# Retek® Merchandising System™ 11.0.4

# 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<br>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<br>Retek on the Mall<br>950 Nicollet Mall<br>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 – Item zone price

## Overview

The Item_zone_price table was removed and replaced with a view called Item_zone_price.

A view is a named query that is saved on the database. To users and code, it looks like a table. But instead of physically storing data, it simply displays data that is physically stored elsewhere. The Item_zone_price view does not allow inserts, or updates. It is a read-only view. However, selects can be done on views, and code can reference the columns in views in %TYPE statements.

User processes remain the same; the user creates items and uses the item retail window to create base retail. These base retails are communicated to RPM, but no additional processing in RMS occurs.

The new Item_zone_price view is based on the Item_loc table. Additional columns are added to the Item_loc table to support the changes. When items are actually ranged, a call to RPM is made to get the appropriate retail for the item/location. This retail is written to the Item_loc table. The new Item_zone_price view selects this value from the Item_loc table.

Non-ranged item/location retails (that is, retails for item/location relationships not found on the ITEM_LOC table) are retrieved from RPM via a new package call.

## Batch summary

The following batch programs were changed:

- STOREADD.PC/WHADD.PC
- STKUPLD.PC
- STKVAR.PC
- POSUPLD.PC
- TCKDNLD.PC
- Std_len.h

| Batch design | Details | Batch dependencies Run before / after |
|---|---|---|
| WHADD.PC | Reads new warehouses, virtual warehouses, and/or internal finishers from the WH_ADD table. Records are inserted into the PRICE_ZONE and PRICE_ZONE_GROUP_STORE for each retrieved record. | Run daily as needed.<br>Run before SLOCRBLD.PC. |
| STOREADD.PC | Creates new stores in RMS. Whenever a new store is created in the online dialog, the store is saved to a temporary table. STOREADD.PC processes the newly-added store from the staging table and creates a new store in RMS. | Run as needed.<br>Run before SLOCRBLD.PC. |

| Batch design | Details | Batch dependencies Run before / after |
|---|---|---|
| STKUPLD.PC | Uploads actual count data uploaded from store or warehouse. The uploaded file INV_BAL sent by the warehouse management system is first translated by the LIFSTKUP.PC module before STKUPLD.PC inputs the file for processing.<br>In a multi-channel environment, STKUPLD.PC calls the distribution module to distribute physical warehouse counts to virtual warehouses. | Run daily in Phase 3 of RMS' batch schedule.<br>Run after STKUPD.PC.<br>Run after RMS upload of count data from retailer location.<br>Run after LIFSTKUP.PC. |
| STKVAR.PC | Processes stock-on-hand adjustments applied through the online variance review form. | Run daily in Phase 3 of RMS' batch schedule.<br>Run after STKUPLD.PC. |
| POSUPLD.PC | Uploads customer created POSU file from customer's point-of-sale system, processes sales and return data, and posts sales transactions to the TRAN_DATA (sales) and ITEM_LOC_HIST (item-location history) tables. | Run daily in Phase 2 of RMS' batch schedule.'<br>Run multiple times a day in a trickle-polling environment.<br>Run after SAEXPRMS.PC when Retek Sales Audit is used. |
| TCKTDNLD.PC | The tickets and labels batch program outputs an interface file for an external ticket printing system. The program runs to create an output file contain all information to be printed on a ticket or label for a particular item and location. It also includes the requested ticket type. . | Run daily during any phase of the batch schedule. |

# Batch design details

## Like store [LIKESTORE]

### Design Overview

When a new store is created in RMS there is an option to specify a like store. When storeadd batch is run it sets the store open date and close date of all the like stores far in the future, so that those records will be picked up in the likestore batch. Likestore batch creates item location relationships for all the items in the existing store with new store. The likestore batch will process like stores and sets the store open and close dates back to original date in the post process. User can specify whether to copy the Replenishment information, delivery schedules and activity schedules from the existing store, which will be copied in the likestore post process. So it is necessary to run the storeadd, likestore and likestore post in the same order to successfully add all the stores in to RMS.

Likestore batch uses multi-threading by department along with array processing to copy item expense information. It also utilizes array processing to fetch all items associated to the likestore and their attributes. The array of these items and their attributes is then looped through, with the NEW_ITEM_LOC_SQL.NEW_ITEM_LOC package function being called for each item to create the new relationship.

### Scheduling Constraints

| | |
|---|---|
| Processing Cycle: | Ad Hoc Phase |
| Scheduling Diagram: | N/A |
| Pre-Processing: | storeadd.pc |
| Post-Processing: | prepost(likestore post) |
| Threading Scheme: | Table based processing, multithreading on Department. |

### Restart/Recovery

The logical unit of work is store, item, pack indicator. The following two cursors will keep track of store, item, and pack indicator in the restart book mark. The c_add_store cursor restart the program based on store and c_get_items will restart the program based on item, pack indicator.

```
EXEC SQL DECLARE c_add_store CURSOR for
       SELECT sa.store,
              sa.like_store,
              ROWIDTOCHAR(st.rowid)
         FROM store_add sa,
              store st
        WHERE sa.store = st.store
          AND st.store_open_date = sa.store_open_date + 500000
          AND st.store_close_date = sa.store_open_date + 500000
          AND (sa.store > NVL(:ps_restart_store,-999) OR
               sa.store = :ps_restart_store)
     ORDER BY sa.store;
```

```
EXEC SQL DECLARE c_get_items CURSOR FOR
      SELECT il.item,
             im.item_desc,
             im.diff_1,
             im.diff_2,
             im.diff_3,
             im.diff_4,
             il.loc_type,
             il.daily_waste_pct,
             iscl.unit_cost,
             il.unit_retail,
             il.multi_units,
             il.multi_unit_retail,
             il.multi_selling_uom,
             il.selling_unit_retail,
             il.selling_uom,
             il.status,
             il.taxable_ind,
             il.ti,
             il.hi,
             il.store_ord_mult,
             il.meas_of_each,
             il.meas_of_price,
             il.uom_of_price,
             il.primary_variant,
             il.primary_supp,
             il.primary_cntry,
             il.local_item_desc,
             il.local_short_desc,
             il.primary_cost_pack,
             il.receive_as_type,
             im.item_parent,
             im.item_grandparent,
             im.dept,
             im.class,
             im.subclass,
```

```
            im.status,
            cl.class_vat_ind,
            im.short_desc,
            im.item_level,
            im.tran_level,
            im.retail_zone_group_id,
            pzgs.zone_id,
            im.sellable_ind,
            im.orderable_ind,
            im.pack_ind,
            im.pack_type,
            im.waste_type,
            st.lang,
            il.source_method,
            il.source_wh
       FROM v_restart_dept vrd,
            store st,
            price_zone_group_store pzgs,
            item_master im,
            class cl,
            item_loc il,
            item_supp_country_loc iscl
      WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)
        AND vrd.thread_val = TO_NUMBER(:ps_thread_val)
        AND vrd.driver_value = im.dept
        AND st.store = TO_NUMBER(:is_like_store)
        AND st.store = il.loc
        AND ((im.pack_ind = NVL(:ps_restart_pack_ind, 'N') AND
im.item > NVL(:ps_restart_item, ' '))
            OR (im.pack_ind > NVL(:ps_restart_pack_ind, 'N') AND
im.item > ' '))
        AND il.item = im.item
        AND im.dept = cl.dept
        AND im.class = cl.class
        AND pzgs.store(+) = TO_NUMBER(:is_store)
    AND im.retail_zone_group_id = pzgs.zone_group_id(+)
        AND il.CLEAR_IND = 'N'
        AND il.ITEM = iscl.ITEM(+)
```

```
            AND il.LOC = iscl.LOC(+)
            AND il.primary_supp = iscl.supplier(+)
            AND il.primary_cntry = iscl.origin_country_id(+)
      ORDER BY im.pack_ind asc,
          il.item;
```

### Program Flow

N/A

### Function Level Description

init()

- Initialize the restart variables
- Get system variables (ELC indicator, VAT indicator, std_av_ind and rpm_ind)

process()

- Select values from the STORE_ADD table for stores that the storeadd.pc program has already processed, as evidenced by the store open date far in the future.
- Loop through all the likestore records and call Copy_Store_Items function for each like store record.

copy_Store_Items()

- If the ELC indicator is “Y”, the item expenses tables are updated with the details of expenses involved in moving the items from one location to other locations. This is done using array possessing.
- C_get_items cursor will fetch all the records for the item location combination of the old store and create all the item location relationships with new store by calling the NEW_ITEM_LOC_SQL.NEW_ITEM_LOC() package function.

size_exp_head()

- Allocates memory to the exp_head structure

size_exp_head_seq()

- Allocates memory to the exp_head_seq structure

size_exp_insert()

- Allocates memory to the exp_insert structure

size_new_itemloc()

- Allocates memory to the new_itemloc structure

free_exp_head()

- Releases the memory allocated in size_exp_head function.

free_exp_head_seq()

- Releases the memory allocated in size_exp_head_seq function.

free_exp_insert()

- Releases the memory allocated in size_exp_insert function.

free_new_itemloc()

- Releases the memory allocated in size_new_itemloc function.

final()

- This function stops restart recovery.

