack contains multiple component items and whether or not those multiple components items are of one type (for example, fashion as opposed to staple).

When creating your packs, consider the following pack assumptions made by Retek Allocation:

- Retek Allocation does *not* have the ability to allocate packs that contain fashion and staple items.
- Retek Allocation does *not* have the ability to allocate fashion packs that contain multiple item level one/differentiator (style/color) combinations.

Summary of items and how Retek Allocation handles them

- **Single staple item**
  These items are individually allocated and can be selected from SKU LOV search criterion.
- **Single fashion item**
  These types of items cannot be allocated individually. They are allocated as part of their style/color.
- **Style/color**
  The transaction level (SKU) items are allocated as visible in the View Assortment Window. However, the allocation is created at the item level one/differentiator (style/color) level. The item level one/differentiator (style/color) level is where retailers work with the allocation.
- **Simple sellable staple pack and complex sellable staple pack**
  These types of packs are included in an allocation when they are individually allocated.
- **Simple non-sellable staple pack and complex non- sellable staple pack**
  These types of packs are included in an allocation when the component of the pack item is allocated or when the non-sellable pack itself is allocated.
- **Simple sellable fashion packs and complex sellable fashion packs**
  These types of packs are included in an allocation when they are individually allocated. They are not be automatically included in any fashion items allocation.
- **Simple non-sellable fashion packs and complex non-sellable fashion pack**
  An allocation for this pack is performed behind the scenes. The user does not have visibility to the non-sellable pack allocation.

Retek Allocation functional assumptions

- The only way to allocate fashion items is by style/color.
- A single allocation cannot have both fashion item(s) and staple item(s).
- Non-sellable fashion packs are never returned as part of any search criterion that is visible to the user. Rather, they are handled behind the scene by the application at the style/color level.
- The SKU list of values on the search screen displays staple items, sellable/non-sellable staple packs and sellable simple/complex fashion packs.
- The store order multiple (SOM) for non-sellable staple packs is assumed to be one (1).
- The stop shipment record for a non-sellable staple pack must be at the component item level for the stop shipment to be recognized by Retek Allocation. A record for the non-sellable staple pack itself has no effect.

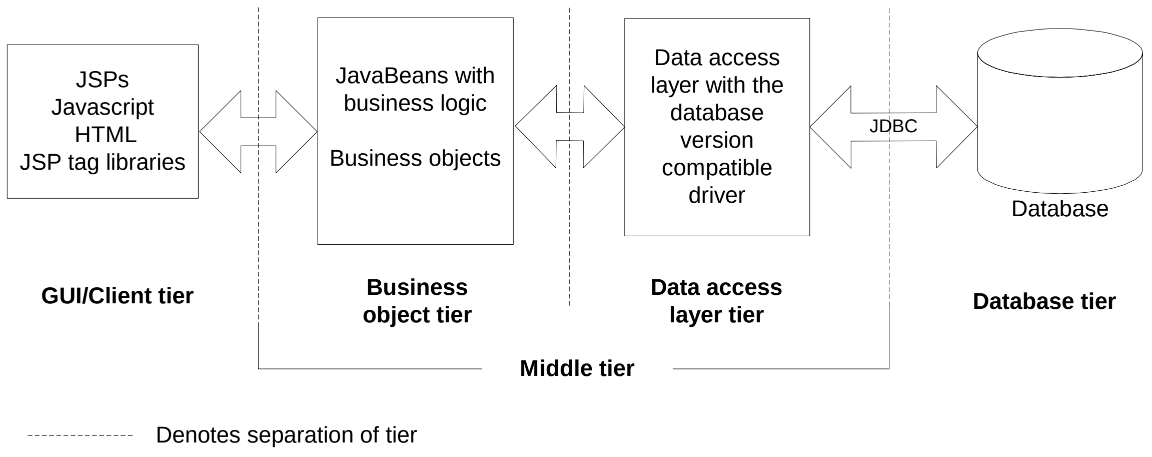
Chapter 3 – Technical architecture

This chapter describes the overall software architecture for Retek Allocation. The chapter provides a high-level discussion of the general structure of the system, including the various layers of Java code.

Overview

Retek Allocation utilizes a Java platform because it offers the optimum solution to the challenges presented by the need for database independence. A Java platform solves, for example, RMS version incompatibility issues.

The n-tier architecture of Retek Allocation allows for the encapsulation of business logic, shielding the client from the complexity of back-end systems. The following diagram, briefly discussed in “Chapter 1 – Introduction”, is explained below in detail according to each of the tiers shown in the diagram.



Retek Allocation n-tier architecture

Component descriptions and standards

Java Development Kit (JDK)

Standard Java development toolkit from Sun Microsystems.

Java Server Pages (JSP)

JSPs contain embedded Java and JavaScript within an HTML page. To the user, these pages appear in the Web browser as files with a .jsp extension. JSPs are part of Sun’s J2EE specification. They compile into servlets and allow for the separation of the user interface from business logic.

Java Servlet

Java Servlets are used for server side Java development, the Java Servlet is part of Sun’s J2EE specification.

JDBC

JDBC is a means for Java-architected applications such as Retek Allocation to execute SQL statements against an SQL-compliant database, such as Oracle. Part of Sun's J2EE specification, most database vendors implement this specification.

Naming conventions in Java

- Packages: The prefix of a unique package name is written in all-lowercase letters.
- Classes: These descriptive names are unabbreviated nouns that have both lower and upper case letters. The first letter of each internal word is capitalized.
- Interfaces: These descriptive names are unabbreviated nouns that have both lower and upper case letters. The first letter of each internal word is capitalized.
- Methods: Methods begin with a lowercased verb. The first letter of each internal word is capitalized.

GUI tier

The GUI is responsible for presenting data to the user and for receiving data directly from the user through the 'front end'.

Thin-client standard

The GUI adheres to today's thin-client standard. Whereas a 'fat' client can perform significant data validations and business processing on the client side itself, a thin client performs very little processing. Most of the application processing load is handled by the server.

Retek Allocation utilizes a thin client because of its advantages. First, there are no special requirements for the client-machine except that it can adequately run a browser. Secondly, client machines require little maintenance. That is, there is no need to install applications on each client machine because the application itself resides on a central server. Clients need only the browser to access the application. Finally, because standard HTTP is used, deployment can occur both inside and outside a firewall.

Java server pages (JSP) and HTML

The GUI is comprised in part of lightweight JSPs. JSP technology is a critical piece of Sun's J2EE-initative.

JSPs are compiled into servlets. JSPs also provide access to middle tier objects.

JSPs consist of JavaScript and standard HTML. They make calls to tag-libraries and contain minimal Java code. This code is located within standard HTML formatting tags. An extension of Java servlet technology, JSPs allow for the separation of the GUI's page layout from the application's content. The look and feel of the GUI is easy to customize, and dynamic functionality is easy to create.

As noted earlier, because the JSP/HTML GUI is 'lightweight' and uses standard hyper test transfer protocol (HTTP), the application can be deployed both, inside the firewall or outside the firewall.

JavaScript

JavaScript is used to handle some non-business rule validations. For example, JavaScript is responsible for the following:

- Date-entry validations
- Field-length validations
- Alphanumeric validations (for example, a US zip code cannot contain characters, and so on)

JSP tag libraries

JSP tag libraries are called for utility purposes. The use of tag libraries enables reusability. In other words, utility code does not have to be duplicated across all JSPs. For example, a paging tag allows pagination on any JSP page that refers to the paging tag. In addition, any changes that may be required can be made in one place.

Middle tier

Broadly speaking, the ‘middle tier’ consists of its own two tiers. The first is comprised of the JavaBeans that contain all of the business logic. The second is a data-access tier, which interacts with the code in the database tier and which contains database version-specific drivers.

Business object tier

Business objects implement business rules. A common business object infrastructure allows for the components to be utilized again and again within an enterprise. The business objects within Retek Allocation are represented as JavaBeans, which are, in essence, reusable Java classes.

Note: The ‘JavaBeans’ that Retek Allocation utilizes are *not* Enterprise JavaBeans (EJB). Retek Allocation does *not* use EJBs.

In terms of Retek Allocation, JavaBeans represent the logic of functional entities. Because the logic is wrapped into a component of software, it may be instantiated repeatedly. For example, in Retek Allocation, ‘item’ is represented as a JavaBean. Thus, as a JavaBean, any type of ‘item’ in the merchandising system becomes a reusable component.

Note that there is not necessarily a one-one relationship between a business object and a database table.

Business rule validations are handled by server-based middle-tier business objects.

Data access tier

This portion of the middle tier allows for business logic to be separated (physically and in the software) from the application’s presentation and database functions. Thus, the data access tier keeps the business logic and GUI isolated from any database issues.

JDBC protocol and drivers

The middle-tier talks with the database via the industry standard Java database connectivity (JDBC) protocol. JDBC facilitates the communication between a Java application and a relational database. In essence, JDBC is a set of application programming interfaces (API)s that offer a database-independent means of extracting and/or inserting data to or from Retek Allocation.

To perform those insertions and extractions, SQL code also resides in this tier facilitating create, read, update, and delete (CRUD) actions.

‘Drivers’ map to different foundation data, giving Retek Allocation the ability to be a stand-alone allocation system. These drivers account for differences in the way each database handles foundation data.

Pooling

When the application ‘disconnects’ a connection, the connection is saved into a pool instead of being actually disconnected. A standard connection pooling technique, this saved connection enables Retek Allocation to reuse the existing connection from a pool. In other words, the application does not have to undergo the connection process for each subsequent connection. Retek Allocation uses an open source connection pool called PoolMan that allows the maximum size of the pool to be configured.

Data storage tier

The database tier is the application’s storage platform, containing the physical data (user and system) used throughout the application. This tier is only intended to deal with the storage and retrieval of information and is not involved in the manipulation or in the delivery of the data. This tier responds to queries; it does not initiate them.

Accessing merchandising system data in real time

The data that Retek Allocation utilizes is located in both allocation-specific tables and merchandising system (RMS, for example) tables. Because Retek Allocation shares the same schema as the merchandising system (RMS, for example), Retek Allocation is able to interact with the merchandising system’s data directly, in real time.

A summation of n-tier architecture’s advantages

The following list is a summary of the advantages that accompany the n-tier architecture.

- N-tier architecture has become an industry standard.
- The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- The hardware and software for each of the tiers can be easily scaled.
- The look and feel of the application can be updated more easily because the GUI is not tightly coupled to the back end.
- Market-proven and industry-standard technology is utilized (for example, JSPs, JDBC, and so on).
- Component-oriented modeling promotes the reuse of code, saving development time.

Chapter 4 – Functional design

This chapter provides an overview as to how Retek Allocation is functionally integrated with other systems (including other Retek systems). The discussion primarily concerns the flow of allocation-related business data across the enterprise.

Overview

The first section in this chapter provides you with a diagram that shows the overall direction of the dataflow among the products. The accompanying explanations are written from a system-to-system perspective, illustrating the movement of data.

The second section in this chapter illustrates how need is determined on a functional level. That is, this section describes the source (at the table level) of the business data that Retek Allocation utilizes to generate its need calculation and shows the sources of that data.

The third section in this chapter describes the planning table that resides in Retek Allocation.

