Retail passion. Innovative solutions.

Retek® Merchandising System™ 9.0.17

Operations Guide Addendum

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

Printed in the United States of America.

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
Hong Kong	800 96 4262
Korea	00 308 13 1342
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

Recommended Order Quantity [ociroq.c]

Design Overview

The purpose of this batch program is to call the PL/SQL packages used to calculate the Net Inventory position of the items on replenishment. The results are stored in the database to be used by REQEXT (Item Requisition Extraction).

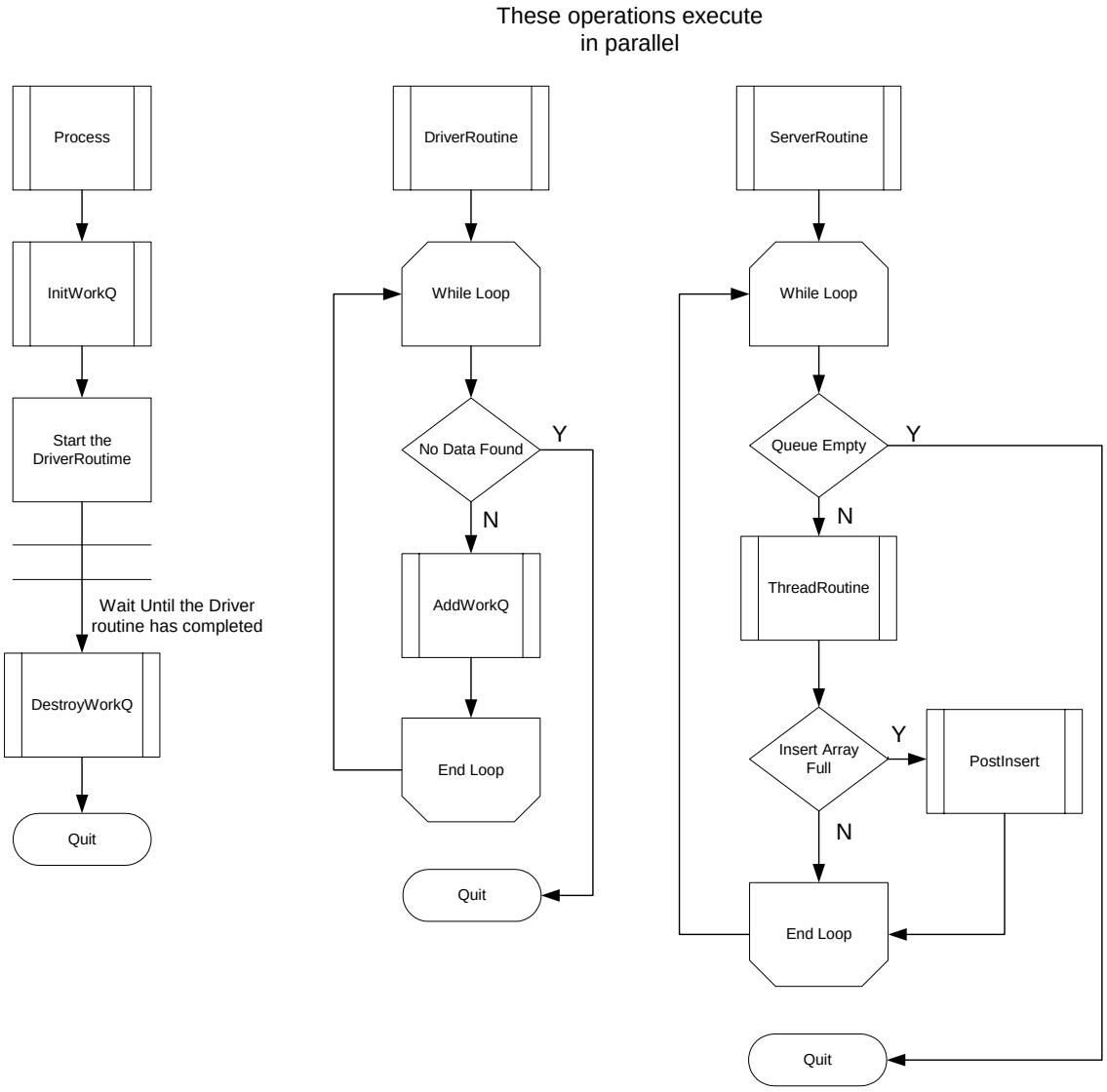
Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Prepost (ociroq pre), rplatupd, rpladjf and rpladjs need to run before reqext so that all replenishment calculation attributes are up to date. Posupld needs to run before reqext so that all stock information is up to date.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	POSIX threads The restart_control.num_threads will control the number of POSIX threads that are run within ociroq. The batch program ociroq.c itself will only need to be run with one thread.