### I/O Specification

N/A

### Technical Issues

N/A

### Processing Cursors

```
/* Any changes made to c_count_item_exp_head must be replicated in
c_item_exp_head */

/* The count returned in c_count_item_exp_head determines the number
of records    */

/* to be processed by c_item_exp_head. The 'FROM' and 'WHERE'
clauses must match.  */

   EXEC SQL DECLARE c_count_item_exp_head CURSOR FOR

      SELECT count(ieh.item)

        FROM v_restart_dept vrd,

                    cost_zone_group czg,

                    item_master im,

                    item_exp_head ieh

       WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)

         AND vrd.thread_val = TO_NUMBER(:ps_thread_val)

         AND vrd.driver_value = im.dept

         AND czg.cost_level = 'L'

         AND czg.zone_group_id = im.cost_zone_group_id

         AND im.item = ieh.item

         AND (:ps_restart_item = '-999' OR :ps_restart_item is NULL)

         AND ieh.zone_group_id = czg.zone_group_id

         AND ieh.zone_id = TO_NUMBER(:is_like_store)

         AND ieh.item_exp_type = 'Z';
```

```
/* Any changes made to c_item_exp_head must be replicated in
c_count_item_exp_head */

/* The count returned in c_count_item_exp_head determines the number
of records    */

/* to be processed by c_item_exp_head. The 'FROM' and 'WHERE'
clauses must match.  */

   EXEC SQL DECLARE c_item_exp_head CURSOR FOR

      SELECT ieh.item,

             ieh.supplier,

             NVL(ieh.item_exp_seq,0),

             ROWIDTOCHAR(ieh.rowid)

        FROM v_restart_dept vrd,

             cost_zone_group czg,

             item_master im,

             item_exp_head ieh

       WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)

         AND vrd.thread_val = TO_NUMBER(:ps_thread_val)

         AND vrd.driver_value = im.dept

         AND czg.cost_level = 'L'

         AND czg.zone_group_id = im.cost_zone_group_id

         AND im.item = ieh.item

         AND (:ps_restart_item = '-999' OR :ps_restart_item is NULL)

         AND ieh.zone_group_id = czg.zone_group_id

         AND ieh.zone_id = TO_NUMBER(:is_like_store)

         AND ieh.item_exp_type = 'Z'

    ORDER BY ieh.item, ieh.supplier, ieh.item_exp_seq desc;
```

# POS upload [POSUPLD]

### Design Overview

The purpose of this batch module is to process sales and return details from an external point of sale system. The sales/return transactions will be validated against Retek item/store relations to ensure the sale is valid, but this validation process can be eliminated if the sales being passed in have already been screened by sales auditing. The following common functions will be performed on each sales/return record read from the input file:

- read sales/return transaction record
- lock associated record in RMS
- validate item sale
- check if VAT maintenance is required, if so determine the VAT amount for the sale
- write all financial transactions for the sale and any relevant markdowns to the stock ledger.
- post item/location/week sales to the relevant sales history tables
- if a late posting occurs in a previous week (i.e. not in the current week), if the item for which the late posting occurred is forecastable, the last_hist_export_date on the item_loc_soh table has to be updated to the end of week date previous to the week of the late posting. This will result in the sales download interface programs extracting the week(s) for which the late transactions were posted to maintain accurate sales information in the external forecasting system.

### Stored Procedures / Shared Modules (Maintainability)

validate_all_numeric: intrface library function.

validate_all_numeric_signed: intrface library function.

valid_date: intrface library function.

PM_API_SQL. GET_RPM_SYSTEM_OPTIONS: called from init(), returns complex_promo_allowed_ind to set pi_multi_prom_ind

CAL_TO_CAL_LDOM database procedure called from get_eow_eom_date() function

CAL_TO_454_LDOM database procedure called from get_eow_eom_date() function

VAT_SQL.GET_VAT_RATE: called from pack_check(), fill_packitem_array() returns the composite vat rate for a packitem.

CURRENCY_SQL.CONVERT: returns the converted monetary amount from

Currency to currency.

NEW_ITEM_LOC: called from item_check(), item_check_orderable(), pack_check_orderable() and pack_check(), creates a new item if one doesn't already exist for the item/location passed in.

UPDATE_SNAPSHOT_SQL.EXECUTE: called from update_snapshot(), updates the stake_sku_loc and edi_daily_sales tables for late transactions. If the item is a return, edi_daily_sales will not be updated.

NEXT_ORDER_NO: called from consignment_data(), returns the next available generated order number.

STKLDGR_SQL.TRAN_DATA_INSERT: called from consignment_data(), performs tran_data inserts (tran_type 20) for a consignment transaction.

DATES_SQL.GET_EOW_DATE: called from get_eow_eom_date(), returns eow and eom dates.

UOM_SQL.CONVERT: called from validate_THEAD(), converts selling uom to standard uom.

SUPP_ATTRIB_SQL.GET_SUP_PRIMARY_ADDR: called from invc_data(), returns primary supplier address.

INVC_SQL.NEXT_INVC_ID: called from invc_data(), returns invoice_id

PRICING_ATTRIB_SQL.GET_BASE_ZONE_RETAIL(), called from get_loc_item_retail(), returns base zone retail from RPM.

Posupld and VAT:

There are three different data sources in POSUPLD.

- the input file
- RMS stock ledger tables (tran_data in this context)
- RMS base tables (other that stock ledger)

Each of these data sources can be vat inclusive or vat exclusive.

There are five different system variables that are used to determine whether of not the different inputs are vat inclusive or vat exclusive.

- system_options.vat_ind (assume Y for this document)
- system_options.class_level_vat_ind
- system_options.stkldgr_vat_incl_retl_ind
- class.class_vat_ind
- store.vat_include_ind (this is retrieved from the table when RESA is on and read from the input file when RESA is off)

Given the three different data source and all combinations of vat inclusive or vat exclusive, we are left with the 8 potential combinations of inputs to POSUPLD.

| Possible POSUPLD inputs | | | |
|---|---|---|---|
| **SCENARIO** | **FILE** | **RMS** | **STOCK LEDGER** |
| 1 | Y | Y | Y |
| 2 | Y | Y | N |
| 3* | Y | N | Y |
| 4* | Y | N | N |
| 5 | N | Y | Y |
| 6 | N | Y | N |
| 7 | N | N | Y |
| 8 | N | N | N |

* Scenarios 3 and 4 are not possible – the file will never have vat when RMS does not.

| The combinations of system variables and the resulting scenarios | | | | |
|---|---|---|---|---|
| **System_options Class_level_vat_ind** | **System_options Stkldgr vat ind** | **Class Class_vat_ind** | **Store Vat_include_ind** | **Resulting Scenario** |
| Y | Y | Y | Y - Ignored | 1 |
| Y | Y | Y | N - Ignored | 1 |
| Y | Y | N | Y - Ignored | 7 |
| Y | Y | N | N - Ignored | 7 |
| | | | | |
| Y | N | Y | Y - Ignored | 2 |
| Y | N | Y | N - Ignored | 2 |
| Y | N | N | Y - Ignored | 8 |
| Y | N | N | N - Ignored | 8 |
| | | | | |
| N | Y | Y – Ignored | Y | 1 |
| N | Y | Y – Ignored | N | 5 |
| N | Y | N – Ignored | Y | 1 |
| N | Y | N – Ignored | N | 5 |
| | | | | |
| N | N | Y – Ignored | Y | 2 |
| N | N | Y – Ignored | N | 6 |
| N | N | N – Ignored | Y | 2 |
| N | N | N – Ignored | N | 6 |

POSUPLD table writes

Scenario 1:

tran code 1 from file retail.

tran code 2 from file retail with vat removed.

retail from file is compared directly with price_hist for off retail check.

Scenario 2:

tran code 1 from file retail with vat removed.

tran code 2 not written.

retail from file is compared directly with price_hist for off retail check.

Scenario 5:

tran code 1 from file retail with vat added.

tran code 2 from file retail.

retail from file has vat added for compare with price_hist for off retail check.

Scenario 6:

tran code 1 from file retail.

tran code 2 not written.

retail from file has vat added for compare with price_hist for off retail check.

Scenario 7:

tran code 1 from file retail with vat added.

tran code 2 from file retail.

retail from file is compared directly with price_hist for off retail check.

Scenario 8:

tran code 1 from file retail.

tran code 2 not written.

retail from file is compared directly with price_hist for off retail check.