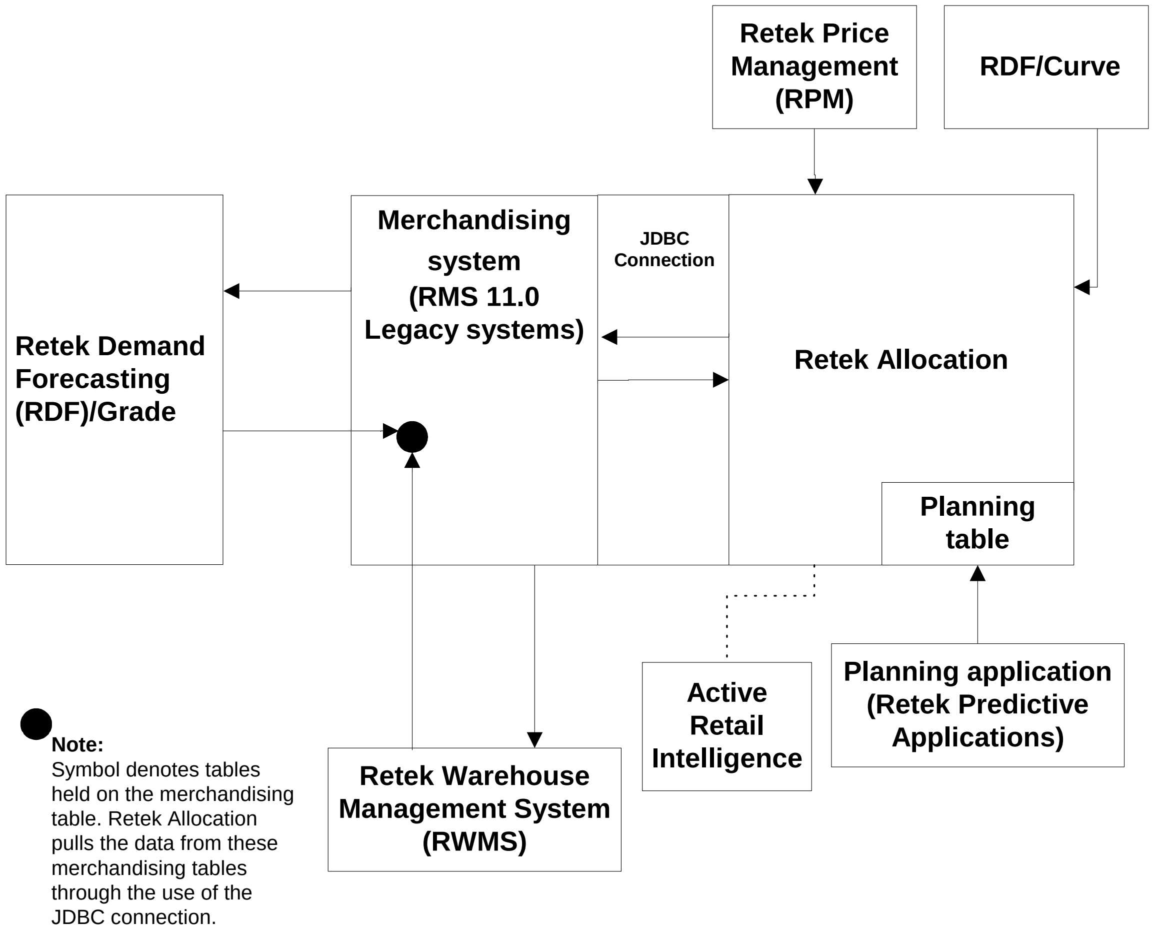
The fourth section in this chapter lists the merchandising system tables by functional area that Retek Allocation uses. The RMS 11.0 tables are provided, along with a generic list of legacy system tables that Retek suggests be made available for retailers who have not implemented RMS.

A note about the merchandising system interface

Many tables and functions within Retek Allocation are held in common with the Retek Merchandising System (RMS). This integration provides the following two important benefits:

- The number of interface points that need to be maintained is minimized.
- The amount of redundant data (required if the rest of the Retek product suite is installed) is limited.

Integration interface allocation-related dataflow


Retek Price Management (RPM)
RDF/Curve
Merchandising system (RMS 11.0 Legacy systems)
JDBC Connection
Retek Allocation
Retek Demand Forecasting (RDF)/Grade
Planning table
Active Retail Intelligence
Planning application (Retek Predictive Applications)
Retek Warehouse Management System (RWMS)
Note:
Symbol denotes tables held on the merchandising table. Retek Allocation pulls the data from these merchandising tables through the use of the JDBC connection.

Retek Allocation-related dataflow across the enterprise

Note: To facilitate a discussion of the diagram above, the text below specifically refers to Retek Demand Forecasting (RDF) and to Retek's products, Grade and Curve. However, a retailer without these Retek products can provide the type of data that Retek Allocation utilizes.

From the merchandising system to RDF/Grade

The merchandising system sends the following information to RDF/Grade:

- History data

From RDF/Grade to Retek Allocation via the merchandising system

Within RDF/Grade, the history data is subjected to processing that yields data that is sent back to the merchandising system. From there, Retek Allocation pulls the following data:

- Forecasting data
  Retek Allocation accesses forecasting data that originates in the Retek Demand Forecasting (RDF) system. RDF is Retek's statistical and causal forecasting solution. It uses state-of-the-art modeling techniques to produce high quality forecasts, with minimal human intervention. RDF is an application that resides on the Retek Predictive Application Server (RPAS). Retek Allocation uses forecasting data as a basis for calculating gross need and can access the following five levels of forecasting data: department, class, subclass, style-color, SKU.
- Store grade groups data
  Retek Allocation accesses store grade groups data that originates in Grade. Grade is Retek's application that groups store locations together intelligently based on similarities in performance, customer type, geography, or some other factor that allows the stores within each group to be treated as one unit. Grade is an application that is part of the Retek Predictive Application Server (RPAS). Internally, Retek Allocation also updates its store grade groups data groups based on the most current definitions. This update plays an important role when many months pass between initial and final allocations.

From the planning application to Retek Allocation

- Plan data
  Retek Allocation accesses plan data that originates in the planning application (including Retek's planning applications that reside on the RPAS server). The RPAS products are applications that provide functionality for developing, reconciling, and approving plans. When interfacing with Retek planning applications, Retek Allocation accesses department, class, subclass, style-color, or SKU plan data at the store-week level. When interfacing with legacy planning information, Retek Allocation accesses SKU, style-color, subclass, class, or department level plan data at the location-week level. Retek Allocation can be used as a tool to verify the final product-store plans and to initiate a PO to execute the plan. In other words, Retek Allocation can take the retailer's plan or forecast and execute it. Both the Retek and the legacy planning applications populate a planning table, ALC_PLAN, which resides within Retek Allocation. See the section, 'Planning table in Retek Allocation', later in this chapter.

From RDF/Curve to Retek Allocation

- Curve data
  Curve data becomes size profile data once it's integrated into Retek Allocation. If allocations are made at the style level, Retek Allocation utilizes the Curve data to get to the SKU level. For more information, see the section, 'Size profile logic' in "Chapter 5 – Allocation calculations" and see "Chapter 6 – RETL batch processing".

From RPM to Retek Allocation

- Future retail price data
  Retek Allocation uses this data to provide the user with the future retail price value of the entire allocation (based on its quantities). In addition, users can access future retail price values by location and by item. For information about configuration related to this setting, see the section, 'Display future unit retail price values' in "Chapter 2 – Backend system administration and assumptions".

From the warehouse management system (such as RWMS) to Retek Allocation via the merchandising system

- Appointment data
  Appointment data is one source that identifies item(s) to be allocated.
- Warehouse inventory position data
- ASN information

From the merchandising system to Retek Allocation

📖 **Note for RMS users only:**
Item, purchase order, supplier, sales and other data are accessed directly from the RMS tables, with no need to interface data via batch modules.

Via a Java database connectivity (JDBC) link, Retek Allocation receives the following data:

- Item data
  Retek Allocation can allocate at the SKU, style-color, pack, or item list level. Styles, SKUs, and packs can be mixed on a single allocation.
- PO data
- Hierarchy data
- Sales history data (for items, user-defined attributes [UDA], warehouses, stores, and so on)
- Foundation data (supplier data, shipping tables, and so on)

From Retek Allocation to the merchandising system

Retek Allocation calculates the allocation based on the information it has received from the merchandising system and/or the planning system. Once the retailer reviews and approves the allocation, Retek Allocation sends the following information back to the merchandising system:

- Approved allocation data
- Worksheet status POs that contain product, supplier and quantity information (the only remaining actions to be taken in the merchandising system are to approve the PO and, if desired, to truck scale the PO.)

From the merchandising system to the warehouse management system (such as RWMS)

📖 **Note for RMS users only:**
Retek Allocation utilizes the existing integration between RMS and RWMS. This interface currently passes purchase order, item, location, and allocation information from RMS to RWMS.

Based upon the approved allocation information from Retek Allocation, the merchandising system sends the following information to the distribution management system:

- Approved allocation data at the warehouse-PO-SKU-store quantity level. This data represents the store quantity instructions for allocating a specific quantity of stock at the store level.

From Active Retail Intelligence to Retek Allocation

Active Retail Intelligence (ARI) is an exception management and resolution system driven by custom business rules. Depending upon ARI's configuration, an ARI user could receive an alert that includes a link to Retek Allocation in the form of a URL address. The user could then log on to Retek Allocation in order to address the contents of the ARI alert.

How need is determined

To accurately determine individual store gross need, retailers want the flexibility to choose forecast data, plan data, sales history data, or combinations of this data.

Through the front end, retailers select a rule based on a portion of this data that accurately gathers gross need. The source of the data used by each rule is illuminated in this section.

To determine the net need at the store level, the system takes the gross need and subtracts from it the stock-on-hand at the store level. The equation that Retek Allocation uses to determine the stock-on-hand at the store is described later in this chapter.

The sources of data used by rules to determine gross need

Note: For a description of how the following rules use the data to determine gross need, see the Retek Allocation User Guide.

History data sources

For this rule, data is gathered primarily from the following tables:

RMS 11.0	Legacy system
DEPT_SALES_HIST	This table contains one row for each dept-location-week-sales type combination. Sales history, forecast and plan information about each combination is held.
CLASS_SALES_HIST	This table contains one row for each class-location-week-sales type combination. Sales history, forecast and plan information about each combination is held.
SUBCLASS_SALES_HIST	This table contains one row for each subclass-location-week-sales type combination. Sales history, forecast and plan information about each combination is held.
ITEM_LOC_HIST	This table contains one row for each item-location-week-sales type combination. Sales history, forecast and plan information about each combination may be held here.

Forecast data sources

For this rule, data is gathered primarily from the following tables:

RMS 11.0	Legacy system
DEPT_SALES_FORECAST	This table holds the forecast information summed to the department-location-eow_date.
CLASS_SALES_FORECAST	This table holds the forecast information summed to the class-location-eow_date and should be partitioned by domain_id, as well. Thus, if only a portion of the domains is forecasted, then the rebuild is done by domain_id.
SUBCLASS_SALES_FORECAST	This table holds the forecast information summed to the subclass-location-eow_date and should be partitioned by domain, as well. Thus, if only a portion of the domains is forecasted, then the rebuild is done by domain_id.
ITEM_FORECAST	This table holds the item level forecasted information from the RDF extractions. This table holds all item types. This table should be partitioned according to the domain level.

Plan data sources

For this rule, data is gathered primarily from the following table:

- ALC_PLAN

For a more detailed description of this table, see the section, ‘Planning table in Retek Allocation’, later in this chapter.

History and Plan data sources

For this rule, some data is gathered from the following plan table in Retek Allocation:

- ALC_PLAN

and some data is gathered from the following tables:

RMS 11.0	Legacy system
DEPT_SALES_HIST	This table contains one row for each dept-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
CLASS_SALES_HIST	This table contains one row for each class-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
SUBCLASS_SALES_HIST	This table contains one row for each subclass-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
ITEM_LOC_HIST	This table contains one row for each item-location-week-sales type combination. Sales history, forecast, and plan information about each combination may be held here.

Plan re-project data sources

Note: For a description of the Bayesian algorithm that is used in this section, see "Chapter 5 – Allocation calculations".

For this rule, some data is gathered from the following plan table in Retek Allocation:

- ALC_PLAN

and some data is gathered from the following tables:

RMS 11.0	Legacy system
DEPT_SALES_HIST	This table contains one row for each dept-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
CLASS_SALES_HIST	This table contains one row for each class-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
SUBCLASS_SALES_HIST	This table contains one row for each subclass-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.

RMS 11.0	Legacy system
ITEM_LOC_HIST	This table contains one row for each item-location-week-sales type combination. Sales history, forecast, and plan information about each combination may be held here.

Corporate rules

For this rule, data is gathered from the selected column of the following tables in Retek Allocation:

- ALC_CORPORATE_RULE_HEAD
- ALC_CORPORATE_RULE_DETAIL

The column selection is based on which corporate rule is picked by the user.

Note: If the retailer plans ideal weeks of supply (IWOS) by product-location, the corporate table can be accessed to create different ideal weeks of supply by store. If the retailer does *not* plan IWOS, a field can be created that contains the same IWOS for every store.

Quantity limits

Quantity limits allow the user to set parameters which affect different stages of the allocation for the product-stores where they are entered. The values for each applicable quantity limits selection are held on the applicable column of the ALC_QUANTITY_LIMITS table.

Stop ship

A ‘stop ship’ is a product/location combination that prevents an item from shipping to that location. The system looks at the release dates entered on the product window and compares them with stop ship records entered via the merchandising system. If the release date is on or between the stop ship dates, the system inserts ‘0’ into the min and max columns of quantity limits for this store-item.

The release date is on the ALC_ITEM_LOC table and is represented by the column, release_date. The STOP_SHIP table contains the stop ship date range: start_date and end_date. In order for a stop ship record to stop shipment of an allocation, the store, department, class, and subclass of the allocated item must match the store, department, class, and subclass on the stop_ship record, or the store and style (for fashion) or SKU (for staple) of the item being allocated must match the store and item_id on the STOP_SHIP table. See the Retek Allocation User Guide for more information.

Net need at the store level calculation

On a fundamental level, net need is gross need minus the on-hand at the store.

Note: Although quantity limits also effect net need, they are not addressed in the calculation illustrated by this passage.

To determine the gross need, Retek Allocation gathers the information based upon one of the rules selected by the retailer through the front end. Retek Allocation uses the following equation to determine the on-hand at the store that is subtracted from the gross need result.