Restart Recovery

The program processes all items on repl_day for the current day. If the program fails, the rpl_net_inventory_tmp table should be truncated prior to restarting (prepost ociroq pre)

Program Flow



Shared Modules

GET_REPL_ORDER_QTY_SQL.REPL_METHOD: Stored PL/SQL procedure for calculating the ROQ of an item at a location.

REPLENISHMENT_SQL.GET_STORE_REVIEW_TIME: Stored PL/SQL procedure for calculating the time between scheduled shipments to a store from a warehouse. This time is used by GET_REPL_ORDER_QTY_SQL in its calculations.

OciInitLogon(): C library function that validates the program usage and performs initial environment set-up; including opening the daily log file for writing. It also calls OciConnect().

OciConnect(): C library function that connects to the database and performs some initial environment set-up. This function calls numerous OCI library routines that create the appropriate OCI handles.

OciDisconnet(): C library function that disconnects from the database and free the OCI handles created by the OCIConnect() call.

ReportError(): C library function that calls the OCIErrorGet() function and returns the appropriate error message.

WriteError(): C library function writes the appropriate message to the error file; indicating the type of error encountered and the Oracle Error number and message.

LogMessage(): C library function writes the appropriate message to the log file; indicating start time, end time and time of failure if the program terminated with errors.

RaiseError(): C library function responsible for passing the error code back to the parent process to ensure correct error handling.

Data Structures

repl_info_struct: Holds information fetched from the driving cursor.

GetOltsStruct: Holds the information passed into and returned from the REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME procedure.

GetReplStruct: Holds the information passed into and returned from the GET_REPL_ORDER_QTY_SQL.REPL_METHOD procedure.

InsertStruct: Used to buffer the inserts into the rpl_net_inventory_tmp table.

DomainStruct: Used to cache forecasting domain information.

Driver_Info: Used by the Driver thread as a container to pass in all the appropriate parameters to the thread routine.

Thread_Info: Used by the Work Queue threads as a container to pass in all the appropriate parameters to the thread routine.

WorkQueue_List: This linked list is used to hold the actual data fetched by the Driver thread to be then consumed by the Work Queue threads.

WorkQueue_Info: Holds all the Work Queue thread control information.

domain_struct: Used to cache forecasting domain information.

Function Level Description

General Controlling Functions

main()

The standard Retek main function, this calls init(), process() and final(), and posts messages to the daily log files.

init()

Fetches system-level global variables and calls other functions to fetch additional global level data; GetMaxCounter(), GetStoreCount() and LoadDomainInfo()

Process()

Controls the bulk of the processing. It initializes the Work Queue threads, creates the Driver thread and waits until the Driver thread has completed prior to calling the DestroyWorkQ() and ThreadCleanUp() functions.

final()

The standard Retek final function, this closes down the process and posts messages to the daily logs.

Thread Controlling Functions

InitWorkQ()

Initializes the specific POSIX Pthread library variables used by the Work Queue threads. It then initializes the WorkQueue_Info structure variables and creates the specified number of threads; Each thread calls the ServerRoutine(). The function performs a loop, allocating memory for each threads data structures and connects each thread to the database by calling the OciConnect() library routine. Finally it calls the DefineWorkerStmts() function.

ServerRoutine()

Controls the consumption of the WorkQueue_List. Each Work Queue thread monitors and consumes data from the list until they are instructed to quit or the queue is empty. Initially while the queue is empty the threads poll the queue every 2 seconds checking the status. All thread synchronization is handled by the use of a mutually exclusive lock (mutex). Each node taken from the list is passed to the ThreadRoutine() function.

ThreadRoutine()

Executed by the Work Queue threads; it calls the PL/SQL packages and buffers the result in the InsertStruct. When an individual thread reaches the MAX_INSERT_SIZE the buffer is inserted into the rpl_net_inventory_tmp table.

GetOlts()

Called by the ThreadRoutine(), this function calls the REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME PL/SQL package.