### Function Level Description

main()

standard Retek main function that calls init(), process(), and final()

init()

initialize restart recovery

open input file (posupld)

- file should be specified as input parameter to program
  fetch system variables, including the SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND.
  fetch pi_multi_prom_ind from RPM interface
  retrieve all valid promotion types and uom class types
  fetch uom class types for look up during THEAD processing
  declare memory required for all arrays setup for array processing
  declare final output filename (used in restart_write_file logic)
  open reject file ( as a temporary file for restart )
- file should be specified as input parameter to program
  open lock reject file ( as a temporary file for restart )

file should be specified as input parameter to program
call restart_file_init logic
assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), lock reject file counters (pl_lock_cnt, pl_lock_dtl_cnt), store, transaction_date
if fresh start (l_file_start = 0)
read file header record (get_record)
write FHEAD to lock reject file
if (record type <> 'FHEAD') Fatal Error
validate file type = 'POSU'
else fseek to l_file_start location
validate location and date are valid
set restart variables to ones from restart image
file_process()

This function will perform the primary processing for transaction records retrieved from the input file. It will first perform validation on the THEAD record that was fetched. If the transaction was found to be invalid, a record will be written to the reject file, a non-fatal error will be returned, and the next transaction will be fetched.

Next, the unit retail from price_hist will be fetched by calling the get_unit_retail() function. The retail retrieved from this function will be compared with the actual retail sent in from the input file to determine any discrepancies in sale amounts.

Fetch all of the TDETL records that exist for the transaction currently being processed until a TTAIL record is encountered. Perform validation on the transaction detail records. If a detail record is found to be invalid, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. If a valid promotion type (code for mix & match, threshold promotions, etc.) was included in the detail record and it is not an employee disc record, write a record to the daily_sales_discount table. If it is an employee discount record write an employee discount record to tran_data. Finally, accumulate the discount amounts for all transaction detail records for the current transaction, unless the record was an employee discount. Next, establish any vendor funding of promotions. This information is expressed as a percentage of the allowed discount and is retrieved by querying the rpm_promo_xxx tables for the promotion_id and component_id. If the promotion type is 9999 (i.e., all promotion types), call get_deal_contribs to append to pr_deals_contribs arrays zero or more lines of deal and vendor contribution information for the current item

Call the item_process() function to perform item specific processing. Once all records have been processed, write FTAIL record to lock reject file and call posting_and_restart to commit the final records processed since the last commit and exit the function.

item_process()

Check to see if any validation failed for the item before this function was called. If a lock error was found, call write_lock_rej() then return. If an other error was found, call write_rej() and process_detail_error() then return.

Set the item sales type for the current transaction. Valid sales types are 'R'egular sales, 'C'learance sales, and 'P'romotional sales. These will be used when populating the sales types for the item-location history tables. If an item is both on promotion and clearance, and the promotion price is less than the clearance price, than the transaction will be written as a promotion transaction, otherwise as a clearance transaction.

If the system's VAT indicator is turned to on, VAT processing will be performed. The function vat_calc() will retrieve the vat rate and vat code for the current item-location. The total sales including and excluding VAT will be calculated for use in writing transaction data records. If any VAT errors occur, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. A record will be written to vat_history for the item, location, transaction date.

Calculate the item sales totals (i.e. total retail sold, total quantity sold, total cost sold, etc.). If VAT is turned on in the system, calculate exclusive and inclusive VAT sales totals.

Calculate any promotional markdowns that may exist by calling the calc_prom_totals() function. The markdown information calculated here will be used when writing tran_data (tran_type 15) records for promotional markdowns.

Calculate the over/under amount the item was sold at compared to its price_hist record. (The complex_promo_allowed_ind indicator is retrieved from RPM by calling PM_API_SQL.GET_SYSTEM_OPTIONS.) Since we do not create price_hist records of type 9 (promotional retail change) when the complex_promo_allowed_ind = 'Y', we do not know what the promotional retail for this item is. Therefore, we will take the total sales reported from the header record plus the total of sales discounts reported in the TDETL records, divided by the total sales quantity for the item to calculate its unit retail. If the complex_promo_allowed_ind = 'N', we can do a comparison of the price_hist record and the unit retail (total retail / total sales) inputted from the POS file. Any difference using either method will write to the daily_sales_discount table with a promotion type of 'in store' and tran_data (tran_type 15) If the transaction is a return, no daily_sales_discount record will be written, and tran_data records will be written as opposite of what they were sold as (i.e. if the sale was written as a markup, which would be written as a negative retail with a tran_data 15, the return would be written as a 15 with a positive retail).

If the item is a packitem and the transaction is a Sale, the process_pack() function will update the last_hist_export_date field on the item_loc_soh table to the transaction date and the item_loc_hist table will be updated with the transaction information.

If the item currently being processed is a packitem, calculate the retail markdown the item takes for being included in the pack and write a transaction data record as a promotional markdown. This markdown is calculated by comparing the retail contribution of the packitem's component item to the packitem to the component item's regular retail found on the price_hist table. The retail contribution for a component item is calculated by taking the component item's unit retail from price_hist, divided by the total retail of all component items in the packitem, and multiplying the packitem's unit retail. So if the retail contribution of a component item within packitem A is $10, and the same component item's price_hist record has a retail of $14, and there is only one packitem sold, and this component item has a quantity of one, a tran_data

Record (tran_type 15) will be written for $4 (assume no vat is used).

Write transaction data records for sales and returns. If the transaction is a sale, write a tran_data record with a transaction code of 1 with the total sales. If the system VAT indicator is on and the system_options.stkldgr_vat_incl_retl_ind is on, write a tran_data record with a transaction code of 2 for VAT exclusive sales. If the transaction is a return, write a tran_data record (tran_type 1) with negative quantities and retails for the amount of the return. If the system VAT indicator is on and the system_options.stkldgr_vat_incl_retl_ind is on, write a tran_data record (tran_type 2) and negative quantities and retails for the VAT exclusive return. Also, write a tran_data record with a transaction code of 4 for the total return. Any tran_data record that is written should be either VAT exclusive or VAT inclusive, depending on the system_options.stkldgr_vat_incl_retl_ind. If it is set to 'Y', all tran_data retails should be VAT inclusive. If it is set to 'N', all tran_data retails should be VAT exclusive. When writing tran_data records for packitems, always break them down to the packitem level, writing the retail as the packitem multiplied by the component item's price ratio. The packitem itself should never be inserted into the tran_data table.

If the transaction is late (transaction date is before the current date) and it is not a drop shipment, call update_snapshot() to update the stake_sku_loc and edi_daily_sales tables. If the transaction is current, update the edi_daily_sales table only (stake_sku_loc will be updated in a batch program later down the stream). The edi_daily_sales table should only be updated if the items supplier edi sales report frequency = 'D'.

If VAT is turned on in the system, write a record to the vat_history table to record the vat amount applied to the transaction. The VAT amount is calculated by taking the sales including VAT minus the sales excluding VAT.

Update the sales history tables for non-consignment items that are Sale transactions. Do not update for returns. Also, update stock count on the item-location table for Sales and Returns unless the item is on consignment or is drop shipped.

If the dropship indicator is set to 'Y', then the sale is drop shipped and there is no update for stock on hand. Drop shipments are used for sales at a virtual or physical location where an order is taken from a customer, but the goods are shipped directly from the vendor to the customer (not via any store or warehouse owned by the retailer). If an item is used only for drop shipments and there is no stock on hand before or after the cost price is changed, the weighted average cost is never updated when average cost accounting method is used. The average cost will be the initial cost price at the time the item is set up. Over a period of time, under average cost accounting method, the cost price used to charge these items will drift away from the actual supplier cost. See SYSTEM_OPTIONS.STD_AV_IND for further details on cost accounting method.

If an off_retail amount was identified for the item/location, call the write_off_retail_markdowns() function to write tran_data records (tran_type 15) to record the difference. If the complex_promo_allowed_ind = 'N' and the item is on promotion, or if the complex_promo_allowed_ind = 'Y' and the TDETL total discount amount is greater than zero, write a promotional markdown. Note: this will also record a tran_data record (tran_type 15) for a TDETL record that has a promotional transaction type with no promotion number in order to record the markdown.

If an employee discount TDETL record has been encountered, a tran_data record with tran_code 60 will be written.

If the item is a wastage item, a tran_data record with tran_code 13 will be written. This record is used to balance the stock ledger, it accounts for the amount of the item that was wasted in processing.

process_detail_error()

This function writes a record to the load_err table for every non-fatal error that occurs.

set_counters()

Depending on the action passed into this function, it will either set a savepoint and store the values of counters or rollback a savepoint and reset the values of certain counters back to where they were originally set. This function is called when a non-fatal error occurs in the item_process() function to rollback and changes that may have been made.

calc_item_totals()

This function will set total retail and discount values including and excluding VAT, depending upon the store.vat_include_ind, system_options.vat_ind, complex_promo_allowed_ind, and the system_options.stkldgr_vat_incl_retl_ind.

calc_prom_totals()

This function will set promotional markdown values including and excluding VAT, depending upon the complex_promo_allowed_ind and the system_options.stkldgr_vat_incl_retl_ind. If the multi_prom_ind is on, the promotional markdown is the sum of the TDETL discount amounts. If the multi_prom_ind is off, the promotional markdown is the difference between the price_hist record with a tran_code of 0,4,8,11 and the price_hist record with a tran_code of 9 multiplied by the total sales quantity. Also, the tran_data old and new retail fields are only written if the multi_prom_ind is off.