Stock-on-hand at the store+
Stock in transit+
Stock on order [stock that is expected by the on order commit date]+
Transfers of stock expected+
Stock on allocation

–

Outgoing transfers +
Return to vendor stock+
Unavailable stock+
Transfers on reserve

An abbreviated version of this equation would be the following:

(SOH +InTransit+OnOrder+TSF Expected+OnAlloc) –

(TSFOut+RTV+Unavailable+TSF Reserved)

Tables populated by external systems

The following tables are owned by Retek Allocation. The data within them is populated by external systems. For descriptions of each table and its columns, see the Retek Allocation Data Model.

- ALC_CORPORATE_RULE_DETAIL
- ALC_CORPORATE_RULE_HEAD
- ALC_IDEAL_WEEKS_OF_SUPPLY
- ALC_PLAN
- ALC_SIZE_PROFILE
    - Can also be populated through size profile setup via the front-end of the application.
    - Can also be populated through RETL processing from Retek Demand Forecasting (RDF)
- ALC_USERS
- ALC_USER_DEPTS

Planning table in Retek Allocation

Both the Retek and the legacy planning applications populate a planning table, ALC_PLAN, which resides within Retek Allocation. This table includes the following columns:

- Plan ID
- Store
- EOW.date
- Department
- Class
- Subclass
- Item
- Diff1
- Quantity

A record can thus exist at any of the following levels by week-store-quantity:

- Department
- Department-class
- Department-class-subclass
- Item-color
- SKU

Merchandising interface tables

This section lists by functional area, the tables that Retek Allocation uses from within the following merchandising systems:

- RMS 11.0
- Legacy systems

RMS 11.0 tables (for retailers with RMS only)

The following table illustrates the tables from which Retek Allocation 11.0 gets its data from RMS 11.0.

RMS 11.0 tables	
Functional area	**Associated tables**
ITEM DATA	SUB_ITEMS_HEAD
	SUB_ITEMS_DETAIL
	ITEM_SUPP_COUNTRY
	ITEM_SUPPLIER
	ITEM_LOC
	ITEM_LOC_HIST
	ITEM_LOC_SOH
	ITEM_PARENT_LOC_HIST
SKULIST DATA	SKULIST_HEAD
	SKULIST_DETAIL
PACK DATA	PACKITEM
	ITEM_MASTER
	ITEM_LOC
ORDER DATA	ORDHEAD
	ORDLOC_WKSHT
	ORDLOC
	ORDSKU
	ALLOC_HEADER
	ALLOC_DETAIL
	SHIPMENT
SUPPLIER DATA	SUPS
	ITEM_SUPPLIER
LOCATION LIST DATA	LOC_LIST_HEAD
	LOC_LIST_DETAIL
	LOC_LIST_CRITERIA
MERCHANDISE HIERARCHY DATA	DEPS
	CLASS

RMS 11.0 tables	
Functional area	**Associated tables**
	SUBCLASS
ORGANIZATIONAL HIERARCHY DATA	STORE
	WH
	WH_STORE_ASSIGN
SHIPMENT DATA	SHIPMENT
	SHIPSKU
STORE GRADE DATA	STORE_GRADE_GROUP
	STORE_GRADE
	STORE
	BUYER
	STORE_GRADE_STORE
SECURITY DATA	SEC_USER_LOC_MATRIX
	SEC_USER_PROD_MATRIX
LOCATION TRAITS DATA	LOC_TRAITS
	LOC_TRAITS_MATRIX
	LOC_AREA_TRAITS
	LOC_REGION_TRAITS
	LOC_DISTRICT_TRAITS
TRANSFER DATA	TSFHEAD
	TSFDETAIL
UDA DATA	UDA
	UDA_VALUES
	UDA_ITEM_LOV
FORECAST DATA	DEPT_SALES_FORECAST
	CLASS_SALES_FORECAST
	SUBCLASS_SALES_FORECAST
	ITEM_FORECAST
SALES DATA	DEPT_SALES_HIST
	CLASS_SALES_HIST
	SUBCLASS_SALES_HIST

RMS 11.0 tables	
Functional area	**Associated tables**
	ITEM_LOC_HIST
	ITEM_PARENT_LOC_HIST
APPOINTMENT DATA	APPT_HEAD
	APPT_DETAIL

Suggested legacy system tables

The following table illustrates the suggested tables from which Retek Allocation gets its data from a legacy system.

Suggested legacy tables	
Functional area	**Associated tables**
ITEM DATA	A table that holds substitute item header information by location.
	A table that holds substitute item detail information by location.
	A table that holds one record for each origin country associated with a given item-supplier.
	A table that holds all item supplier relationships for all items.
	A table that contains one row for each item stocked at each location within the company.
	A table that contains one row for each item-location-week-sales type combination. Sales history, forecast, and plan information about each combination may be held here.
	A table that contains one row of stock-on-hand information for each item stocked at a location within the company.
	A table that holds the rolled up sales history for item parents.
SKULIST DATA	A table that contains the header information for each item list created within the system. An item list can contain SKUs and styles.
	A table that contains one row for each item or item parent within an item list.

Suggested legacy tables	
Functional area	**Associated tables**
PACK DATA	A table that contains one row for each pack item-item combination that has been created. Base information about each item in each pack is held. Because a pack may contain other packs, some of the component items on this table may also be packs themselves.
	A table that holds all the main attributes and records for all items and pack items in the merchandising system. Additionally, this table has referential integrity on itself which hold the links between grandparent, parent and child items.
	A table that contains one row for each item stocked at each location within the company.
ORDER DATA	A table that contains one row for each order that has been placed by the company.
	A table that contains worksheet records for an order. A user is able to place items on the worksheet table and leave them there until the item is fully distributed.
	A table that contains one row for each order-SKU-store or warehouse combination that has been placed by the company.
	A table that contains one row for each order-item combination that has been placed by the company. Base information about each item on each order is held.
	A table that contains header level information for the allocation of a SKU from a warehouse to a group of stores or other warehouses.
	A table that contains one row for every allocation store-warehouse combination. Allocations can be attached to a purchase order or can be created as stand-alone allocations.
	A table that contains one row for each shipment within the system. Base information about each shipment for each order is held in this table for as long as its associated order header is retained.

Suggested legacy tables	
Functional area	**Associated tables**
SUPPLIER DATA	A table that contains one row for each supplier within the company. Whenever a supplier name and so on is used by Retek, or a supplier number is validated, it is always selected from this table.
	A table that holds all item supplier relationships for all items.
LOCATION LIST DATA	A table that contains the header level information for each location list set up in the system. A location list can contain store and warehouse. The information includes the stored grouping criteria for store and for warehouse. These criteria are used to rebuild all lists.
	A table that contains one row for each location (store or warehouse) within a location list.
	A table that contains one row for each step performed to obtain a store grouping criteria and a warehouse grouping criteria. For each step performed, a query where clause is formed to include/exclude location in/from the location list. These grouping criteria can be used to rebuild the location list.
MERCHANDISE HIERARCHY DATA	A table that contains one row for each department within the company. Whenever Retek uses a department name, and so on, or a department number is validated, the data is selected from this table.
	A table that contains one row for each class within the company. Whenever Retek uses a class name, or a class is validated, it is always selected from this table.
	A table that contains one row for each department-subclass combination within the company. Whenever Retek uses a subclass name, or a subclass is validated, it is always selected from this table.
ORGANIZATIONAL HIERARCHY DATA	A table that contains one row for each store within the company.

Suggested legacy tables	
Functional area	**Associated tables**
	A table that contains one row for each warehouse within the company. Whenever Retek uses a warehouse name or address and so on, or a warehouse number is validated, it is always selected from this table.
	A table that contains warehouse store assignment information. Each record determines on what date a store is supposed to be assigned to a warehouse.
SHIPMENT DATA	A table that contains one row for each shipment within the system. Base information about each shipment for each order is held in this table for as long as its associated order header is retained.
	A table that contains one row for each shipment-SKU combination in the system. When a shipment header is purged all associated rows in this table are also purged.
STORE GRADE DATA	A table that contains store grade group information. It is the header table for store grades. A store grade group is a mechanism to group stores together. A store grade group consists of multiple store grades, each containing many stores.
	A table that contains store grade information. Each store grade within a group contains one or more stores.
	A table that contains one row for each store within the company.
	A table that contains one row for each person authorized to create purchase orders.
	A table that contains a record for each store grade group.
SECURITY DATA	A table that is used to store user location security attributes.
	A table that is used to store user product security attributes.

Suggested legacy tables	
Functional area	**Associated tables**
LOCATION TRAITS DATA	A table that contains one row for each location trait in the system. Location traits allow stores to be grouped based on common characteristics.
	A table that contains store-location trait relationships.
	A table that contains one row for each area level location trait defined within Retek.
	A table that contains one row for each region level location trait defined within Retek.
	A table that contains one row for each district level location trait defined within Retek.
TRANSFER DATA	A table that contains one row for each transfer that has been created in the system.
	A table that contains one row for each transfer-SKU-prepack-inv_status combination held in the system.
UDA DATA	A table that contains one row for each user-defined attribute (UDA), defined within the merchandising system. Generally, a UDA is any attribute that does not have specific processing in the merchandising system.
	This table contains all valid values associated with a UDA.
	This table contains one row for each item-attribute combination for UDAs with display_type of list of values (LV).
FORECAST DATA	This table holds the forecast information summed to the department-location-eow_date.
	This table holds the forecast information summed to the class-location-eow_date and should be partitioned by domain_id. Thus, if only a portion of the domains is forecasted, then the rebuild is done by domain_id.

Suggested legacy tables	
Functional area	**Associated tables**
	This table holds the forecast information summed to the subclass-location-eow_date and should be partitioned by domain. Thus, if only a portion of the domains is forecasted, then the rebuild is done by domain_id.
	This table holds the item level forecasted information from the demand forecasting application's extractions. This table holds all item types.
SALES DATA	This table contains one row for each dept-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
	This table contains one row for each class-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
	This table contains one row for each subclass-location-week-sales type combination. Sales history, forecast, and plan information about each combination is held.
	This table contains one row for each item-location-week-sales type combination. Sales history, forecast, and plan information about each combination may be held here.
	This table holds the rolled up sales history for item parents.
APPOINTMENT DATA	This table holds header-level information for warehouse management system-generated appointments. The table contains one record per appointment/location combination.
	This table holds detail-level information for warehouse management system-generated appointments. The table contains one record per appointment/location/item/ASN combination.

Chapter 5 – Allocation calculations

This chapter provides an overview of allocation calculations, including the allocation queue processes. Because allocation involves the distribution of a set number of items across a number of different locations, an allocation's measure of success is how well it solves the overall need for an item across various locations.

Retek Allocation's calculation engine is designed to try and fill each store's predefined need as closely as possible given a constrained quantity of product.

Assumptions related to calculations

The system is programmed to assume that any given allocation follows these guidelines.

Store order multiple assumption

The store order multiple (SOM) is the default unit of measure by which an item is shipped from the warehouse to the store (for example, cartons, inner packs, eaches, and so on). Because the system allocates in groups equal to the SOM, the system may not be able to exactly apply the size profile.