GetRepl()

Called by the ThreadRoutine(), this function calls the GET_REPL_ORDER_QTY_SQL.REPL_METHOD PL/SQL package.

DriverRoutine()

Executed by the Driver thread; it's responsible for defining and fetching the driving cursor and adding the batch to the queue. The execution of this function by a thread allows it to run in parallel with the Work Queue threads. The Work Queue threads will start after the first batch has been placed on the WorkQueue_List.

AddWorkQ()

Loads the array fetched by the DriverRoutine() onto the WorkQueue_List. It allocates memory for each node and will continue to load the queue while the number of records on the queue has not exceeded the MAX_QUEUE_SIZE. All thread synchronization is handled by the use of a mutually exclusive lock (mutex). The function will wait until the queue less than half full prior to recommencing.

DestroyWorkQ()

Waits until all the Work Queue threads have consumed all the data from the list; it then performs some cleanup duties. All thread synchronization is handled by the use of a mutually exclusive lock (mutex).

ThreadCleanUp()

Frees the memory allocated to each threads data structures (including statement handles) and disconnects from the database.

Database DML Handling

PostInsert()

When the Insert buffer reaches the MAX_INSERT_SIZE the array is posted to the database and the work committed.

OCI Statement Functions

DefineDriver()

Performs OCI specific statement set-up; including statement handle preparation, statement handle attribute set-up (pre-fetch size), statement column definition and the array of structure definition (skip size etc.) for the Driving Cursor.

DefineGetRepl() **

Performs OCI specific statement set-up; including statement handle allocation, statement handle preparation and statement column binding for the PL/SQL package call GET_REPL_ORDER_QTY_SQL.REPL_METHOD.

DefineGetOlts() **

Performs OCI specific statement set-up; including statement handle allocation, statement handle preparation and statement column binding for the PL/SQL package call REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME.

DefineInsert() **

Performs OCI specific statement set-up; including statement handle preparation, statement handle attribute set-up (pre-fetch size), statement column definition and the array of structure definition (skip size etc.) for the rpl_net_inventory_tmp insert Statement.

Note: These functions are called for each Work Queue thread. Each thread will have its own database connection and statement handles.

Database Interaction

The following database tables related to the Net Inventory dialog of RMS and the types of access that will be used by this process: *

Table	Select	Insert	Update	Delete
DOMAIN_CLASS	Y	N	N	N
DOMAIN_DEPT	Y	N	N	N
DOMAIN_SUBCLASS	Y	N	N	N
ITEM_SUPP_COUNTRY	Y	N	N	N
PERIOD	Y	N	N	N
REPL_DAY	Y	N	N	N
REPL_ITEM_LOC	Y	N	N	N
RESTART_CONTROL	Y	N	N	N
RPL_NET_INVENTORY_TMP	N	Y	N	N

Table	Select	Insert	Update	Delete
STORE	Y	N	N	N
SYSTEM_OPTIONS	Y	N	N	N
WH	Y	N	N	N
WIN_WH	Y	N	N	N

Note: This list does not include the tables accessed by the PL/SQL package calls executed by this program.

I/O Specification

N/A

Technical Issues

N/A

POS Download [posdnld]

Design Overview

The posdnld program is used to download pos_mods records created in the RMS to the store POS systems. This program has one output file which contains all records for all stores in a given run.

Scheduling Constraints

Processing Cycle:	PHASE 4 (daily)
Scheduling Diagram:	This program is run towards the end of the batch run when all pos_mods records have been created for the transaction day.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

Restart/recovery for this program is set up at the store/upc or sku level. Threading is done by store using the v_restart_store view to thread properly.

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Program Flow

Shared Modules

N/A

Function Level Description

Init

This function initializes restart/recovery for this program. It also opens the output file, retrieves system variables and calls a function to size the arrays used in this program.

Process

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from pos_mods where the pos_mods.store value is greater than zero. Once the records are fetched, the write_rec() function is called to perform processing on them. Restart/Recovery and commiting of records is also performed here.

Write_rec

This function will prepare records for insert into the output file. This program used the Retek standard file format FHEAD, FDETL, FTAIL.

Final

This function will finish restart/recovery logic, close the output file and delete the temporary output file used while the program processes.

Init_format_strs

This function formats the strings for the FHEAD, FDETL, and FTAIL records in the output file.

Init_arrays

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

Resize_arrays

This function increases the memory for the driving cursor array by the size of the restart max counter on restart_control.