Where vendor funding is present, compute the vendor contributions of the promotional discount in local and deal currencies, write local currency vendor funding invoices with tran_code = 6 to tran_data, and write deal currency vendor funding details to the deal_actuals_item_loc in deal currency. Call calc_vendor_funding (passing in the ex-vat total promotional mark down), to compute each vendor contribution (if any) in local currency for writing to the stock ledger and in deal currency for writing to deal_actuals_item_loc.

calc_vendor_funding()

This function accepts an ex-vat promotional discount amount and splits it by percentage for each of the vendors and deals in the list in both local and deal currency. A call is made to de-encapsulated currency conversion module convert(…), for efficiency in place of calling the PL/SQL equivalent function

process_sales_and_returns()

If a non-pack concession item is being processed, concession_data() is called to write accounts receivable data to the concession_data table. If the item is on consignment and not a packitem, the consignment_data() function will be called to perform consignment processing. The function write_tran_data() will be called to write a tran_data record with a tran_type 1 (always written), a tran_type 2 (if the system_options. vat_ind = Y and system_options.stkldgr_vat_incl_retl_ind = Y), a tran_type 3 (for non-inventory/non-deposit container item sales and returns), and a tran_type 4 (if the transaction was a return). If the transaction is a return, any tran_data records with tran_types of 1 and 2 will be written with negative retails. Also the update_price_hist() function will be called to update the most recent price_hist record.

If the retail price has changed since the sale occurred, process_reversal_records() function is called to write a tran_data record to reverse the price change for the items sold. Either a cancel markup or cancel markdown code is written. The retail amount to be cancelled is the difference between the retail sale price and current retail price multiplied by the total number of items sold or returned.

process_reversal_records()

If the retail price has changed since the sale occured, an unjustified loss on the stock ledger vs. the store tables is created. To correct this, a record needs to be written to tran_data reversing the price change for the items sold. This will use either a cancel markdown or markdown code. The quantity and retail will be the negative of the actual qty and retail, since a reversal is being processed.

validate_FHEAD()

Do standard string validations on input fields. This includes null padding fields, checking that numeric fields are all numeric, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the store location exists.

If the sales audit indicator is on currency and vat information will be provided in the file that has already been validated.

get_eow_eom_date()

This function returns the eow_date and eom_date for the current tran_date. For the eom_date, the appropriate base function is called to return the correct date for Gregorian or 454 calendar.

validate_THEAD()

Do standard string validations on input fields. This includes null padding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the reference item exists.

If a reference item is passed in from the input file, retrieve the item for the reference item. Once the item is an item, retrieve the transaction and item level values, pack indicator, department, class, subclass, waste_type, waste_pct. Once this information is retrieved, check that the item/location relationship exists for the appropriate item type and call check_item_lock() and/or check_pack_lock depending on item type to lock this item's ITEM_LOC record.

If the sale audit indicator is 'Y' on system_options, the item will be a item and the dept, class, subclass, item level, transaction level and pack_ind will be included in the file. The UOM is assumed to already by have been converted to the standard UOM by Sales Audit.

If the Sales Audit indicator is 'N' on system_options, the UOM at which the item was sold will be compared with the items standard UOM value. If they are different, the quantity will be converted to the standard UOM amount. The ratio of the difference will also be computed and saved for use by validate_TDETL().

If an item is a wastage item set the wastage qty. The qty sent in the file shows the weight of the item sold. The wastage qty is the qty that was processed to come up with the qty sold. So if .99 of an item was sold, and item wastage percent is 10. The wastage qty is .99 / (1-.10) = 1.1 The wastage qty will be used through out the program except when writing tran_data records(see write_wastage_markdown) and daily_sales_discount records which will uses the processed qty from the file.

Class-level vat functionality is addressed here. The c_ get_class_vat cursor is fetched into the pi_vat_store_include_ind variable if vat is tracked at the class level in RMS (SYSTEM_OPTIONS.VAT_IND = 'Y' and SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND = 'Y'). The vat inclusion indicator passed in the input file is overwritten with the vat indicator for the class passed in the THEAD record of the input file.

If catchweight_ind is Y, call valid_all_numeric() to check that the actualweight_qty is all numeric, else call all_blank() to validate that it is blank. If the catchweight_ind is Y, convert actualweight_qty to 4 places of decimals reflecting the correct sign. Validate that the subtrans_type is either A, D or null.

If the item is part of an item transformation (pi_item_xform is TRUE), call get_item_xform_detail() to populate the pr_xform_items structure with the associated orderables, and return the total yield for all rows retrieved and also the calculated unit cost of the sellable item based on its component orderable items. This value overwrites pd_unit_cost_loc, which for standard items is populated by function item_check(…). If the returned sum of all retrieved pr_xform_items.as_yield does not equal 1, reject the record

get_ref_item()

This function is being called by the validate_THEAD function if the item_type is 'REF'. This function will return the item_parent of a specific item.

get_item_info()

This function gets item data from item_master and deps for an item_id passed in.

validate_TDETL()

This function will perform validation on the TDETL records passed into the program. The standard string validation on these fields includes null padding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true.

The quantity is multiplied by the UOM ratio determined in validate_THEAD().

If a promotional transaction type is passed in, verify it is valid. If a promotional transaction type is passed in, but it is not valid, return non-fatal error then set non-fatal error flag to true.

If the item is a wastage item set the tdetl wastage qty. This is done the same way as setting the THEAD wastage qty.

If the promotion type is 9999 (i.e., all promotion types), verify that the promotion and promotion component are all numeric. If the promotion type is not 9999 (i.e., non-promotional), then verify that the promotion and promotion component are blank. If the promotion type is 9999, call validate_prom_info.

uom_convert()

This function is called by validate_THEAD to convert the selling UOM to the standard UOM.

validate_prom_info()

This function looks up the promotion in the rpm_promo table and the promotion_component in the rpm_promo_comp table. If either row does not exist, an error is reported and the function returns non-fatal. At the same time, any promotional consignment rate is retrieved and returned to the calling function

get_deal_contribs()

This function re-sizes the arrays to receive the list of vendor funding details if necessary and then appends the arrays with data, leaving a contribution count of zero or more in pl_deal_contribs_ctr. The function also fetches records from the deal_head, deal_comp_prom and deal_actuals_forecast tables to variables that will be used by the batch program in later processing. This function can process multiple promotions per deal component.

item_store_cursors()

This function checks the item_loc for the item / store combination. It is called by the item_check() and item_check_orderable().

new_item_loc()

This function creates a new store item relationship for items. It is called by item_check.

item_check()

This function verifies the fashion item/location relationship exists. It is only called when the item being processed is a fashion item. If the item/location relationship does not exist, it is created and a record is written to the Invalid item/location output file.

item_check_orderable()

This function gets the item information of a transform orderable item. If orderable pack indicator of the item is 'Y', call pack_check_orderable(). Else, it calls on the item_store_cursors function to check if location exists for the item. If none, it calls on procedure NEW_ITEM_LOC to create new store item relationship for the items.

pack_check_orderable()

This function calls on procedure NEW_ITEM_LOC to create new store item relationship for the items.

get_vat_rate()

This function calls on package VAT_SQL.GET_VAT_RATE and returns the vat rate of a specific item. This is being called by pack_check() and fill_packitem_array().

pack_check()

This function verifies the pack item/location relationship exists and retrieves the component items for the packitem. It is only called when the item being processed is a packitem. The component item, system indicator, department, class, subclass, cost, retail, price_hist retail, and component item quantity are fetched. If the packitem/location relationship does not exist, it is created for the Packitem and all of its components and a record is written to the Invalid item/location output file for the packitem.