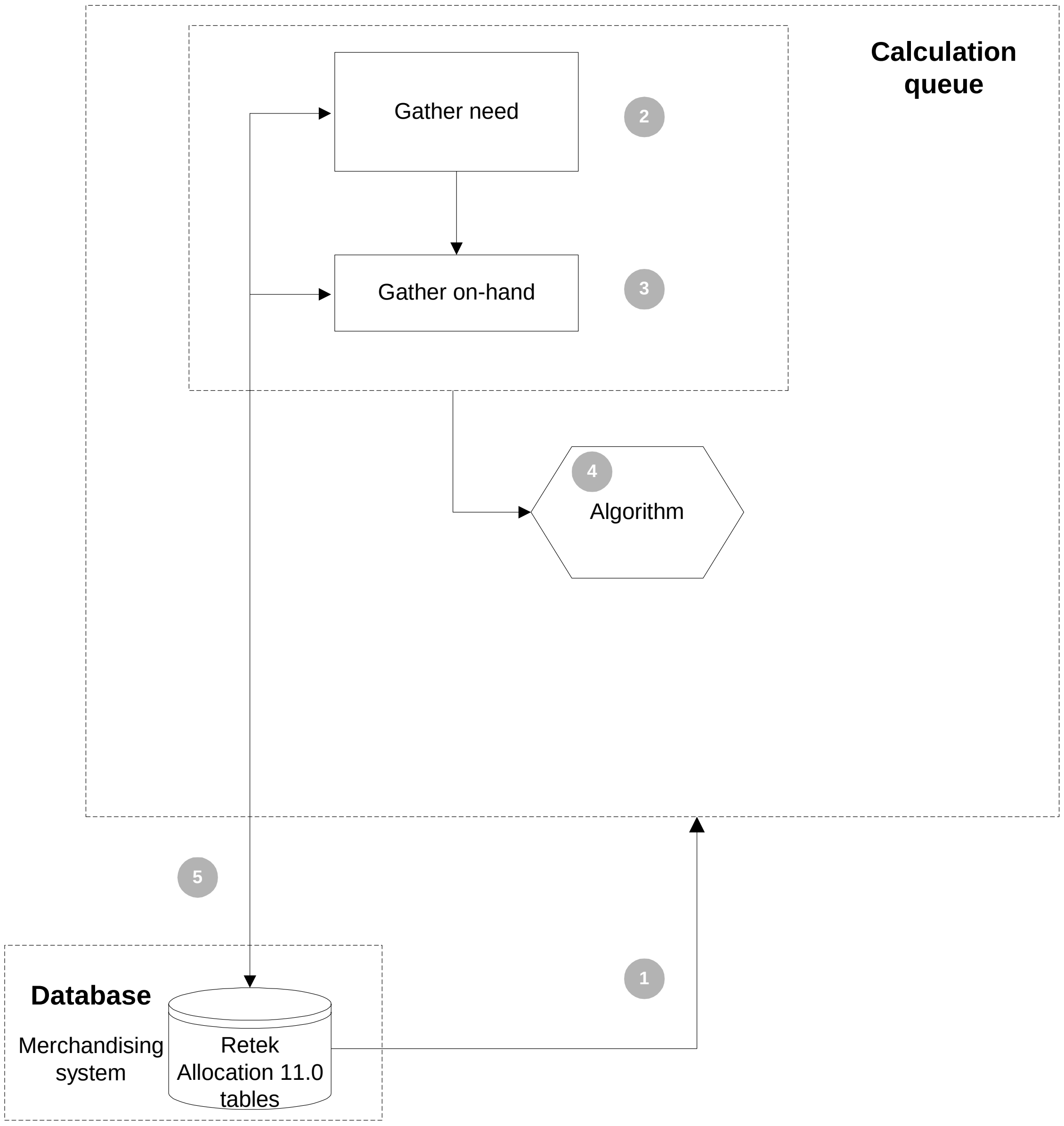
The system assumes that all of the SKUs under a style have the same SOM as the style. In other words, the SOM cannot differ by SKU under a style.

Proportional allocation assumption

The system assumes that a proportional allocation will *not* contain more than 10,000 units going to one store. The 10,000-unit value is a hard limit, and if exceeded, an infeasible solution arises in the system's algorithm.

Calculation queue processing

The following diagram offers an overview of the calculation queue process. Explanations of the numbered steps follow the diagram. Note that the numbers do not necessarily reflect the system's order of operation but are provided to facilitate the discussion of the process.



Retek Allocation calculation queue process

Calculation queue process description

1 A continuously running thread pulls an allocation to be calculated into the calculation queue.

2 Need is gathered based upon the applicable rule and the data from the database. For example, the need gathered could be based upon plan data, sales history data, forecast data, plan re-project (Bayesian-calculated) data, and so on. These calculations are internal to the Java in Retek Allocation. For more information about the data that the system uses to generate gross need, see “Chapter 4 – Functional design”.

3 On-hand is gathered, as necessary, from the database depending upon whether gross need or net need is desired. These calculations are internal to the Java in Retek Allocation. For more information about the source of on-hand data and how on-hand is calculated, see “Chapter 4 – Functional design”.

4 The algorithm, an external library written in C++, is called to make a statistical determination of the best allocation possible given the parameters and constraints of the problem. Inputs passed to the algorithm function include the following:

   - Available quantity by SKU
   - An exact/proportional flag
   - SKU-store matrices of need, on-hand, minimum, maximum, and threshold

   Cascade mode uses additional vectors for cascade-level targets, minimum, maximum, and threshold. Pack mode uses a matrix representing the pack component quantities.

   Depending upon the mode (for example, simple, cascade, or pack) and the constraints of a given problem, an algorithm is selected to calculate the allocation, including an ‘objective’ function representing a relative score for the evaluation of different options. In every mode, the return values represent the SKU (pack)-store amount to be allocated and shipped. In all modes (for example, simple mode, cascade mode, and so on), optimization is performed through a heuristic algorithm.

5 The results are retrieved and saved to the database.

Calculation queue scripts

Retek Allocation includes sample scripts for calculation and optimum prepack calculation queue management. These scripts are examples provided for the retailer’s convenience.

There is no maximum number of calculation and optimum prepack calculation queues that can be run at any given time. Processor and memory limits (specific to a retailer’s implementation) on a given Windows or UNIX box determine a maximum number.

Start (queue_11.sh)

This script is used for calculation queue management. The script calls set10.sh to set variables that are needed to run the calculation queue. The command syntax is as follows:

```
queue_11.sh start x
```

The command starts an instance of the calculation queue. ‘x’ represents an integer which serves as the ID for the queue being started. This integer should be unique among all instances of the calculation queue being run.

When the queue is started, the script attempts to create a log for the calculation queue in a subdirectory of the current directory. Note that this subdirectory is referred to as ‘logs’ in the script template. Retailers can specify any location they wish for these log files. The referenced directory should be created prior to running the script. Existing log files are overwritten when a queue is started.

Stop (queue_11.sh)

The command syntax that stops an instance of the calculation queue specified by the integer ‘x’ is as follows:

```
queue_11.sh stop x
```

Status (queue_11.sh)

The command syntax that provides status information about an instance of the calculation queue specified by the integer ‘x’. is as follows:

```
queue_11.sh status x
```

Restart (queue_11.sh)

The command syntax that stops and restarts an instance of the calculation queue specified by the integer ‘x’ is shown below. The newly started instance of the queue contains a new PID and creates a new log file that replaces the existing log file.

```
queue_11.sh restart x
```

The operative command used to start a calculation queue is:

```
java -classic -cp $CLASSPATH com.retek.alloc.calculation.CalcQueue x
```

where ‘x’ represents the unique integer specifying the ID of the calculation queue.

Note: Some JDK implementations on UNIX boxes do *not* have the classic option. The ‘-classic’ argument can be removed from the script/command or substituted with ‘-server’ (if the server option exists on your JDK implementation).

Optimal prepack queue management (queue_pre11.sh)

This script is used for optimal prepack calculation queue management. The script calls set11.sh to set variables that are needed to run the optimal prepack calculation queue. The command syntax is the same as for queue_11.sh, except that ‘queue_11.sh’ is replaced by ‘queue_pre11.sh’.

The operative command used to start a calculation queue is:

```
java -classic -cp $CLASSPATH com.retek.alloc.calculation.CalcQueue x
-prepack
```

where ‘x’ represents the unique integer specifying the ID of the calculation queue.

Note: Some JDK implementations on UNIX boxes do *not* have the classic option. The ‘-classic’ argument can be removed from the script/command or substituted with ‘-server’ (if the server option exists on your JDK implementation).

Set environment variable (set11.sh)

This script template is called by queue_11.sh and queue_pre11.sh to set an environment variable needed for the calculation and prepack calculation queues. It sets a $J2EE_HOME, a $JAVA_HOME and library path settings specific to the box vendor (Sun, AIX, HP). Note the comments in this script that specify which settings should be used in your implementation.

Plan re-project algorithm

A Bayesian algorithm is one that is based on a mathematical theorem developed in the eighteenth century by the Reverend Thomas Bayes. This theorem is a basic starting point for inference problems that use probability theory as logic.

The forecasting algorithm is provided here only to give the technical reader insight into exactly how actual sales are combined with a plan to produce a forecast.

Let:

N be the number of periods in the current season (and $N = \infty$ for staple products),

M be the current period,

$p(j)$ be the sales plan for periods $j = 1, \ldots, N$,

$x(j)$ be the achieved sales for period $j = 1, \ldots, M$, and

α be a constant between 0.0 and 1.0 (This parameter influences the balance between model sensitivity and robustness, that is, how responsive the model is to new sales data).

Additionally, define

$P' \equiv \sum_{j=1}^{M} p(j)$ as the sum of the sales plan up to the current period,

$P'' \equiv \sum_{j=1}^{N} p(j)$ as the sum of the sales plan over the entire season, and

$X' \equiv \sum_{j=1}^{M} x(j)$ as the sum of the achieved sales up to the current period.

Finally, compute the forecasts as

$$\hat{x}(j) \equiv p(j)\left(\frac{X'}{P'}\right)\left(\frac{P'}{P''}\right)^{\alpha} + p(j)\left(1-\left(\frac{P'}{P''}\right)^{\alpha}\right) \text{ for periods } j = M+1, \ldots, N.$$

The motivation for this forecast is relatively clear. The forecast is a convex combination of a scaled version of the sales plan, $p(j)\left(\frac{X'}{P'}\right)$, and the original sales plan itself. The scaled version of the sales plan is scaled based upon the ratio of the achieved historical sales to the historical sales plan (for example, if the retailer sold twice what it had planned to sell in the past, then the scaled plan would be twice the original plan). Thus, if the retailer had no confidence in the magnitude of the original sales plan (but still believed in its time profile or shape), then the scaled plan would probably be a good forecast on its own. On the other hand, if the retailer really still believes in the plan and the retailer does not believe that the recent past performance is indicative of future performance, then it would make sense to stick with the original sales plan as the forecast.

In the forecasting algorithm, the weights assigned to the scaled and original plans represent the confidence in the respective portions. As $\left(\frac{P'}{P''}\right)$ becomes larger (that is, the portion of the plan that can be compared to historical sales increases), the retailer tends to have more confidence in the scaled plan. For example, if $\left(\frac{P'}{P''}\right)$ = 0.01, then the retailer really does not have much information upon which to base the scaled plan. On the other hand, for example, as the quantity approaches 0.5 (that is, the season is half way over), then the retailer really should start seriously considering why the plan was incorrect. The retailer may have greater belief in the scaled plan. Additionally, the α parameter is used to tweak the sensitivity of the forecasting method. As α increases, the forecast will tend to stay closer to the original plan. For small values of α, the forecast will move rapidly towards the scaled plan as historical sales data becomes available. Retailers should use their own data and judgment to determine an appropriate α for their particular business problem at hand. For more information, see the section ‘Bayesian sensitivity factor’ in “Chapter 2 – Backend system administration and assumptions”.

Guidelines

Bayesian forecasting is primarily designed for use with new product-location positions. The following guidelines should be followed:

1. No more than one plan should exist for a given product-location position.
2. Any time period with non-zero actuals for a given product-location position should have a corresponding plan component (otherwise the system will assume a plan exists and equals zero and will act accordingly).
3. Any non-zero actuals not within the time period of interest should be overridden to zero.

Satisfying need across multiple locations

One way to think of the application's methodology is by comparing its process to that of water filling a container.

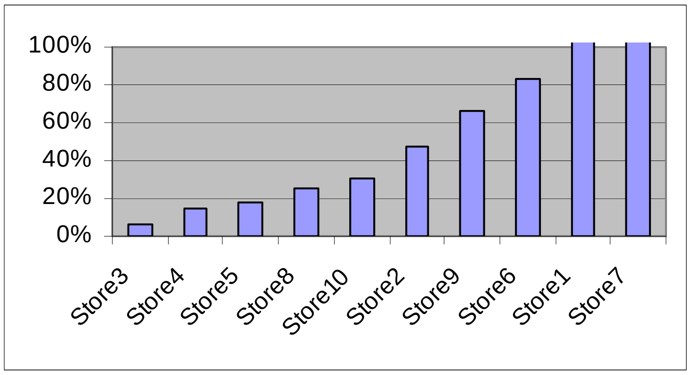
The fill line of the container represents 100% of need for all the locations, and the quantity of water available represents the stock available. By pouring all of the water into the container, the water level naturally reaches the target stock.

Suppose that the bottom of the container is partitioned, and each partition represents a different store.

The water would naturally level itself to maintain consistent stock across all stores.

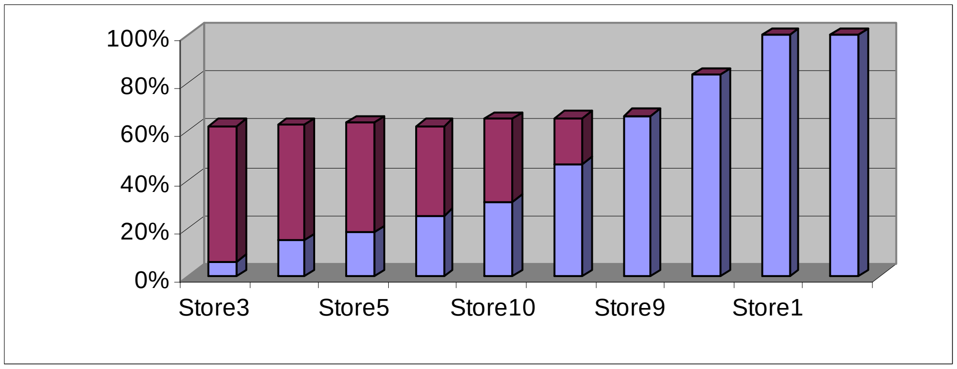
Suppose too the following three conditions, as depicted in the accompanying chart:

- Stock is already available at each store.
- The height of each partition in the container represents the amount of that location's existing stock.
- The partitions are sorted from left to right in ascending stock percentages.