Get_upc_type

This function query the upc_ean table to get the var_type given the passed in upc and upc_supp.

I/O Specification

Output file

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Sequenctial number Created by program.	ID of current line being created for output file.
	File Type Definition	Char(4)	POSD	Identifies file as 'POS Download'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'.
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Sequenctial number Created by program.	ID of current line being created for output file.
	Location Number	Number(4)	store identifier	Store identifier from the store table in Retek
	Update Type	Char(1)	update type	Code used for client specific POS system.
	Start_Date	Char(14)	start date	date for the change to take effect at the POS, formatted to 'YYYYMMDDHH24MISS'.
	Upc	Number(13)	upc identifier	the id number of a UPC if a UPC exists on the UPC_EAN for that SKU.

Record Name	Field Name	Field Type	Default Value	Description
	Upc Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item. From the Retek upc_ean table.
	Upc var type	Char(1)	Variable upc type indicator	Identifies what type of variable UPC is being sent. Valid values are 'W'eight, 'P'rice, and NULL.
	Tran Type	Number(2)	transaction type	the transaction type for the record from the Retek pos_mods table.
	SKU	Number(8)	SKU identifier	the id number of a SKU from the Retek desc_look table.
	SKU Description	Char(40)	SKU description	the description of the SKU from the Retek desc_look table.
	Dept	Number(4)	dept id	the id of the dept for the item from the Retek win_skus or rag_style table.
	Class	Number(4)	class id	the id of the class for the item from the Retek win_skus or rag_style table.
	Subclass	Number(4)	subclass id	the id of the subclass for the item from the Retek win_skus or rag_style table.
	New Price	Number(20)	new price	the new price to be taken at the POS. This value is from the Retek pos_mods table.

Record Name	Field Name	Field Type	Default Value	Description
	Multi Units	Number(12)	multi units	Number of multi units
	Multi Units Retail	Number(20)	multi units retail	unit retail for the multi units
	Status	Char(1)	status	Populates if tran_type for the item is 1(new item added) or 25 (change item status) or 26 (change taxable indicator).
	Taxable Indicator	Char(1)	taxable ind	Populates if tran_type for the item is 1(new item added) or 25 (change item status) or 26 (change taxable indicator).
	Promotion Number	Number(4)	promotion number	Promotion number for SKU. This value is from the Retek system.
	Mix Match Number	Number(4)	mix match number	mix match number for SKU. This value is from the Retek ssytem.
	Mix Match Type	Char(1)	mix match type	mix match type (Buy or Get) for SKU. This value is from the Retek system.
	Threshold Number	Number(4)	threshold number	Threshold number for SKU. This value is from the Retek system.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

Item Requisition Extraction [reqext]

Design Overview

Reqext (Item Requisition Extraction) handles automatic replenishment of items from warehouses to stores. It cycles through every item-store combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be transferred to the store (if any). In addition, it distributes this Recommended Order Quantity (ROQ) over any applicable alternate items associated with the item. The program then takes this information and either creates new transfer line items or adds to existing ones.

Alternate items are either simple packs or substitute items. Simple packs are sellable and orderable packs that contain only a single item, such as a six-pack of cola or twelve-pack of socks. Substitute items are items predefined to be interchangeable with the item being replenished (referred to as the master item).

When an item is set up to use simple packs (designated by an indicator on the REPL_ITEM_LOC table), the ROQ must be distributed among these packs according to desirability. If a master item has no simple packs associated with it, it will be requested as itself. If there is only one pack associated with the item (referred to as the primary simple pack), then there is no distribution needed – the item will be transferred in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program! Because the pack distribution depends on access to the huge sales history tables (PACKSTORE_HIST, RAG_SKUS_ST_HIST, WIN_STORE_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

If an item is not set up to use simple packs, the program will see if any substitute items are associated with it. If there are no substitute items associated with the master item, it will be transferred alone. If there are substitute items, they will be fetched into a list and the master item placed at either the head or tail end of the list, depending on the fill priority (set on the SUB_ITEMS_HEAD table). The priority determines which items are transferred first.

No matter what type of alternate items (if any) are used, the program will account for availability when building transfer line items. For simple packs, the share of ROQ allocated to each pack may be decreased or increased if the source warehouse has a shortage of some packs but a surplus of others. For substitute items, transfer quantities are prorated by calculating the ratio of total availability to total need, and items are transferred in order of priority until all need is filled or until no stock is available.