The component items price ratios are also calculated. This indicates the retail contribution the component item gives towards the unit retail of the packitem. This ratio is calculated by taking the price_hist unit retail of the component divided by the total price_hist retail of all the component items for the packitem. Below is an example of how this ratio is calculated:

| | Unit Retail | Qty | Retail | Calculation | Ratio |
|---|---|---|---|---|---|
| packitem A | $60 | | | | |
| item 1 | $15 | 2 | $30 | ($30/$90) * $60 | .3333 |
| item 2 | $10 | 6 | $60 | ($60/$90) * $60 | .6667 |

item_supplier()

This function populates item information for the given item's supplier. This is called from the item_process() function, if the item_type is not = 'PACK' item.

get_unit_retail()

This function retrieves the current unit retail and the retail price of the item at the time of the sale from price_hist for the item/location being processed. If a tran_code of 8 is returned, the item is on clearance. The function will always return retail that are vat inclusive. If retail is stored in RMS with out vat (system_options.class_level_vat_ind = Y and class.class_vat_ind = Y) it will add vat to the retails.

get_base_price()

This function gets the unit_retail from price_hist (tran_type 0).

daily_sales_insert_update()

This function is called by write_off_retail, write_in_store, and process_daily_sales_discount. It performs the actual insert or fills a update array for the daily_sales_discount table.

check_daily_exists()

This function will check the daily_sales_discount for the existence of a record matching the input parameters.

process_daily_sales_discount()

This function will insert/update a record to daily_sales_discount for each TDETL record that has a promotional transaction type except employee discounts. Employee discount records are not written to daily_sales_discount, they are put on tran_data with a tran_code of 60. When employee discount records are encountered, values are set for the tran_data insert and the discount amount is added to the total sales value. This is done so employee discounts do figure into the promotional and in store calculations. When the multi_prom_ind is on all promotion types except employee discount will be ignored.

write_in_store()

This function will handle record sent in as 'is store' discounts amounts. It will call check_daily_exists and daily_sales_insert_update.

write_off_retail()

This function will calculate discrepancies between the amount sold for an item, and the amount it should have sold for (price_hist record). If these amounts are not in balance, a record is written to the daily_sales_discount table with a prom_type of 'in store' for reporting.

remove_stklgdr_vat()

This function will remove vat from 3 fields after the daily_sales_discount processing is complete. The variables od_off_retail_amt, od_new_retail, and od_old_retail are stripped of vat by calling vat_convert if the stock ledger does not contain vat.

write_off_retail_markdowns()

The write_tran_data() function will be called to write the off_retail markdown unless the item is on consignment or the off_retail amount is zero.

write_promotional_markdowns()

The write_tran_data() function will be called to write the promotional markdown unless the item multi_prom_ind is off and the transaction is a return, the item is on consignment, or the promotional markdown amount is zero. The tran_data new and old retails are only written if the multi_prom_ind is off. If any vendor funding rows are in the pr_deal_contribs arrays, call function write_vendor_tran_data to write the vat-inclusive vendor funding information to tran_data, and call function write_vendor_deal_actuals to write the vat-exclusive vendor funding information to deal_actuals_item_loc

write_vendor_tran_data()

This function writes a deal contribution record to the stock ledger for each of the vendor contributions stored in the deal contributions arrays by calling write_tran_data for the TRAN_CODE_VENDOR_FUNDING tran_type (type 6).

write_wastage_markdown()

This function will call to the write_tran_data() function if the item is a wastage item. A wastage item is an item that loses some of its weight (value) in processing. For example, a 1 pound chicken is broiled and loses 10% of its weight. The item is sold at .9 pounds, but in reality selling that .9 pounds of chicken removes 1 pound of chicken from the inventory. This function writes a tran_code 13 tran_data record to account for the amount of the chicken that was lost due to wastage in processing.

process_items()

Update the stock on hand on the item_loc_soh table for Sales and Returns unless the item is on consignment, drop shipped, non-inventory or concession. The SOH is updated for all the orderable components of a transformed item, but not the sellable component. Also, update the item_loc_hist table for Sale transactions. Do not update for returns.

Sales history is updated at week level and also, if the Gregorian calendar is in use (ps_cal_454_ind= 'N'), at month level. Additionally, sales history is updated for both sellable and orderable components of transformed items.

process_pack()

Update the stock on hand on the item_loc_soh table for Sales and Returns. Also, update the item_loc_hist table for Sale transactions (week-level sales history for pack items, and also month-level sales history if the Gregorian calendar is in use). Do not update for returns.

process_packitems()

This function performs processing for the component items of the packitems. This would include updates/inserts into stake_item_loc, edi_daily_sales, item_loc, item_loc_hist, vat_history_data, and tran_data. All of these tables do not write records at the packitem level, but at the component item level. When figuring retails to write to these tables, the component items price ratio should always be applied against the packitems retail to come up with the correct retail for each component item. If an employee discount TDETL record has been encountered, an tran_data record with tran_code 60 will be written for each component item.

write_tran_data()

Writes a record to the tran_data insert array.

write_edi_sales()

Writes a record to edi_daily_sales.

update_snapshot()

Calls the UPDATE_SNAPSHOT_SQL.EXECUTE function to update the stake_sku_loc and edi_daily_sales tables for late transactions.

get_454_info()

Calls on the CAL_TO_454 procedure to get the equivalent 454 info of a given date.

write_vat_err_message()

This function will create and write to the VAT output file when an item does not have VAT information setup when it is expected.

vat_history_data()

Writes  a record to the vat_history table. History will only be written for the sellable item, not the orderable, and the orderable will never appear in the POS file.

consignment_data()

This function will perform processing for consignment items.  Consignment items are such when the item_supplier table has a consignment rate applied to it.  Consignment is when a retailer will allow a third party to operate under its umbrella and be paid for what it sells.  An example of consignment may be a mass-merchant who consigns the magazine section of their store to a magazine vendor.  The magazine vendor would have control over keeping the product stocked within the store.  When a magazine is sold, the retailer would get paid for the magazine, then the retailer would essentially buy the magazine from the vendor.  The consignment cost paid by the retailer to the vendor is the VAT-inclusive retail multiplied by the consignment rate divided by 100.  So if the VAT-inclusive retail price of a magazine was $10 and the consignment rate was 50, the consignment cost would be $5.

Also a completed order to the vendor should be found/created for the supplier with an orig_ind = 4 (consignment).  Consignment type invoices will be created for all PO's created for consignments if the system_options.self_bill_ind is 'Y'.

Purchase order headers are created at supplier, supplier/dept, supplier/dept/location or supplier/dept/location/item levels depending on the system_options flag gen_con_invc_itm_sup_loc being S, L or I. Purchase orders are matched 1 to 1 with sales invoices, but for returns there is no purchase order and an invoice is created for every transaction regardless of the consolidation level.  The flag system_options.gen_con_inv_freq can have values P (multiPle), W (Weekly) or M (Monthly). This controls the date used for the 1 to 1 matching which is vdate, vdate or eom_date respectively.

Also a tran_data record (tran_type 20) will be written to record the consignment transaction to the stock ledger.  The retails should be VAT inclusive or exclusive, depending on the system_options.stkldgr_vat_incl_retl_ind.

This function uses support functions: check_order(), order_head(), invc_data(), to handle the order creation-update and the invoice creation-update.

If a promotional consignment rate is present for the current promotion, over-write that returned from item_supplier

order_head()

This function inserts records into ordhead to create new orders (except for return consignment items). It sets the location to the current store number if the gen_con_invc_itm_sup_loc_ind flag is I or L, otherwise (for S) should set null. The order date is set depending on system_options.gen_con_inv_freq. The values are P (multiPle), W (Weekly) or M (Monthly). This controls the date used for the 1 to 1 matching which is vdate, vdate or eom_date respectively.

invc_data()

This function inserts/updates invc_head, invc_detail records if invc_match ind is 'Y'. Before writing the invoice records, the retail and consignment cost are converted to the associated order's currency.

The system_options parameter system_options.gen_con_invc_itm_sup_loc_ind carries values S, L or I and states the level at which separate invoices are to be generated for each supplier/dept(S), supplier/dept/location(L) or item/supplier/location(I). When a new invoice at the appropriate level is created, then for gen_con_invc_itm_sup_loc_ind values L and I, an invc_xref row is also created to link the invoice to the target location

find_and_fill_invc_detail ()

This function fills the invc_detail, updates the array and posts if the array is full

get_prom_type_info()

This function will retrieve all valid promotional transaction types from the code_detail table. Valid promotional transaction types are those where the code_type = ‘PRMT’.

get_uom_classes()

This function loads all the uom codes and their classes into a global table for look up during THEAD processing.

get_item_xform_details()

This function populates the pr_xform_items structure with the associated orderables, and returns the total yield for all rows retrieved, and also the calculated unit cost of the sellable item based on its component orderable items. This value overwrites pd_unit_cost_loc, which for standard items is populated by function item_check(…). If the returned sum of all retrieved pr_xform_items.as_yield does not equal 1, reject the record.