Stores' existing stock and need before allocation

When this staircase container fills with water, all locations to the left of the waterline would receive an allocation quantity that satisfies need. All locations to the right of the waterline would not receive an allocation because their existing stock satisfies need. The final allocation does not guarantee that all stores end with the same percentage of need, but instead guarantees that all locations which receive an allocation has the same percentage of need and that all others are at or above this percentage of need.



Stores' stock after allocation

Rounding conditions

Some allocation algorithms use rounding rules to determine when to round fractional components up or down to whole number answers. However, simple rounding often causes problems if not handled appropriately, including the allocation of too many or too few items. By using sophisticated optimization techniques, Retek's allocation algorithm deals in whole numbers directly and avoids rounding.

Size profile logic

The system uses historical sales data to create a size profile (a graph) that illustrates how many units should be allocated by size. The data that answers the question is useful because the size of people in different regions can differ. For example, a question that could arise during an allocation might be: how many size 8 red shirts did I sell in Chicago as opposed to Los Angeles? Retek Allocation calculates that the Los Angeles area needs x percentage more of smaller sized shirts than Chicago.

The allocation of packs

Retek optimally allocates packs of multiple items based on stores' need for the individual component items in the packs. The benefit of this approach is that the system does not follow arbitrary rules for allocating packs and items. Instead, a consistent holistic approach is used. The potential answers are evaluated by how well the resulting allocation of component items satisfies the stores' need for the individual items. See front-end documentation for a definition of the two types of packs, sellable and non-sellable.

Note the following conditions that apply to sellable packs:

- A sellable pack is connected to plan data and to forecast data via the pack_no rather than the component SKUs.
- The ITEM_LOC_HIST table is used to determine the SKU-level history for a sellable pack.
- The on-hand value for a sellable pack is for the pack rather than for the components.

Note the following condition that applies to non-sellable packs:

- When the user selects a style-color to allocate, the system retrieves all non-sellable packs that contain only that style-color.
- Retrieved non-sellable packs are not displayed for the user, but the contents of the pack are included in the available quantity.
- The on-hand value for a non-sellable pack is at the component level. The on-hand value can be generated for the style-color.
- The need value for a non-sellable pack is determined at the component level. The need value can be generated for the style-color.
- Allocation transactions respect both open stock (SKUs) and the associated pack level.

By considering the entire matrix of available packs against component-store need for optimization, Retek Allocation provides a sophisticated solution for the distribution of multi-product packs.

The following example illustrates a relatively simple fashion prepack allocation:

The user selects a style to allocate. The system determines the availability of the style, in the warehouse or from a PO, and understands the quantity available to allocate and the prepack matrix (the component makeup of the pack). The user sees the total number of items available for the style, 420 units in the example below.

Prepack Matrix				Packs Available	Items Available
	S	M	L		
Pack 1	3	3	6	20	240
Pack 2	3	6	3	15	180

The user picks a rule that defines the need for the styles being allocated. Note that this need is for the style, not for the component items or for the packs.

Need for style	
Store 1	150
Store 2	150
Store 3	100

The allocation system looks up the appropriate size curve for each style-store, based on the style or subclass. Size curves are not calculated during the allocation process. Retek Curve or another similar system is used to calculate size profiles.

Size curve			
	S	M	L
Store 1	20%	50%	30%
Store 2	20%	30%	50%
Store 3	20%	30%	50%

Need for each individual item is calculated by multiplying style need by the size curve.

Item need				
	S	M	L	Total
Store 1	30	75	45	150
Store 2	30	45	75	150
Store 3	20	30	50	100

Finally the allocation algorithm determines the optimal allocation of packs. Inputs to the process are the Prepack Matrix, Packs Available, and Item Need above. Other inputs not shown in this example may be included (for example, minimum, maximum, threshold, store on-hands, and so on).

Pack allocation		
	Pack 1	Pack 2
Store 1	1	12
Store 2	11	2
Store 3	8	1

The user sees the result in terms of number of allocated per style, as shown in the following table:

Style allocation		
	Need	Allocation
Store 1	150	156
Store 2	150	156
Store 3	100	108

The user can also view the results by individual item to make sure the size distribution is correct. If the distribution is poor, action may need to be taken such as the ordering of more products or the breaking packs before allocation.

Item allocation			
	S	**M**	**L**
Store 1	39	75	42
Store 2	39	45	72
Store 3	27	30	51

The objective of the algorithm is to minimize the variance between the actual allocation and need across all item-store combinations.

The algorithm with packs and singles

The algorithm determines the optimum result based on the overall allocation of every individual product. Both packs and singles are allocated to stores as needed. Packs, however, are preferred to singles in order to minimize the total number of units distributed, thereby lowering handling costs.

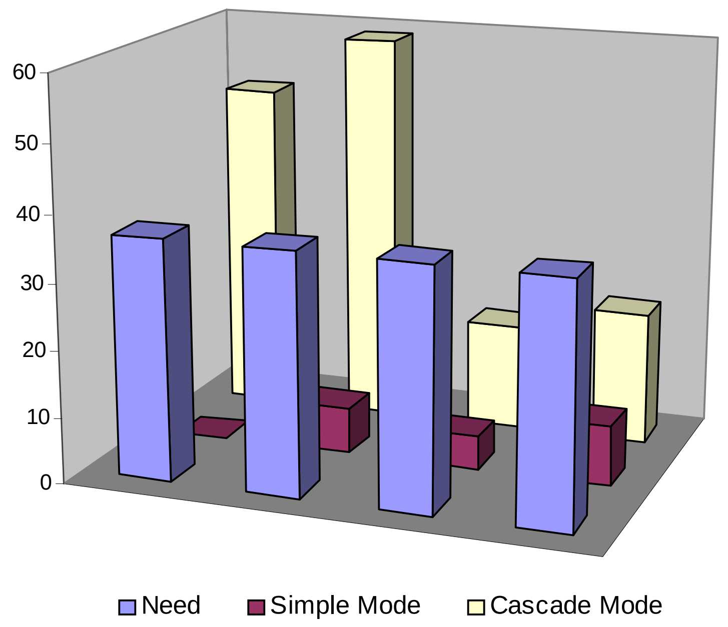
Cascade allocations

In many circumstances, merchandise issues that come up at the SKU level can be accommodated by adjusting allocations at higher levels in the merchandise hierarchy. The Retek Allocation’s allocation optimization algorithm is applied to cascade allocations.

The following example illustrates a cascade allocation:

The retailer is running an ad for t-shirts, with a picture of the yellow one. All t-shirts are displayed on a store table all together. The yellow t-shirt has been allocated, and the next step is to make sure that every store has an adequate stock of all t-shirts to support other customer choices.

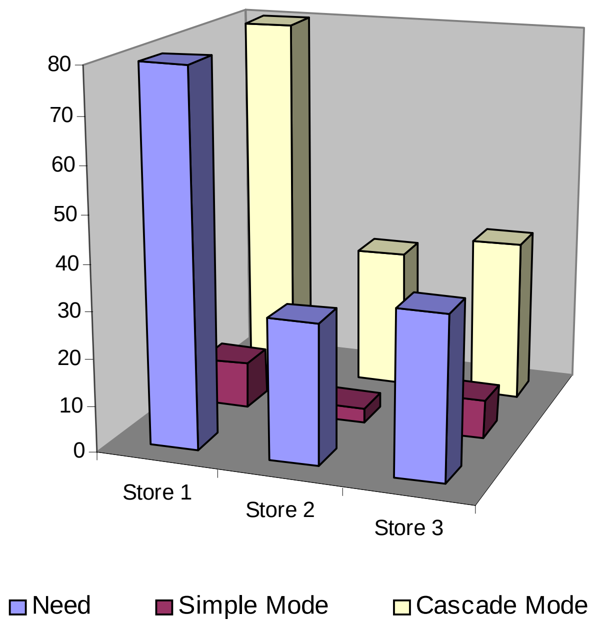
On a per item basis, the simple mode allocation distributes as much quantity as is available, up to the calculated need. However, the cascade mode allocation exceeds the item need if it is necessary to meet the category need. The chart, “Total item need vs. allocation”, illustrates that more of all the t-shirts are distributed by cascade mode, and that two of the t-shirts are allocated above and beyond their need in order to satisfy the total category demand.



Total item need vs. allocation

Note: The overstocking of two items, which takes place in this example, is due to the fact that constraints (for example, minimum, maximum, threshold, and so on) are not being used to limit the results. The example is designed to highlight the effects of the two different modes of calculating.

The chart, “Total category-store need vs. allocation”, illustrates that the category need is exactly met by cascade mode. This simplistic example illustrates the system’s preference for stocking the store for category performance rather than for mere item performance.



Total category-store need vs. allocation

To obtain these results, the user selects the t-shirts to be allocated, using a plan rule, at subclass (or any appropriate) level, in cascade mode. Remember that selecting cascade means instructing the allocation to favor the need to fill the t-shirt table, over the need or lack of available quantity for any specific shirt. Constraints such as minimum and maximum can be used to ensure the resulting assortment is reasonable.

Behind the scenes, the system determines setup information, including the number of items available to allocate, the number of items on hand at the stores, and the need by category-store.

Item available quantity to allocate							
Style	**Solid**	**Striped**	**Solid**	**Striped**	**Logo**	**Logo**	
Color	**Yellow**	**Yellow**	**White**	**Blue**	**White**	**Blue**	**Total**
Available	0	0	80	85	16	20	201

Item on hand at stores							
Style	**Solid**	**Striped**	**Solid**	**Striped**	**Logo**	**Logo**	
Color	**Yellow**	**Yellow**	**White**	**Blue**	**White**	**Blue**	**Total**
Store 1	50	50	20	20	15	15	170

Item on hand at stores							
Style	**Solid**	**Striped**	**Solid**	**Striped**	**Logo**	**Logo**	
Color	**Yellow**	**Yellow**	**White**	**Blue**	**White**	**Blue**	**Total**
Store 2	30	30	10	5	10	10	95
Store 3	30	30	10	5	10	5	90
Total	110	110	40	30	35	30	

Category need and on-hand at stores		
	Need	**On-hand**
Store 1	250	170
Store 2	125	95
Store 3	125	90

The next steps involve the examination of the need by category-store in order to determine the need by item-store. The following results occur when allocating using the Retek Allocation algorithm in simple and cascade mode. Cascade mode uses exactly the same information as simple mode, with the addition of the category targets. Constraints such as minimum, maximum, and threshold could be added to both calculations. Cascade mode can consider optional constraints at the category level.

Simple mode allocation – not cascade							
Style	**Solid**	**Striped**	**Solid**	**Striped**	**Logo**	**Logo**	
Color	**Yellow**	**Yellow**	**White**	**Blue**	**White**	**Blue**	**Total**
Store 1	0	0	0	0	5	5	10
Store 2	0	0	0	3	0	0	3
Store 3	0	0	0	4	0	4	8
Total	0	0	0	7	5	9	

Cascade-mode allocation							
Style	**Solid**	**Striped**	**Solid**	**Striped**	**Logo**	**Logo**	
Color	**Yellow**	**Yellow**	**White**	**Blue**	**White**	**Blue**	**Total**
Store 1	0	0	35	35	5	5	80

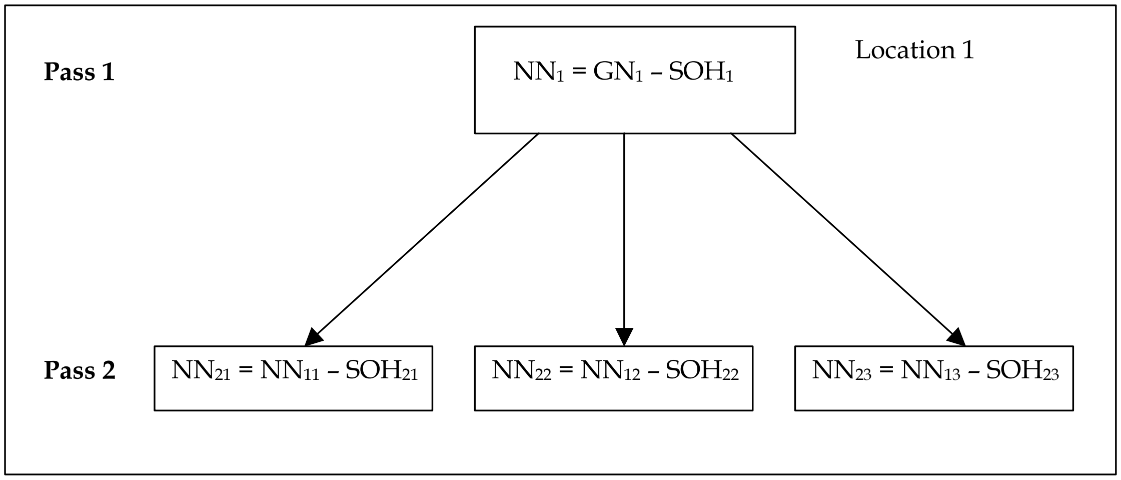
Cascade-mode allocation							
Style	**Solid**	**Striped**	**Solid**	**Striped**	**Logo**	**Logo**	
Color	**Yellow**	**Yellow**	**White**	**Blue**	**White**	**Blue**	**Total**
Store 2	0	0	7	12	6	5	30
Store 3	0	0	8	12	5	10	35
Total	0	0	50	59	16	20	

Staple cascade and fashion cascade

The following diagrams illustrate the system's approach to a staple cascade calculation and to a fashion cascade calculation.