Once the transfer quantity of an item has been calculated, the transfer line item is posted to the database if 1) the actual quantity to transfer is greater than zero, and 2) the replenishment order control indicator for the item-store combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, even if the quantity to order is zero. Whenever a transfer line item is placed, the appropriate item-location table (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH and/or PACKWH) is updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Rplatupd, rpladjf, rpladjs, prepost ociroq and ociroq need to run before reqext so that all replenishment calculation attributes are up to date. Posupld, tsfoupld, tsfiupld, ctniupld, and rcvupld need to run before reqext so that all stock information is up to date. Rplext should run after reqext, since the ROQ for a warehouse is influenced by any transfers created.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	ITEM (partition)

Restart Recovery

The logical unit of work is item, source warehouse. The driving cursor is ordered by item, source warehouse, order control indicator and simple pack indicator. When any of these values change during the course of processing (i.e., the current value is different than that of the previous record), then a transfer will be created, taking total quantities and availability into consideration (see replenish_item(), below).

Program Flow

N/A

Shared Modules

ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL: Stored PL/SQL procedure for calculating the amount of a given item available at a given warehouse.

NEXT_TRANSFER_NUMBER: Stored PL/SQL procedure used for getting the next valid transfer number for use in creating new transfers.

RMS_ROUND_TO_PACKSIZE: Shared C function (see rpl.h) used in rounding an item's quantity up to the size of a simple pack, or for rounding an order quantity up to a receivable pack size.

Data Structures

repl_info_struct: Holds information fetched from the driving cursor.

store_struct: Holds information about item-location combinations, used for ROQ and distribution calculations.

alt_item_struct: Holds information about alternate items associated with a given master item. Used in distribution calculations.

tsfhead_struct: Used to buffer inserts to the TSFHEAD table.

tsfdetail_struct: Used to buffer inserts and updates to the TSFDETAIL table.

item_loc_struct: Used to buffer updates to the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH).

repl_results_struct: Used to buffer inserts to the REPL_RESULTS table.

domain_struct: Used to cache forecasting domain information.

Function Level Description

General Controlling Functions

main()

The standard Retek main function, this calls init(), process() and final(), and posts messages to the daily log files.

init()

Initializes the Restart-Recovery API and fetches system-level global variables.

driving_cursor()

Opens, fetches data from, or closes the driving cursor. This is a support function for process().

process()

This function fetches records from the driving cursor (driving_cursor()), passes them to replenish_item() to perform all appropriate actions, and commits work when appropriate (post_all(), restart_commit()).

replenish_item()

The controlling function for replenishment calculations. This function copies records out of the driving cursor buffer (copy_repl_to_store()), and calculates the ROQ for each record (get_repl_order_qty()). If a change in item, source warehouse, order control indicator, or simple pack indicator has occurred, the appropriate functions are called to calculate distribution of need over all appropriate alternate items and stores, and to place the transfers. If item's ROQ is zero or negative, no mater simple pack indicator is on the master item will be used for replenishment (build_pack_ratio(), calc_pack_dist(), calc_sub_dist()).

place_tsf_line_item()

This function takes a item-location combination and a transfer quantity, and actually builds the transfer line item (handle_tsf()). It then updates the item-location tables to reflect the change in stock (handle_item_loc()), and writes a record to the reporting table (handle_repl_results()) when appropriate.

final()

The standard Retek final function, this closes down the Restart-Recovery API.

Simple Pack Distribution and Transfer

build_pack_ratio()

Calculates distribution of the master item’s recommended order quantity (ROQ) over simple packs. Simple packs are sellable and orderable packs containing only a single item (e.g., six-pack of cola). Since the cost per item will always be less in a pack than singularly, the item will only be ordered in terms of simple packs (if any are applicable). This function tries to divide the total ROQ for the item among all applicable simple packs by using the packs' relative sales history to build a distribution 'mask' containing ratios used to calculate each pack's share of the ROQ. This mask is then adjusted to account for availability (shortages of some packs, surpluses of others).