The processing to do this is de-encapsulated from packaged function ITEM_ XFORM_ SQL.CALCULATE_COST, as this is expected to be more efficient than calling the packaged function directly. The de-encapsulated logic is performed by the following three functions: get_loc_item_retail(), get_orderable_cost(), get_orderable_retail().

get_loc_item_retail()

This function returns the unit_retail from item_loc.  If a unit retail for the input item/location combination does not exist on the item_loc table, a call is made to retrieve the unit retail from RPM (via the PRICING_ATTRIB_SQL.GET_BASE_ZONE_RETAIL package function).

get_orderable_cost()

This function returns unit_cost from item_supp_country_loc or item_supp_country.

get_orderable_retail()

This function returns the unit_retail for each sellable item, computes the apportioned sellable retail and adds it into the returned total orderable retail.

fill_packitem_array()

This function will retrieve the component items for a packitem with the appropriate item level information into an array.

write_item_store_report()

This function will create and write to the Invalid item/location output file when an item does not exist at a location it was sold/returned at.

posting_and_restart()

Post all array records to their respective tables and call restart_file_commit to perform a commit the records to the database and restart_file_write to append temporary files to output files.

post_tran_data()

This function inserts records in the tran_data table. This is called by posting_and_restart function.

post_item_loc()

This function updates the stock_on_hand of the item_loc_soh table. This is called by posting_and_restart function.

post_item_loc_hist()

This function updates the various fields (sales_issues, value, gp, last_update_datetime and last_update_id) of the item_loc_hist table. This is called by posting_and_restart function.

post_item_loc_hist_mth()

This function updates the various fields (sales_issues, value, gp, last_update_datetime and last_update_id) of the item_loc_hist_mth table. This is called by posting_and_restart function.

post_pack()

This function updates the various fields (last_hist_export_date, first_sold, last_sold, qty_soldm, last_update_datetime and last_update_id) of the item_loc_soh table. This is called by posting_and_restart function.

post_packstore_hist()

This function updates the various fields (sales_issues, value, retail, last_update_datetime and last_update_id) of the item_loc_hist table. This is called by posting_and_restart function

post_packstore_hist()

This function updates the various fields (sales_issues, value and retail) of the item_loc_hist_mth table. This is called by posting_and_restart function.

post_vat_hist_upd()

This function updates the various fields (vat_amt, last_update_datetime and last_update_id) of the vat_history table. This is called by posting_and_restart function.

post_ edi_daily_sales_upd ()

This function updates sales_qty of the edi_daily_sales table. This is called by posting_and_restart function.

post_daily_sales_discount ()

This function updates the various fields (sales_qty, sales_retail, discount_amt, expected_retail and actual_retail) of the daily_sales_discount table. This is called by posting_and_restart function.

post_invc_detail_upd ()

This function inserts into the invc_detail_temp table. This is called by posting_and_restart function.

post_invc_detail_upd ()

This function inserts into invc_head_temp table. This is called by posting_and_restart function.

size_arrays()

This function allocates memory for the arrays used in this program.

resize_arrays()

This function reallocates memory for the insert arrays.

write_lock_rej()

This function will write the current record set from the input file (THEAD-{TDETL}-TTAIL) that was rejected due to  lock error to the lock file.

concession_data()

This function inserts records into concession_data for non-pack concession items.

deal_actuals_insert_update ()

This function accepts a list of primary key values and update values for the deal_actuals_item_loc table, and a row_id which is null if the row does not exist yet. If it does not exist, a new row is inserted, otherwise the row_id and update values are written to the holding array, for bulk update later.

check_deal_actuals_exists()

This function accepts a list of primary keys for table deal_actuals_item_loc, does a look up and returns the row_id or null if it exists, or not.

write_vendor_deal_actuals ()

This function causes actual vendor contribution amounts to be written to the deal_actuals_item_loc table for each of the computed vendor funding contributions held in the pr_deal_contribs array. Calls check_deal_actuals_exists to check if each target primary key set exists, and calls deal_actuals_insert_update to insert a new row, or write update information to the holding array if a row already exists.

post_deal_actuals ()

This function updates the various fields (actual_turnover_units, actual_turnover_revenue and actual_income) of the deal_actuals_item_loc. This is called by posting_and_restart function.

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

- write out rejected record to the reject file using write_to_rej_file function by passing pointer to detail record structure, number of bytes in structure, and reject file pointer, or use the write_lock_rej() function to write to the lock reject file in case the non-fatal error was a lock error,

Input File:

The input file should be accepted as a runtime parameter at the command line. All number fields with the number(x,4) format assume 4 implied decimal included in the total length of 'x'.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Vat include indicator, Vat region, Currency code, and Currency retail decimals. When the sa_ind is 'N' these values will not be used and retrieved from the system.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Item Level, Transaction Level, Pack_ind, Dept, Class, and Subclass. When the sa_ind is 'N' these values will not be used and retrieved from the system. Also, the UOM at which the item was sold will been converted to the standard UOM for the item. When the sa_ind is on, all items are assumed to be items.

| 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 | Char(10) | specified by external system | ID of current line being processed by input file. |
| | File Type Definition | Char(4) | POSU | Identifies file as 'POS Upload' |
| | File Create Date | Char(14) | create date | date file was written by external system |
| | Location Number | Number(10) | specified by external system | Store identifier |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Vat include indicator | Char(1) | | Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit |
| | Vat region | Number(4) | | Vat region the given location is in. Not required but populated by Retek sales audit |
| | Currency code | Char(3) | | Currency of the given location. Not required but populated by Retek sales audit |
| | Currency retail decimals | Number(1) | | Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit |
| Transaction Header | File Type Record Descriptor | Char(5) | THEAD | Identifies transaction record type |
| | File Line Identifier | Char(10) | specified by external system | ID of current line being processed by input file. |
| | Transaction Date | Char(14) | transaction date | date sale/return transaction was processed at the POS |
| | Item Type | Char(3) | REF<br>ITM | item type will be represented as a REF or ITM |
| | Item Value | Char(25) | item identifier | the id number of an ITM or REF |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Dept | Number(4) | Item’s dept | Dept of item sold or returned. Not required but populated by Retek sales audit |
| | Class | Number(4) | Item’s class | Class of item sold or returned. Not required but populated by Retek sales audit |
| | Subclass | Number(4) | Item’s subclass | Subclass of item sold or returned. Not required but populated by Retek sales audit |
| | Pack Indicator | Char(1) | Item's pack indicator | Pack indicator of item sold or returned. Not required but populated by Retek sales audit |
| | Item level | Number(1) | Item's item level | Item level of item sold or returned. Not required but populated by Retek sales audit |
| | Tran level | Number(1) | Item's tran level | Tran level of item sold or returned. Not required but populated by Retek sales audit |
| | Wastage Type | Char(6) | Item’s wastage type | Wastage type of item sold or returned. Not required but populated by Retek sales audit |
| | Wastage Percent | Number(12) | Item’s wastage percent | Wastage percent of item sold or returned. Not required but populated by Retek sales audit |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Transaction Type | Char(1) | ‘S’ – sales<br>‘R’ - return | Transaction type code to specify whether transaction is a sale or a return |
| | Drop Shipment Indicator | Char(1) | 'Y'<br>'N' | Indicates whether the transaction is a drop shipment or not. If it is a drop shipment, indicator will be 'Y'. This field is not required, but will be defaulted to 'N' if blank. |
| | Total Sales Quantity | Number(12) | | Number of units sold at a particular location with 4 implied decimal places. |
| | Selling UOM | Char(4) | | UOM at which this item was sold. |
| | Sales Sign | Char(1) | ‘P’ - positive<br>‘N’ - negative | Determines if the Total Sales Quantity and Total Sales Value are positive or negative. |
| | Total Sales Value | Number(20) | | Sales value, net sales value of goods sold/returned with 4 implied decimal places. |
| | Last Modified Date | Char(14) | | For VBO future use |
| | Catchweight Indicator | Char(1) | NULL | Indicates if item is a catchweight item. Valid values are ‘Y’ or NULL |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Actual Weight Quantity | Number(12) | NULL | The actual weight of the item, only populated if catchweight_ind = 'Y' |
| | Sub Trantype Indicator | Char(1) | NULL | Tran type for ReSA<br>Valid values are 'A', 'D', NULL |
| Transaction Detail | File Type Record Descriptor | Char(5) | TDETL | Identifies transaction record type |
| | File Line Identifier | Char(10) | specified by external system | ID of current line being processed by input file. |
| | Promotional Tran Type | Char(6) | promotion type – valid values see code_detail table. | code for promotional type from code_detail, code_type = 'PRMT' |
| | Promotion Number | Number(10) | promotion number | promotion number from the RMS |
| | Sales Quantity | Number(12) | | number of units sold in this prom type with 4 implied decimal places. |
| | Sales Value | Number(20) | | value of units sold in this prom type with 4 implied decimal places. |
| | Discount Value | Number(20) | | Value of discount given in this prom type with 4 implied decimal places. |
| | Promotion Component | Number(10) | NULL | Links the promotion to additional pricing attributes |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| Transaction Trailer | File Type Record Descriptor | Char(5) | TTAIL | Identifies file record type |
| | File Line Identifier | Char(10) | specified by external system | ID of current line being processed by input file. |
| | Transaction Count | Number(6) | specified by external system | Number of TDETL records in this transaction set |
| File Trailer | File Type Record Descriptor | Char(5) | FTAIL | Identifies file record type |
| | File Line Identifier | Number(10) | specified by external system | ID of current line being processed by input file. |
| | File Record Counter | Number(10) | | Number of records/transactions processed in current file (only records between head & tail) |

Invalid Item/Store File:

The Invalid Item/Store File will only be written when a transaction holds an item that does not exist at the processed location. In the event this happens, the relationship will be created during the program execution and processing will continue with the item and store number being written to this file for reporting.