Staple cascade

Explanations of Pass 1 and Pass 2 follow the diagram.


Pass 1
$NN_1 = GN_1 – SOH_1$
Location 1
Pass 2
$NN_{21} = NN_{11} – SOH_{21}$
$NN_{22} = NN_{12} – SOH_{22}$
$NN_{23} = NN_{13} – SOH_{23}$

Staple cascade

Pass 1
GN_1 = Gross need at a location
SOH_1 = Sum of SOH for all SKUs in ItemList-D-C-SC at that location
NN_1 = Net Net (after Pass 1)

$NN_1 = GN_1 = SOH_{1-}$

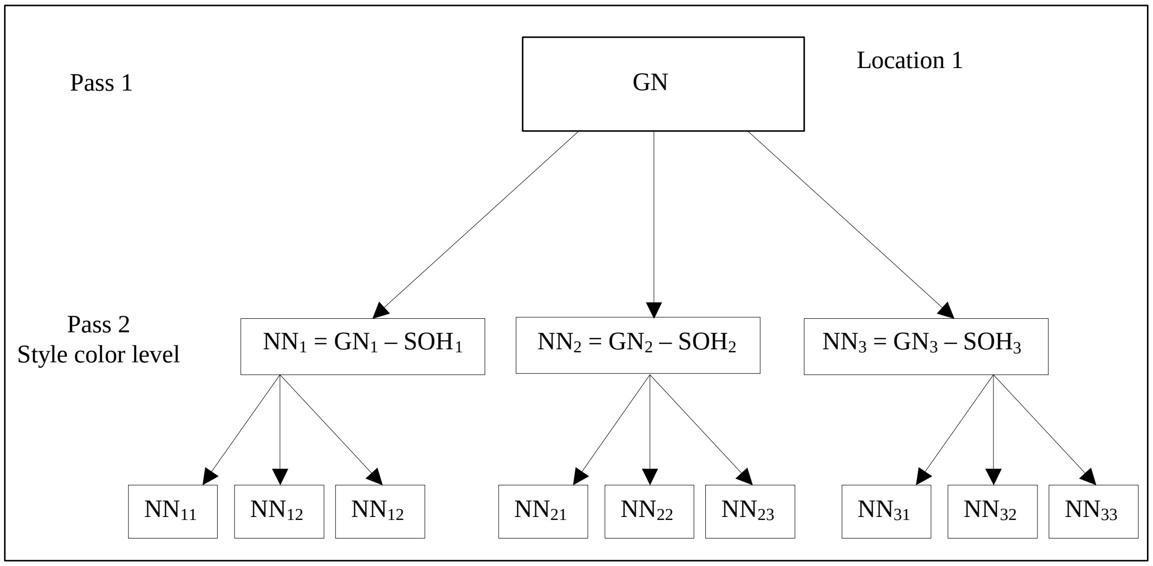
Pass 2
Divide NN_1 among all items (X) at a location [NN_{1x}=NN_1/3]
NN_{11} = Cascaded need for item 1 at the location
NN_{12} = Cascaded need for item 2 at the location
NN_{13} = Cascaded need for item 3 at the location

Each item-location still has individual SOH.
SOH_{21} = SOH for item 1 at the location
SOH_{22} = SOH for item 2 at the location
SOH_{23} = SOH for item 3 at the location

Need applied to each individual item at the location
$NN_{21} = NN_{11} - SOH_{21}$
$NN_{22} = NN_{12} - SOH_{22}$
$NN_{23} = NN_{13} - SOH_{23}$

Fashion cascade

Explanations of Pass 1, Pass 2, and Pass 3 (not shown in the diagram) follow the diagram.



Fashion cascade

Pass 1
GN=Gross need at a location

Pass 2
Divide GN among all styles at given location to get GN, and subtract SOH, for each style-color to get NN_x
$NN_1 = GN_1 - SOH_1$
$NN_2 = GN_2 - SOH_2$
$NN_3 = GN_3 - SOH_3$

Pass 3
Explode NN_x out to sizes to get applied net need NN_{xy}

Style 1	Style 2	Style 3
NN_{11}	NN_{21}	NN_{31}
NN_{12}	NN_{22}	NN_{32}
NN_{13}	NN_{23}	NN_{33}

Prepack algorithm

Retailers sometimes want to configure the number of prepacks and the configuration of prepacks that are to be used to provide merchandise to stores. The benefit of optimizing prepack definitions is reduced warehouse handling costs. More efficient packs result in better in-store levels using packs purchased, and less time and money spent breaking packs at season's end.

In these instances, a prepack algorithm provides suggested prepack configurations that will most closely fit store needs. The prepack configuration algorithm is a different algorithm than the one called out earlier in the diagram, "Retek Allocation calculation queue process". The prepack algorithm does *not* write any transactions inside the merchandising system.

The following example illustrates the prepack algorithm:

The table below shows item-store need, which was calculated as the product of style-store need and a size curve.

Item need				
	S	M	L	Total
Store 1	30	75	45	150
Store 2	30	45	75	150
Store 3	20	30	50	100

The need represents the ideal allocation. Configuring two prepacks allows that need to be satisfied, assuming that the working environment is pre-season mode and that the supplier allows for the specification of prepack configurations.

For a given set of items, and a corresponding item-store allocation such as the need above, the user specifies some values:

- How many prepacks would you like to create?
- What is the minimum and maximum size for these, in total items?

For this example, the values selected are 2, 12, and 12.

Prepack configuration			
	# Packs	Min	Max
Item Set	2	12	12

Using the same logic that the system utilizes when engaged in a prepack calculation, the system evaluates the possible prepack combinations to find which allows for the best allocation for the need defined.

Prepack matrix			
	S	M	L
Pack 1	3	9	0
Pack 2	2	2	8

By running the allocation above with this prepack matrix, the following allocation can be achieved.

Pack allocation		
	Pack 1	Pack 2
Store 1	7	6
Store 2	3	9
Store 3	2	6

At a style level, the following values more closely satisfy the stores' total need.

Style allocation		
	Need	Allocation
Store 1	150	156
Store 2	150	144
Store 3	100	96

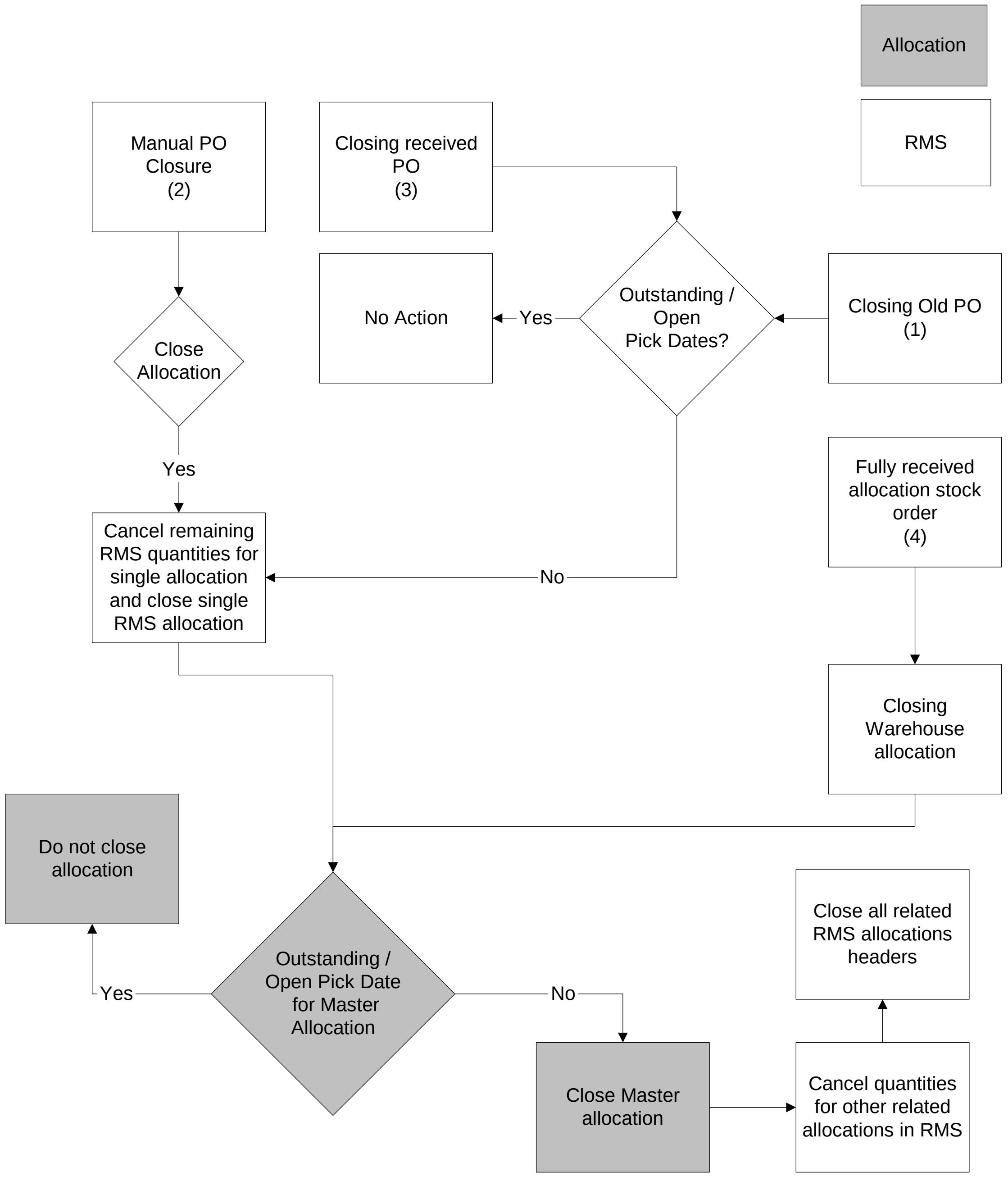
At an item level, the results for medium and large items are very similar, but for the small item, the result is much better. Need by store is 30 30 20, and the result, 33 27 18, is better than 39 39 27.

Item allocation			
	S	M	L
Store 1	33	75	48
Store 2	27	45	72
Store 3	18	30	48

A retailer could now communicate the ideal configuration to the supplier. The configurations can be set up in the merchandising system and used when creating the purchase order.

Closing allocations

This section addresses the four possible methods of closing allocations. The diagram below illustrates the dependencies between Retek Allocation and RMS within the context of when and how allocations are closed. Explanations for the numbers in the diagram follow it. Note that the closure of the master allocation in Retek Allocation entails ‘all or nothing’ processing logic.


Allocation
RMS
Manual PO Closure (2)
Closing received PO (3)
Close Allocation
No Action
Yes
Outstanding / Open Pick Dates?
Closing Old PO (1)
Yes
Cancel remaining RMS quantities for single allocation and close single RMS allocation
No
Fully received allocation stock order (4)
Closing Warehouse allocation
Do not close allocation
Yes
Outstanding / Open Pick Date for Master Allocation
No
Close all related RMS allocations headers
Close Master allocation
Cancel quantities for other related allocations in RMS

Retek Allocation and RMS in the context of the four methods of allocation closures

1 For purchase orders closed via batch functionality
Allocations attached to these closed purchase orders are closed if the ‘pick not after’ days is less than the current date. RMS cancels the associated quantities on the allocation and closes the single allocation. An attempt is made to delete the master allocation. An allocation trigger verifies whether the entire master allocation can be closed based on whether there are no other outstanding / open ‘pick not after’ date records. If the entire master allocation can be closed, all the quantities remaining for related allocations in RMS are cancelled, and the allocations are closed if no quantities are in transit. If the master allocation cannot be closed, there is no further action.

2 For purchase orders closed manually online
If allocations exists when the user attempts to cancel all items, a message offers the user an option to cancel the associated allocations or not. This message also appears when the user attempts to ‘delete order’. This action only cancels remaining quantities for the associated RMS allocation. An attempt is made to delete the master allocation. An allocation trigger verifies whether the entire master allocation can be closed based on whether there are no other outstanding / open ‘pick not after’ date records. If the entire master allocation can be closed, all the quantities remaining for related allocations in RMS are cancelled, and the allocations are closed if no quantities are in transit. If the master allocation cannot be closed, there is no further action.