This function performs the following steps to optimally distribute the ROQ among any and all simple packs:

- If a primary simple pack was defined for this item, that pack will be the only one used to supply the item (add_primary_pack()).
- If no primary pack was defined, the program will build a list of all simple packs associated with the item (get_multi_simple_pack()).
- If no appropriate simple packs are found, the item will be ordered as itself (add_single_item()).
- The historical sales for all simple packs and the master item are added up.
- The ROQ is distributed among the simple packs by taking the ratio of each pack's historical sales to the total historical sales (first_ratio_pass()).
- If the first pass through the list did not account for the entire ROQ because of lack of availability for some packs, the program must keep cycling through the items until it has either distributed the ROQ among all available packs or there is simply no available stock left to supply the need (next_ratio_pass()).

first_ratio_pass()

Performs initial distribution of an item's ROQ among its associated simple packs. Calculates each pack's share as a ratio of its historical sales to the total historical sales (adjust_pack_ratio()). The historical sales of the master item are added to those of the simple pack with the lowest cost to give it a greater share of the ROQ. This is a support function for build_pack_ratio().

next_ratio_pass()

This function readjusts the ratios of still-available packs to try and cover the share of ROQ not yet allocated, still distributing the leftover ROQ proportionally by historical sales. This is a support function for build_pack_ratio().

adjust_pack_ratio()

Sets a simple pack's share of the ROQ to reflect its desirability (in terms of historical sales patterns), adjusting for availability. This is a support function for first_ratio_pass() and next_ratio_pass().

add_primary_pack()

If an item is flagged to have a primary simple pack, that pack is the only one that will be transferred. This function adds the primary pack to the simple pack distribution array and assigns it the full share of the ROQ. This is a support function for build_pack_ratio().

get_multi_simple_pack()

Finds all simple packs associated with a given master item and information about them (historical sales, availability, etc). This is a support function for build_pack_ratio().

get_single_sales_hist()

Gets the historical sales of the master item at all stores supplied by the given warehouse for use in calculating distribution among simple packs. Since the master item will only be transferred as a pack, this sales amount will be added to that of the pack with the lowest cost, increasing its share of the ROQ. This is a support function for build_pack_ratio().

add_single_item()

If an item is flagged to use simple packs, but none are found, it will be ordered as itself. This function adds the master item to the simple pack distribution structure and assigns it the full share of the ROQ. This is a support function for build_pack_ratio().

calc_pack_dist()

Once each simple pack's share of the item's ROQ has been calculated in build_pack_ratio(), this function calculates actual transfer quantities and places the transfer line items (place_tsf_line_item()). The function loops through each pack in the list, calculating the amount of the pack to transfer to each store (calc_pack_tsf_qty()). If the total transfer quantity of the pack exceeds its availability at the warehouse, each store will have its quantity reduced by one receivable pack until a reasonable number has been reached. Finally, a transfer line item is placed for the pack to the store.

calc_pack_tsf_qty()

Calculate the actual quantity to transfer for a store based on an alternate item's share of the ROQ at a store, adjusted for any applicable simple pack and/or shipping pack sizes. This is a support function for calc_pack_dist().

Substitute Item Distribution and Transfer

calc_sub_dist()

Calculate distribution of the ROQ over substitute items. Substitute items are items (selected by the user beforehand) that can be requested in place of a given item to cover situations where availability is too low or demand is too high.

After calling get_sub_items() to generate a list of appropriate items for transfer, the function loops through every item-location combination and performs the following steps to make sure that both need and availability are accounted for when placing transfers from the warehouse to the stores:

- If the total availability of all items in the substitute list cannot cover the full need over all stores, then the ratio of the total availability to the total ROQ is calculated. If total availability *can* cover total ROQ, the ratio is set to 1.
- The initial transfer quantity for the item at the location is calculated as the store's need adjusted by the availability ratio, and rounded up to a receivable pack size.
- If there is not enough of the item available at the warehouse to fill the calculated transfer quantity, the quantity will be decremented to an orderable amount.
- The transfer line item is placed by calling place_tsf_line_item().
- The store's ROQ, total ROQ, availability of the item, and total availability are all decremented by the amount just transferred to prepare for the next item-location's calculation.

get_sub_items()

Retrieves substitute items for the current master item and information about them from the database (receiving pack size, availability, etc.). If the fill priority for this set of items (SUB_ITEMS_HEAD.fill_priority) is set to 'M'aster, the master item will be the first one in the list, and will be used first to fill need. If it is set to 'S'ubstitute, the master item will be placed at the tail end of the list. This is a support function calc_sub_dist().

add_master()

Adds the master item to the appropriate position in the substitutes list. This is a support function for get_sub_items().

shift_subs()

If the fill priority for the substitutes list is set to 'M'aster, the master item must be placed at the head of the list. This function clears out the first position by moving each substitute item 'back' a slot. This is a support function for get_sub_items().