VAT File:

The VAT file will only be written if a particular item cannot retrieve a VAT rate when one is expected (e.g. the system_options.vat_ind is on). In this event, a non-fatal error will occur against the transaction and a record will be written to this file and the Reject file.

Reject File:

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will be created by the interface library routines and the detail records will be created using the write_to_rej_file function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the restart_write_function should be called (passing the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

Error File:

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

### Technical Issues

Assumption: Variable weight UPCs are expected to already be converted to a VPLU with the appropriate quantity.

### Output Specifications

N/A

### Scheduling Considerations

| | |
|---|---|
| Processing Cycle: | PHASE 2 (daily) |
| Scheduling Diagram: | This program will likely be run at the beginning of the batch run during the POS polling cycle. It can be scheduled to run multiple times throughout the day, as POS data becomes vailable. |
| Pre-Processing: | N/A |
| Post-Processing: | N/A |
| Threading Scheme: | N/A |

### Restart Recovery

The logical unit of work for the sales/returns upload module will be a valid item sales transaction at a given store location. The location type will be inferred as a store type and the item can be passed as an item or reference item type. The logical unit of work will be defined as a number of these transaction records. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups of numbers equal to the commit_max_ctr. After all records in a given read are processed (or rejected either as a reject record or a lock error record), the restart commit logic and restart file writing logic will be called, and then the next group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters) and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

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).

Error handling will recognize three levels of record processing: process success, non-fatal errors, and fatal errors. Item level validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining fields will be validated, but the record will be rejected and written to the reject file or written to the lock file depending on the reject reason. If a fatal error is returned, then file processing will end immediately. A restart will be initiated from the file pointer position saved in the restart_bookmark string at the time of the last commit point that was reached during file processing.

# Upload stock count results [STKUPLD]

### Design Overview

The purpose of this batch module is to accept cycle count details from an external system. The cycle count transactions will be compared with Retek system snapshots of stock on hand at the time of the cycle count to determine the stock and/or dollar adjustments to be made. The following common functions will be performed on each stock record read from the input file:

- if record exists on STAKE_SKU_LOC then update it
- if record doesn't exist on STAKE_SKU_LOC validate that item/location exists in system
- insert a record into STAKE_SKU_LOC
- insert stock take record into STAKE_SKU_LOC.
- if record is orderable-only transformed item then treat as if it is a regular item and mark 'O' on xform_item_type column in STAKE_SKU_LOC
- if record is sellable-only transformed item and has no associated orderable-only item already in the stock count then record will be rejected
- if record is sellable-only transformed item then program will roll the physical count quantity of the sellable-only transformed item up to its associated orderable-only transformed item since sellable-only transformed item has no snapshot and mark 'S' on xform_item_type column in STAKE_SKU_LOC
- if record is non-inventoriable item then reject except if it is part of the transformed item
- if record is a pack - update/insert information on STAKE_SKU_LOC for all component items

| TABLE | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|
| item_loc | Yes | No | No | No |
| item_loc_soh | Yes | No | No | No |
| item_master | Yes | No | No | No |
| item_xform_head | Yes | No | No | No |
| item_xform_detail | Yes | No | No | No |
| partner | Yes | No | No | No |
| price_zone_group_store | Yes | No | No | No |
| stake_head | Yes | No | No | No |
| stake_location | Yes | Yes | No | No |
| stake_prod_loc | Yes | No | No | No |
| stake_product | Yes | No | No | No |
| stake_qty | No | Yes | No | No |
| stake_sku_loc | No | Yes | Yes | No |
| system_options | Yes | No | No | No |

| TABLE | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|
| v_packsku_qty | Yes | No | No | No |
| Wh | Yes | No | No | No |

This program reads a user-created interface file of cycle counts. Files will be unique to location and cycle count ID. All records will be validated for layout. Invalid layouts will produce fatal errors. Fields will be validated for content. Invalid contents will produce non-fatal errors. Valid records will update the physical_count_qty field on STAKE_SKU_LOC for a given item/location/cycle count combination. If the item is a pack, component items will have their component quantity added to the pack_comp_qty field on STAKE_SKU_LOC. If an item does not exist on STAKE_SKU_LOC, the item/location combination will be validated on the item/location tables and a new record will be inserted to STAKE_SKU_LOC.

Fatal errors will terminate file processing. Non-fatal errors will discontinue record processing and will write invalid record to a reject file.

File layout will be verified by interface library routines:

- get_record: validates common fields in file head record and fills structure of remaining fields that are passed from this program
- process_dtl_ftail: called after end-of-file is reached. Will process file trailer record by validating its layout and verifying that the file record counter is set properly.

Re-run:

- If this program terminates normally, restart without recovery.
- If this program terminates abnormally, restart without recovery.

#### Scheduling Constraints

Processing Cycle: PHASE 3 (Daily)

Scheduling Diagram: This program will probably be run at the start of the batch cycle during POS polling, or possibly at the end of the batch run if pending warehouse transactions exist. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

The logical unit of work for the stock take upload module will be a count of discrete inventory transactions. Each record will be uniquely identified by a location and item. The logical unit of work will be defined as a number of these transaction records, determined by the commit_max_ctr field on the restart_control table.

The file records will be grouped in numbers equal to the commit_max_ctr. After all records in a given read are processed (or rejected), the restart commit logic and restart file writing logic will be called, after which the following group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters) and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

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 10,000 records (subject to change based on experimentation).

Error handling will recognize three levels of record processing: process success, non-fatal errors, and fatal errors. Item level validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining fields will be validated but the record will be rejected and written to the reject file. If a fatal error is returned, file processing will end immediately. A restart will be initiated from the file pointer position saved in the restart_bookmark string at the time of the last commit point that was reached during file processing.

### Program Flow

N/A

### Shared Modules

valid_date: interface library function.

DISTRIBUTE_SQL.DISTRIBUTE

STKCOUNT_SQL.ROLLUP_SELLABLE_ONLY_ITEM

NEW_ITEM_LOC

PRICING_ATTRIB_SQL.GET_EXTERNAL_FINISHER_RETAIL

PRICING_ATTRIB_SQL. GET_BASE_ZONE_RETAIL

### Function Level Description

init()

initialize restart recovery, call out restart_file_init().

open input file

- file should be specified as input parameter to program
  declare final output filename (used in restart_write_file logic)
  open reject file ( as a temporary file for restart )
- file should be specified as input parameter to program
  call restart_file_init logic
- assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), cycle_count, stocktake date
  if fresh start (l_file_start = 0)
  read file header record (get_record)
  validate head (validate_head())
  else fseek to l_file_start location
  initialize locations

process()

loop - fread rows (equal to commit counter) of input file

if end of file encountered, decrement for loop counter and set end of file flag to true

for loop to process all records read

copy input detail structure elements to stake_sku_loc structure elements

validate elements (validate_detail())

if non-fatal error occurs write detail structure to reject file (write_to_rej_file) and

continue at the top of the for-loop

if multi-channel check

if record is sellable-only transformed item, distribute count_qty among the virtual warehouses within the physical warehouse based on its associated orderable-only item's snapshot

if record is not sellable-only transformed item, continue with normal processing

update stake_sku_loc

if record doesn't exist, validate that item/location is valid

if invalid then non-fatal error -write record & continue

insert to stake_sku_loc (if display pack also insert component items)

end loop for loop to process individual records

insert structure of arrays (for valid record counter) into stake_sku_loc

restart file commit - save current input file position, and application image (cnt, cycle count & date)

restart write file function

if end of file reached then break from while loop

end outer loop to read from file

restart commit final

validate_head()

if file type != ‘STKU’ then fatal file type error

copy stocktake_date into variable

nullpad stocktake_date

copy loc_type into variable ( value will always be warehouse ‘W’) nullpad stocktake_dat

nullpad loc_type

copy loc_value into variable

nullpad loc_value

copy store_value, wh_value, and loc_value into variables ( store will always be –1)

get cycle count  for location and stocktake_date.

validate  cycle count.

validate_detail()

if record type != FDETL then fatal file layout error

do standard string validations - if any return non-fatal error then set non-fatal error flag to true

nullpad all fields

left shift item and qty

check that store and qty are all numeric

place decimal in qty field

check if record is non-inventoriable and/or not a sellable-only transformed item then write to reject file and return non-fatal error

check record’s item type

if ‘ITM’, use record’s item value for processing

if ‘REF’, use record’s parent item for processing

if not ‘ITM’ nor ‘REF’ return fatal error

validate if record is sellable-only transformed item and use ‘S’ for marking on xform_item_type column in STAKE_SKU_LOC

for unit and value stock count if record is not sellable-only transformed item and does not match dept/class/subclass found on STAKE_PROD_LOC then write to reject file and return non-fatal error

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

- write out rejected record to the reject file using write_to_rej_file function, pass pointer to detail record structure, number of bytes in structure, and reject file pointer