3 For fully received purchase orders
Allocations attached to received purchase orders are closed if the release date and the ‘pick not after’ days is less than the current date. RMS cancels the associated quantities on the allocation and closes the single allocation. An allocation trigger verifies whether the entire master allocation can be closed. If there are no other outstanding / open ‘pick not after’ date records, the master allocation can be closed (and as such all the quantities remaining for related allocations in RMS are cancelled and the allocations are closed if no quantities are in transit). If the master cannot be closed, there is no further action.

4 For fully received allocation stock orders
An allocation trigger verifies whether the entire master allocation can be closed. If the master cannot be closed, there is no further action.

Chapter 6 – RETL batch processing

The module works in conjunction with the Retek Extract Transform and Load (RETL) framework. This architecture optimizes a high performance data processing tool that allows database batch processes to take advantage of parallel processing capabilities.

The RETL framework runs and parses through the valid operators composed in XML scripts.

This chapter provides an overview of Retek Allocation RETL processing and defines the export file from Curve to Retek Allocation that is used when exporting Curve values. More information about the RETL tool is available in the latest RETL Programmer’s Guide.

Note: In this chapter, some examples refer to RETL programs that are *not* related to Retek Allocation. References to these programs are included for illustration purposes only.

Functional overview

The extract from Curve may contain up to four levels of profile. They consist of the department level, class level, subclass level and item level. All of these levels are contained in a single normalized file. Each record in the file has a dedicated ‘space’ and distinct position for department, class, subclass, item, store, diff1, diff2, diff3, diff4 and size profile qty values. It is crucial that the records are mapped using the correct positions and space/padding rules for each data value.

Regardless of the level of profile, each record must include a store, diff value in one of the four diff value fields, and a quantity value.

Department-level profiles include a department data value in the dedicated department field. The class, subclass and item fields do not contain any values. They remain empty.

Class-level profiles include a department and class data value in the dedicated department and class fields. The subclass and item fields do not contain any values. They remain empty.

Subclass-level profiles include a department, class and subclass data value in the dedicated department, class and subclass fields. The item fields do not contain any values. They remain empty.

All of the department, class and subclass record exports contain only the non-aggregate diff values mapped from the specific diff value in ITEM_MASTER to the corresponding diff value in the export file. It is crucial that the non-aggregate diffs are mapped to the correct diff_id in the export file.

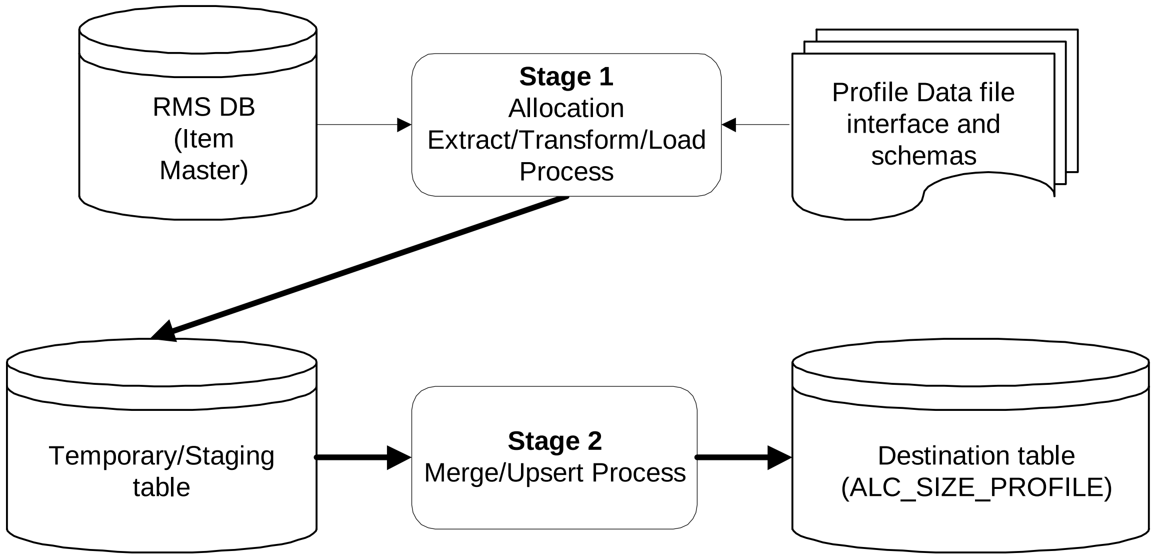
Item-level profiles include transaction level item data values (fashion SKUs) in the dedicated item field. The department, class and subclass fields do not contain any values. They remain empty. The item level export records contains both the aggregate and non-aggregate diff values mapped from the specific diff id in ITEM_MASTER to the associated diff position in the export file.

All the data values must start in the beginning of the corresponding field, and padding comes after the data to fill all the dedicated space for that data field.

RETL batch processing architecture

The diagram below illustrates the extraction processing architecture. The size profile architecture adheres to what is shown in the diagram:

The architecture relies upon two distinct stages, each of which is described in the passages that follow.


RMS DB (Item Master)
Stage 1 Allocation Extract/Transform/Load Process
Profile Data file interface and schemas
Temporary/Staging table
Stage 2 Merge/Upsert Process
Destination table (ALC_SIZE_PROFILE)

RETL processing architecture

Processing stage 1

Stage 1 involves importing profile data and looking up required information in the RMS ITEM_MASTER table (item-level profiles only). The resulting output from this stage is a temporary table that contains any item-level and department/class/subclass-level profiles.

The detailed flow is as follows:

1 Insert dept-level profiles directly into the staging table.

2 Insert class-level profiles directly into the staging table.

3 Insert subclass-level profiles directly into the staging table.

4 The item-level profiles require lookups with the ITEM_MASTER table. The processing logic for transaction-level items is to do a three-way left outerjoin on the ITEM_MASTER table to retrieve each parent and grandparent item aggregate indicators. The Item file then lookups its item id in the item master table and uses the STYLE set as the parent or grandparent item id whose ITEM_AGGREGATE_IND = ‘Y’. These item level profiles are then inserted into the staging table.

Error handling

Any item records that do not have a parent and grandparent are flagged as warnings. Any items in the incoming data file that do not match an item in the ITEM_MASTER table are flagged as errors.

Processing stage 2

Stage 2 involves inserting and updating the profile records into the final destination ALC_SIZE_PROFILE table.

The detailed processing is as follows:

1 Update quantity when matched department, class, subclass, style, store, size1, size2, size3, size4.

2 Otherwise, insert record.

Installation

Select a directory where you would like to install Retek Allocation RETL. This directory (also called MMHOME) is the location from which all of the Retek Allocation RETL files are extracted.

The following code tree is utilized for the RETL framework during the extractions, transformations and loads and is referred to in this documentation.

```
<base directory (MMHOME)>
         /data
                     /error
         /log
      /rfx
         /bookmark
      /etc
         /lib
         /schema
      /src
```

Configuration

RETL

Before trying to configure and run Retek Allocation RETL, install RETL version 11.1 or later, which is required to run Retek Allocation RETL. Run the ‘verify_retl’ script (included as part of the RETL installation) to ensure that RETL is working properly before proceeding.

RETL user and permissions

Retek Allocation RETL should be installed and run as the RETL user. Additionally, the permissions should be set up as per the RETL Programmer’s Guide. Retek Allocation RETL reads, creates, deletes and updates data for tables. If these permissions are not set up properly, processing fails.

Environment variables

See the RETL Programmer’s Guide for RETL environment variables that must be set up for your version of RETL. You will need to set MMHOME to your base directory for Retek Allocation RETL. This is the top level directory that you selected during the installation process (see the section, ‘Installation’, above). In your .kshrc, you should add a line such as the following:

```
export MMHOME=<base directory for RMS RETL>
```

alc_config.env settings

On the Retek Allocation side, make sure to review the environmental parameters in the alc_config.env file before executing the batch module. Depending upon your local settings, the variables may need to be changed.

Configure RETL

1 Log in to the Unix server with a Unix account that runs the RETL scripts.

2 Change directories to $MMHOME/rfx/etc.

3 Modify the alc_config.env script:

   a Change the DBNAME variable to the name of the Retek Allocation database. For example:

   ```
   export DBNAME=int9i
   ```

   b Change the RMS_OWNER variable to the username of the Retek Allocation schema owner. For example:

   ```
   export RMS_OWNER=steffej_rms1011
   ```

   c Change the BA_OWNER variable to the username of the Retek Allocation batch user. For example:

   ```
   export BA_OWNER=rmsint1011
   ```

Also, you must set up the environment variable PASSWORD in either the alc_config.env, .kshrc or some other location that can be referenced. In the example below, adding the line to the alc_config.env causes the password ‘mypasswd’ to be used to log into the database:

```
export PASSWORD=mypasswd
```

Running the module

Schema file

RETL uses a schema file to specify the format of an incoming or outgoing dataset. The schema file defines each column’s data type and format, which is then used within RETL to format/handle the data. Schema file names are hard-coded within each module because they do not change on a day-to-day basis. All schema files end with ‘.schema’ and are placed in the ‘rfx/schema’ directory. For more information about schema files, see the latest RETL Programmer’s Guide.

The schema file for the Retek Allocation module is named profile.schema and is shown later in this chapter.

Mandatory multi-threading and command line parameters

In contrast to the way in which multi-threading is defined in Unix, Retek Allocation uses 'multi-threading' to refer to the running of a single RETL program multiple times on separate groups of data simultaneously. Multi-threading can reduce the total amount of processing time.

For this Retek Allocation module, multi-threading is *mandatory*, and the file-based module has to be run once for each input file.

The alcl_size_profile module requires the following two input parameters:

- The incoming uniquely named data file(s) from the planning solution.
- The uniquely named thread number. Note that the thread number is used internally and is *not* related to any output file or table name.

The following example illustrates a scenario in which the retailer runs the alcl_size_profile.ksh module three times for three input files:

```
alcl_size_profile.ksh  ${MMHOME}/data/alc_size_profile.dat.1 1
alcl_size_profile.ksh  ${MMHOME}/data/alc_size_profile.dat.2 2
alcl_size_profile.ksh  ${MMHOME}/data/alc_size_profile.dat.3 3
```

Program features

The extraction programs are written in the RETL framework and include the following features:

- Program return code
- Program status control files
- Restart and recovery
- Message logging
- Program error file
- Reject files

Program return code

RETL programs use one return code to indicate successful completion. If the program successfully runs, a zero (0) is returned. If the program fails, a non-zero is returned.

Program status control files

To prevent a program from running while the same program is already running against the same set of data, the Retek Allocation RETL code utilizes a program status control file. At the beginning of each module, alc_config.env is run. It checks for the existence of the program status control file. If the file exists, then a message stating, '${PROGRAM_NAME} has already started', is logged and the module exits. If the file does not exist, a program status control file is created and the module executes.

If the module fails at any point, the program status control file is not removed, and the user is responsible for removing the control file before re-running the module.

File naming conventions

The naming convention of the program status control file allows a program whose input is a text file to be run multiple times at the same time against different files.

The name and directory of the program status control file is set in the configuration file (alc_config.env). The directory defaults to $MMHOME/error. The naming convention for the program status control file itself defaults to the following dot separated file name:

- The program name
- The output filename, if one is specified on the command line
- 'status'
- The business virtual date for which the module was run

For example, the program status control file for the invildex program would be named as follows for the batch run of January 5, 2001:

```
$MMHOME/error/invildex.invilddm.txt.status.20010105
```

Retek Allocation RETL restart and recovery

The Retek Allocation RETL module imports data from a flat file, performs transformations if necessary and then loads the data into the applicable Retek Allocation table.

This module uses a single RETL flow and does not require the use of restart and recovery. If the extraction process fails for any reason, the problem can be fixed, and the entire process can be run from the beginning without the loss of data. For a module that takes a text file as its input, the following two choices are available that enable the module to be re-run from the beginning:

1 Re-run the module with the entire input file.

2 Re-run the module with only the records that were not processed successfully the first time and concatenate the resulting file with the output file from the first time.

Message logging

Message logs are written daily in a format described in this section.

Daily log file

Every RETL program writes a message to the daily log file when it starts and when it finishes. The name and directory of the daily log file is set in the configuration file (alc_config.env). The directory defaults to $MMHOME/log. All log files are encoded UTF-8.