Database DML Handling

post_all()

The DML handling functions (handle_tsf(), handle_item_loc(), handle_repl_results()) normally only post information to the database tables when their respective buffers are full. When a commit point is reached, however, all buffers must be flushed to ensure restartability. This function forces all the buffers to be posted to the database.

handle_tsf()

Controls handling of inserts and updates to the Transfer tables.

add_tsfhead()

Deals with transfer header information. Either finds an appropriate transfer to add line items to (matching to/from locations, department and freight code), or creates a new one. passes back the transfer number for use in add_tsfdetail(). This is a supporting function for handle_tsf().

add_tsfdetail()

Deals with transfer detail information. Either finds an appropriate record on the TSFDETAIL table to add quantity to (matching transfer number and item), or creates a new one if none is found. This is a supporting function for handle_tsf().

get_next_seq_no()

Every line item on a transfer has a unique identifier within that transfer. This function gets the next sequence number for a new line item. This is a supporting function for add_tsfdetail().

post_tsf()

Posts transfer information to the database. Inserts to TSFHEAD, inserts and updates to TSFDETAIL. This is a supporting function for handle_tsf().

handle_item_loc()

Whenever a transfer is created or modified, the source location's reserved quantity and the receiving location's expected quantity must be adjusted to reflect the new stock status. This function controls the handling of updates to the RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH and PACKWH tables.

add_item_loc()

Adds records to arrays for update of expected and reserved quantities on the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH) based on the appropriate item types. This is a support function for handle_item_loc().

post_item_loc()

Posts item-location stock status changes to the database (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH). This is a support function for handle_item_loc().

handle_repl_results()

Controls posting of report information to the REPL_RESULTS table.

add_repl_results()

Adds records to the replenishment results structure for reporting. This is a supporting function for handle_repl_results().

post_repl_results()

Posts replenishment information to the REPL_RESULTS table. This is a supporting function for handle_repl_results().

update_review_date()

Updates the last_review_date column on the REPL_ITEM_LOC table to reflect the fact that item-location combinations have just been evaluated.

PL/SQL Stored Procedure Calls

get_wh_current_avail()

Gets the available quantity of a given item at a given warehouse. This function is a wrapper for the ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL stored PL/SQL procedure.

next_transfer_number()

Gets the next transfer number in the Oracle stored sequence for creating new transfer headers. This function is a wrapper for the NEXT_TRANSFER_NUMBER stored procedure.

Domain Validation

Domain validation is done in the ociroq.c batch program.

Support Functions

copy_repl_to_store()

Copies a record from the structure holding rows from the driving cursor into a structure holding item-location information for ROQ calculation, distribution, and transfer placement.

reset_store_struct()

Resets summary variables in a store information structure to prepare it for the next set of line items.

reset_alt_item_struct()

Resets summary variables in an alternate item structure to prepare it for the next set of alternates.

Array Sizing

size_repl_info_struct()

Allocates memory to the structure used to buffer fetches from the driving cursor.

size_store_struct()

Allocates memory to the structure used to hold item-location level information.

size_alt_item_struct()

Allocates memory to the structure used to hold information about alternate items (either simple packs or substitute items).

size_tsfhead_struct()

Allocates memory to the structure used to buffer inserts to the Transfer Header table.

size_tsfdetail_struct()

Allocates memory to structures used to buffer inserts and updates to the Transfer Detail table.

size_item_loc_struct()

Allocates memory to structures used to buffer updates of the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH).

size_repl_results_struct()

Allocates memory to the structure used to buffer inserts to the Replenishment Results table.

Database Interaction

Tables Selected From:

- RPL_NET_INVENTORY_TMP
- ITEM_SUPP_COUNTRY
- PACKHEAD
- PACKITEM
- PACKSTORE_HIST
- PERIOD
- RAG_SKUS_ST_HIST
- REPL_DAY
- REPL_ITEM_LOC
- STORE
- SUB_ITEMS_HEAD
- SUB_ITEMS_DETAIL
- SYSTEM_OPTIONS
- TSFDETAIL
- TSFHEAD

- WH
- WIN_STORE_HIST

Tables Inserted To:

- REPL_RESULTS
- TSFDETAIL
- TSFHEAD

Tables Updated:

PACKWH

RAG_SKUS_ST

RAG_SKUS_WH

REPL_ITEM_LOC

TSFDETAIL

WIN_STORE

WIN_WH

I/O Specification

N/A

Technical Issues

N/A

Daily Stock Ledger Processing [saldly]

Design Overview

This program is responsible for performing the daily summarization processing in the stock ledger in which transaction-level records are fetched from the transaction-level staging table and summed to the subclass/location/day level. Once the records are summarized, they are written to the DAILY_DATA table.

First, the program reads in the calendar type option (regular or 454), then gets last day of the month for the previous and current month. To call this program the end of day process for the stock ledger would not be completely correct, however, because a day does not really "close" in the stock ledger until the month closes. Each time that the Daily Stock Ledger Processing program runs, all transaction-level data is processed, whether it is for the current date, a date since the last month closing or even a date prior to the last month closing. For transactions occurring on the current date or since the last month close, they are processed by simply summarizing the date and updating the current information on DAILY_DATA for the date of the transaction. However, if a transaction occurred prior to the last month that was closed (i.e. the transaction was dated 3/15 and the last end of month date was 3/20), then that transaction will be dated with the current date and summarized with the current date's records. Also, in this last case, a warning message will be written to the batch log that alerts the user to the problem. The message the users will receive is "*ALERT* Transactions have been found for previous months."

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
IF_TRAN_DATA	No	Yes	No	No	No
DAILY_DATA	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 3 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Threaded by Dept
V_restart_dept

Restart Recovery

```
   SELECT /*+ index(if_tran_data if_tran_data_i2)*/ dept,
                 class,
                 subclass,
                 store,
                 wh,
                 DECODE(SIGN(tran_date -
   TO_DATE(:os_last_eom_date_for_eow,
                                              'YYYYMMDD')),
                  -1, :os_fdom_date,
                  0,  :os_fdom_date,
                  TO_CHAR(tran_date, 'YYYYMMDD')),
                  SUM(DECODE(tran_code,1,NVL(units,0),0)),
                  SUM(DECODE(tran_code,1,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,1,NVL(total_retail,0),0)),

   SUM(DECODE(:oi_stkldgr_vat_incl_retl_ind+:oi_vat_ind,2,0,
                           DECODE(tran_code,1,NVL(total_retail,0),0))+
                           DECODE(tran_code,2,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,4,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,4,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,11,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,12,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,13,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,14,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,15,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,16,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,20,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,20,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,22,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,22,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,24,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,24,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,26,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,30,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,30,NVL(total_retail,0),0)),
                  SUM(DECODE(tran_code,32,NVL(total_cost,0),0)),
                  SUM(DECODE(tran_code,32,NVL(total_retail,0),0)),
```

```
                SUM(DECODE(tran_code,34,NVL(total_cost,0),0)),
                SUM(DECODE(tran_code,34,NVL(total_retail,0),0)),
                SUM(DECODE(tran_code,36,NVL(total_cost,0),0)),
                SUM(DECODE(tran_code,36,NVL(total_retail,0),0)),
                SUM(DECODE(tran_code,60,NVL(total_retail,0),0)),
                SUM(DECODE(tran_code,70,NVL(total_cost,0),0)),
                SUM(DECODE(tran_code,80,NVL(total_retail,0),0)),
                SUM(DECODE(tran_code,81,NVL(total_retail,0),0))
   FROM if_tran_data,
                v_restart_dept rv
WHERE rv.driver_value     = if_tran_data.dept
       AND rv.driver_name      = :os_restart_driver_name
       AND rv.num_threads      = :oi_restart_num_threads
       AND rv.thread_val       = :oi_restart_thread_val
       AND (if_tran_data.dept  >
NVL(:os_restart_dept,if_tran_data.dept - 1) OR
                    (if_tran_data.dept = :os_restart_dept AND
                       (if_tran_data.class > :os_restart_class OR
                          (if_tran_data.class = :os_restart_class
AND
                             (if_tran_data.subclass >
:os_restart_subclass)))))
  GROUP BY  dept,
                         class,
                         subclass,
                         store,
                         wh,
                         DECODE(SIGN(tran_date –
                         TO_DATE(:os_last_eom_date_for_eow,
'YYYYMMDD')),
                         -1, :os_fdom_date,
                         0,  :os_fdom_date,
                         TO_CHAR(tran_date, 'YYYYMMDD'))
  ORDER BY  dept,
                         class,
                         subclass;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A