The naming convention of the daily log file defaults to the following 'dot' separated file name:

- The business virtual date for which the module is run
- '.log'

For example, the location and the name of the log file for the business virtual date of January 5, 2001 would be the following:

```
$MMHOME/log/20010105.log
```

Format

As the following examples illustrate, every message written to a log file has the name of the program, a timestamp, and either an informational or error message:

```
cusdemogdm 13:20:01: Program Starting...
cusdemogdm 13:20:05: Build update and insert data.
cusdemogdm 13:20:13: Analyze table rdw10dev.cust_demog_dm_upd
cusdemogdm 13:20:14: Insert/Update target table.
cusdemogdm 13:20:23: Analyze table rdw10dm.cust_demog_dm
cusdemogdm 13:20:27: Program Completed...
```

If a program finishes unsuccessfully, an error file is usually written that indicates where the problem occurred in the process. There are some error messages written to the log file, such as ‘No output file specified’, that require no further explanation written to the error file.

Program error file

In addition to the daily log file, each program also writes its own detail flow and error messages. Rather than clutter the daily log file with these messages, each program writes out its errors to a separate error file unique to each execution.

The name and directory of the program error file is set in the configuration file (alc_config.env). The directory defaults to $MMHOME/error. All errors and all routine processing messages for a given program on a given day go into this error file (for example, it will contain both the stderr and stdout from the call to RETL). All error files are encoded UTF-8.

The naming convention for the program’s error file defaults to the following ‘dot’ separated file name:

- The program name
- The first filename, if one is specified on the command line
- The business virtual date for which the module was run

For example, all errors and detail log information for the slsilddm program would be placed in the following file for the batch run of January 5, 2001:

```
$MMHOME/error/slsildmdm.slsildmdm.txt.20010105
```

Retek Allocation reject files

The Retek Allocation module may produce a reject file if it encounters data related problems, such as the inability to find data on required lookup tables. A given module tries to process all data and then indicates that records were rejected. All data problems are thus identified in one pass and corrected. The module can then be re-run to successful completion. If a module does reject records, the reject file is not removed. The user is responsible for removing the reject file before re-running the module.

Typical run and debugging situations

The following examples illustrate typical run and debugging situations for each type of program. The file names referenced in the example below (log, error, and so on) assume that the module is run on the business virtual date of March 9, 2001.

Example

Run alcl_size_profile.ksh:

1 Change directories to `$MMHOME/rfx/src`.

2 At a Unix prompt, enter:

```
%alcl_size_profile.ksh <input datafile 1> <thread #>


...
```

If the module runs successfully, the following results:

- Log file: Today's log file, 20010309.log, contains the messages "Program started …" and "Program completed successfully" for alcl_size_profile.
- Data: The ALC_SIZE_PROFILE table exists in the Retek Allocation database and contains the extract records.
- Error file: The program's error file, alcl_size_profile.20010309, contains the standard RETL flow (ending with "All threads complete" and "Flow ran successfully") and no additional error messages.
- Program status control: The program status control file, alcl_size_profile.status.20010309, does not exist.
- Reject file: No reject files exist.

If the module does not run successfully, the following results:

- Log file: Today's log file, 20010309.log, does not contain the "Program completed successfully…" message.
- Data: The ALC_SIZE_PROFILE table exists in the Retek Allocation database but may not contain all the records from the profile file interface.
- Error file: The program's error file, alcl_size_profile.20010309, may contain an error message.
- Program status control: The program status control file, alcl_size_profile.status.20010309, exists.
- Reject file: The reject file, alcl_size_profile.status.20010309, does not exist because this module does not reject records.
- Bookmark file: The bookmark file, alcl_size_profile.bkm.20010309, does not exist because this module does not utilize restart and recovery.

Re-run the module:

1 Determine and fix the problem causing the error.

2 Remove the program’s status control file.

3 Change directories to $MMHOME/rfx/src. At a Unix prompt, enter:

```
% alcl_size_profile.ksh <input datafile 1> <thread #>
```

Retek Allocation program reference

This section serves as a reference to the Retek Allocation program.

By reviewing this section and the section, ‘API flat file specification’, the retailer should be able to track, down to the table and column level, all the extraction data that flows into Retek Allocation.

Program Name	Tables/Files Extracted	Fields Extracted	Target File or Table	Target Field	Field Type	Field Length	Notes
alcl_size_profile.ksh	item_master	item	Alc_size_profile	style	VARCHAR2 (25)	25	
		item_parent					Use item_parent as style if Item_parent_aggregate_ind = ‘Y’
		item_grandparent					Use item_grandparent as Style if item_grandparent_ aggregate_ind = ‘Y’
		item_aggregate_ind					
		item_parent_aggregate_ind					
		item_grandparent_ aggregate_ind					
	profile file	dept		dept	NUMBER(4)	4	
		class		class	NUMBER(4)	4	

Program Name	Tables/Files Extracted	Fields Extracted	Target File or Table	Target Field	Field Type	Field Length	Notes
		subclass		subclass	NUMBER(4)	4	
		store		store	NUMBER(10)	10	
		size1		size1	VARCHAR2 (10)	10	
		size2		size2	VARCHAR2 (10)	10	
		size3		size3	VARCHAR2 (10)	10	
		size4		size4	VARCHAR2 (10)	10	
		qty		qty	NUMBER(12,4)	17	

Application programming interface (API) specification

File layout

- ABBREVIATION: An abbreviation of the value in the hierarchy (for example, D for department).
- ITEM_MASTER VALUE: Field data on the RMS ITEM_MASTER table.
- ALC_SIZE_PROFILE VALUE: Field data on the Retek Allocation ALC_SIZE_PROFILE table.
- SQL DATA TYPE: SQL data types identifies one of three valid data types and the maximum length possible for a field. A field may not exceed this length. Data types include the following:
  - Character: Can hold letters (a,b,c…), numbers (1,2,3…), and special characters ($,#,&…)
  - Numbers: Can hold only numbers (1,2,3…)
  - Date: Holds a specific year, month, day combination
- RETL DATA TYPE: The data type identified in the schema file.
- NULLABLE?: Identifies whether the field can hold a null value. This section holds either a 'yes' or a 'no'. A 'yes' signifies the field may not hold a null value. A 'no' signifies the field may, but is not required to, hold a null value.

Profile record definitions

ABBREVIATION	ITEM_MASTER VALUE	ALC_SIZE_ PROFILE VALUE	SQL DATA TYPE	RETL DATA TYPE	NULLABLE?
D	DEPT	DEPT	NUMBER (4)	Int16 (len=4)	Yes
C	CLASS	CLASS	NUMBER (4)	Int16 (len=4)	Yes
S	SUBCLASS	SUBCLASS	NUMBER (4)	Int16 (len=4)	Yes
I	ITEM	STYLE	VARCHAR2 (25)	String (len=25)	Yes
L	n/a	STORE	NUMBER (10)**	Int64 (len=10)	No
1	DIFF_1	SIZE1	VARCHAR2 (10)	String (len=10)	Yes
2	DIFF_2	SIZE2	VARCHAR2 (10)	String (len=10)	Yes
3	DIFF_3	SIZE3	VARCHAR2 (10)	String (len=10)	Yes
4	DIFF_4	SIZE4	VARCHAR2 (10)	String (len=10)	Yes

ABBREVIATION	ITEM_MASTER VALUE	ALC_SIZE_ PROFILE VALUE	SQL DATA TYPE	RETL DATA TYPE	NULLABLE?
Q	n/a	QTY	NUMBER (12,4)	Dfloat (len=17)	No
“BLANKS”	Null values	Null values	Null values	Null values	

Extract profile file format

Record format:

```
D...C...S...I........................L.........1.........2.........3.........4.........Q.............
1...5...9...13.......................38........48........58........68........78........88.........105
```

Where D, C, S, I, L, 1, 2, 3, 4, Q are as defined in Table 1, and numbers are the appropriate fixed-width positions of each field. The ellipses are used to denote field lengths. In the examples below, diff1 is color and is an aggregate diff. Diff2 is size and size the non-aggregate diff.

Total record length = 104

Example

```
[dept only]
```

```
1001                                 1000000000          XXL                           0.2
1001                                 1000000000          XL                            0.3
1001                                 1000000000          L                             0.2
1001                                 1000000000          M                             0.1
1001                                 1000000000          S                             0.1
1001                                 1000000000          XS                            0.1
```

```
[dept, class level]
```

```
10012000                             1000000000          XXL                           0.2
10012000                             1000000000          XL                            0.3
10012000                             1000000000          L                             0.2
10012000                             1000000000          M                             0.1
10012000                             1000000000          S                             0.1
10012000                             1000000000          XS                            0.1
```

```
[dept, class, subclass level]
```

```
100120002050                         1000000000          XXL                           0.2
100120002050                         1000000000          XL                            0.3
100120002050                         1000000000          L                             0.2
100120002050                         1000000000          M                             0.1
100120002050                         1000000000          S                             0.1
100120002050                         1000000000          XS                            0.1
```

```
[item level]
            1000000000              1000000000Color02   XXL                             0.2
            1000000000              1000000000Color02   XL                              0.3
            1000000000              1000000000Color02   L                               0.2
            1000000000              1000000000Color02   M                               0.1
            1000000000              1000000000Color02   S                               0.1
            1000000000              1000000000Color02   XS                              0.1
            1000000100              1000000000Color01   XXL                             0.2
            1000000100              1000000000Color01   XL                              0.3
            1000000100              1000000000Color01   L                               0.2
            1000000100              1000000000Color01   M                               0.1
            1000000100              1000000000Color01   S                               0.1
            1000000100              1000000000Color01   XS                              0.1
            1000000100              1000000000Color02   XXL                             0.2
            1000000100              1000000000Color02   XL                              0.3
            1000000100              1000000000Color02   L                               0.2
            1000000100              1000000000Color02   M                               0.1
            1000000100              1000000000Color02   S                               0.1
            1000000100              1000000000Color02   XS                              0.1
```

Note: The text in brackets ([…]) is for illustration/commentary purposes only and should not exist in the file.

Schema file (profile.schema)

This section describes the RETL schema file (profile.schema) used in the RETL script that loads the Curve export file into Retek Allocation’s ALC_SIZE_PROFILE table.

```
<RECORD type="fixed" len="104" final_delimiter="0x0A">

  <!-- start pos 1  --> <FIELD name="DEPT" len="4" datatype="int16"
nullable="true" nullvalue=""/>

  <!-- start pos 5  --> <FIELD name="CLASS" len="4" datatype="int16"
nullable="true" nullvalue=""/>

  <!-- start pos 9  --> <FIELD name="SUBCLASS" len="4" datatype="int16"
nullable="true" nullvalue=""/>

  <!-- start pos 13 --> <FIELD name="ITEM" len="25" datatype="string"
nullable="true" nullvalue=""/>

  <!-- start pos 38 --> <FIELD name="STORE" len="10" datatype="int64"
nullable="false"/>

  <!-- start pos 48 --> <FIELD name="SIZE1" len="10" datatype="string"
nullable="true" nullvalue=""/>

  <!-- start pos 58 --> <FIELD name="SIZE2" len="10" datatype="string"
nullable="true" nullvalue=""/>

  <!-- start pos 68 --> <FIELD name="SIZE3" len="10" datatype="string"
nullable="true" nullvalue=""/>

  <!-- start pos 78 --> <FIELD name="SIZE4" len="10" datatype="string"
nullable="true" nullvalue=""/>

  <!-- start pos 88 --> <FIELD name="QTY" len="17" datatype="dfloat"
nullable="true" nullvalue=""/>

  <!-- end pos 105  -->

</RECORD>
```

Chapter 7 – Java batch process

Retek Allocation contains one batch process that is run in Java. This batch process deletes allocation records that have been marked as ‘delete’ in the database.

Characteristics of the Java batch process

Note the following characteristics of Retek Allocation’s Java batch process:

- It is not accessible through a graphical user interface (GUI).
- It is scheduled by the retailer.
- It is designed to process large volumes of data.
- It should only be executed during ‘off-hours’ (that is, during a time when users are not in the system such as nights).

Java batch name and Java package

The following table describes Retek Allocation’s batch process and its associated Java package.

Batch name	Batch process	Package
Batch purge	Purge.java	com.retek.alloc.batch

Functional description

The following table includes a description of Retek Allocation’s batch process.

Batch processes	Details
Batch purge (Purge.java)	The batch purging process deletes Retek Allocation records that have been marked as ‘delete’ in the database. This Retek Allocation deletion process must be run only during ‘off-hours’ (that is, during a time when users are not using the online Retek Allocation application system such as nights).

Running a Java-based batch process

Scheduler and the command line

If the retailer uses a scheduler, arguments are placed into the scheduler.

If the retailer does not use a scheduler, arguments must be passed in at the command line.

For Unix systems, the Java process is scheduled through an executable shell script (.sh file). For Windows systems, the Java process is scheduled through an executable batch file (.bat file).

Retek provides sample shell scripts (.sh files) and batch files (.bat files). These sample shell scripts must be modified according to the retailer's installation. They perform the following internally:

- Set up the Java runtime environment before the Java process is run.
- Trigger the Java batch process.

Summary of executable files

To 'kick off' the deletion process, use one of the scripts shown in the table below.

Executable shell scripts (UNIX)	Executable batch file for windows
setPurge.sh *and* purge.sh	purge.bat (Windows)