### I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

| Record Name | Field Name | Field Type | Description |
|---|---|---|---|
| File Header | file type record descriptor | Char(5) | hardcode 'FHEAD' |
| | file line identifier | Number(10) | Id of current line being processed., hardcode '000000001' |
| | file type | Char(4) | hardcode 'STKU' |
| | file create date | Date(14) YYYYMMDD DHHMISS | date written by convert program |
| | stocktake_date | Date(14) YYYYMMDD DHHMISS | stake_head.stocktake_date |
| | cycle count | Number(8) | stake_head.cycle_count |
| | loc_type | Char(1) | hardcode 'W','S' or 'E' |
| | location | Number(10) | stake_location.wh or stake_location.store |
| Transaction record | file type record descriptor | Char(5) | hardcode 'FDETL' |
| | file line identifier | Number(10) | Id of current line being processed, internally incremented |
| | item type | Char(3) | hardcode 'ITM' |
| | item value | Number(25) | item id |
| | inventory quantity | Number(12,4) | total units or total weight |

| Record Name | Field Name | Field Type | Description |
|---|---|---|---|
| | location description | Char(30) | NULL |
| File trailer | file type record descriptor | Char(5) | hardcode 'FTAIL' |
| | file line identifier | Number(10) | Id of current line being processed, internally incremented |
| | file record count | Number(10) | Number of detail records. |

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will be created by the interface library routines and the detail records will be created using the write_to_rej_file function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the restart_write_function should be called (passing the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

### Technical Issues

N/A

## Stock count stock on hand updates [STKVAR]

### Design Overview

The stkvar.pc program updates the stock on hand and computes the total cost and total retail in the stake_prod_loc.

Tables Affected:

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|---|
| CLASS | No | Yes | No | No | No |
| ITEM_LOC | Yes | No | No | Yes | No |
| ITEM_LOC_SOH | Yes | No | No | Yes | No |
| ITEM_MASTER | No | Yes | No | No | No |
| ITEM_SUPP_COUNTRY | No | Yes | No | No | No |
| ITEM_XFORM_DETAIL | No | Yes | No | No | No |
| ITEM_XFORM_HEAD | No | Yes | No | No | No |
| NWP | No | No | Yes | Yes | No |
| NWP_FREEZE_DATE | No | Yes | No | No | No |
| STAKE_CONT | No | Yes | No | No | Yes |
| STAKE_HEAD | No | Yes | No | No | No |
| STAKE_PROD_LOC | Yes | No | No | Yes | No |
| STAKE_QTY | No | Yes | No | No | No |
| STAKE_SKU_LOC | No | Yes | No | Yes | No |
| WH | No | Yes | No | No | No |

Indexes:

STAKE_PROD_LOC (dept, store, wh, data_type)

This program updates the stock on hand for all items as a result of a stock take.

The program is driven by STAKE_CONT, in conjunction with STAKE_SKU_LOC where the ITEM, loc and cycle count on STAKE_SKU_LOC match those on STAKE_CONT, and where STAKE_CONT run_type = 'A' (for adjustment).

For each row retrieved from the above tables, the unit systems are processed as follows:

An ITEM_LOC_SOH record is updated for every ITEM/loc combination. The new stock on hand = item_loc_soh.stock_on_hand - snapshot_stock_on_hand_qty (from the STAKE_SKU_LOC table) + the physical count quantity on STAKE_SKU_LOC. In addition, the pack_comp_soh field is updated on ITEM_LOC when a pack is processed for each component ITEM in the pack.

Total cost and total retail are computed as the snapshot unit retail times the sum of the physical count quantity plus the snapshot in-transit (from the STAKE_SKU_LOC table).

STAKE_PROD_LOC total cost and total retail amounts are updated with the total cost and total retail for each department, class, subclass, location combination that exists on the cycle count. A record for each is added. If a record already exists on the table, the total cost or retail amount value is adjusted to be the existing total cost or retail amount + the cycle count cost or retail. If no record exists, a new one is added to the table with the value of total cycle count cost or retail for the total cost or retail amount. If the stock ledger is designated not to include VAT on the SYSTEM_OPTIONS table, the total retail amount will have any VAT amount stripped from it.

Re-run:

If this program terminates normally, ITEM_LOC, STAKE_QTY,

ITEM_LOC_SOH, STAKE_PROD_LOC_STOCK and STAKE_CONT must be recovered prior

to restart. If this program terminates abnormally, restart without recovery.

The syntax for invoking this program is:

stkvar userid/pswd [ report_name ].

Here are some examples:

- stkvar *userid/pswd* ( it will not produce any report.)
- stkvar *userid/pswd* any.rpt ( it will produce a report, any.rpt. )

### Input Specifications

```
EXEC SQL DECLARE c_item CURSOR FOR
   SELECT /* ORDERED USE_HASH(stake_cont) FULL(stake_head)*/
          ssl.item,
          ssl.loc_type,
          ssl.location,
          c.class_vat_ind,
          ROWIDTOCHAR(sc.ROWID),
          ssl.snapshot_on_hand_qty,
          NVL(ssl.snapshot_in_transit_qty,0),
          NVL(ssl.snapshot_unit_cost, 0),
          NVL(ssl.snapshot_unit_retail, 0),
          NVL(ssl.physical_count_qty, 0),
          ssl.pack_comp_qty,
          NVL(ssl.dept, 0),
          NVL(ssl.class, 0),
          NVL(ssl.subclass, 0),
          ROWIDTOCHAR(ssl.ROWID),
          im.pack_ind,
          TO_CHAR(sh.stocktake_date,'YYYYMMDD'),
          sh.stocktake_type,
          sh.cycle_count,
```

```
          ssl.xform_item_type,
          im.deposit_item_type,
          ';' || ssl.item   ||
          ';' || TO_CHAR(ssl.loc_type) ||
          ';' || TO_CHAR(ssl.location)
     FROM stake_sku_loc ssl,
          stake_cont sc,
          stake_head sh,
          item_master im,
          class c,
          wh w,
          stake_qty sq,
          v_restart_dept rv
    WHERE sc.run_type(+)                = 'A'
      AND sc.item(+)                    = ssl.item
      AND sc.loc_type(+)                = ssl.loc_type
      AND sc.location(+)                = ssl.location
      AND sc.cycle_count(+)             = ssl.cycle_count
      AND sh.cycle_count                = ssl.cycle_count
      AND NVL(ssl.xform_item_type, ' ') <>'S'/*exclude sellable only
items*/
      AND im.item                       = ssl.item
      AND im.item_level                 = im.tran_level
      AND im.dept                       = c.dept
      AND im.class                      = c.class
      AND ssl.location                  = w.wh (+)
      AND (im.pack_ind     = 'N' OR
          (ssl.loc_type    = 'W' AND
           im.pack_ind     = 'Y' AND
           w.finisher_ind  = 'N'))
      AND (sc.rowid is not null
           OR (sh.stocktake_date + :pi_cycle_count_lag_days <=
TO_DATE(:ps_vdate,
               'YYYYMMDD')
               AND ssl.processed != 'P'))
      AND sq.cycle_count   = ssl.cycle_count
      AND sq.loc_type      = ssl.loc_type
      AND sq.location      = ssl.location
```

```
        AND sq.item          = ssl.item
        AND rv.driver_value  = im.dept
        AND rv.driver_name   = :ora_restart_driver_name
        AND rv.num_threads   = TO_NUMBER(:ora_restart_num_threads)
        AND rv.thread_val    = TO_NUMBER(:ora_restart_thread_val)
        AND (ssl.item           > NVL(:ora_restart_item,-999) OR
               (ssl.item        = :ora_restart_item AND
                  (ssl.loc_type > TO_CHAR(:ora_restart_loc_type) OR
                  (ssl.loc_type = TO_CHAR(:ora_restart_loc_type) AND
                  (ssl.location  >
TO_NUMBER(:ora_restart_location))))))
 ORDER BY ssl.item,
          ssl.loc_type,
          ssl.location;
```

### Scheduling Constraints

| | |
|---|---|
| Processing Cycle: | PHASE 3 |
| Scheduling Diagram: | N/A |
| Pre-Processing: | N/A |
| Post-Processing: | N/A |
| Threading Scheme: | DEPT |

### Restart Recovery

This program will be threaded by department and will utilize restart/recovery logic based on item/store/wh.

### Shared Modules

PRICING_ATTRIB_SQL.GET_RETAIL

# Store add [STOREADD]

### Design Overview

This program will add all information necessary for a new store to function properly. When a store is added to the system, the store will be accessible in the system only after storeadd.pc is run.

The batch program loops through each record on the store_add table.

Also, it supports the replenishment system in RMS 9.0.

### Scheduling Constraints

| | |
|---|---|
| Processing Cycle: | Daily, Ad Hoc Phase |
| Scheduling Diagram: | N/A |
| Pre-Processing: | N/A |
| Post-Processing: | N/A |
| Threading Scheme: | Table based processing, don't use multithreading. |

### Restart/Recovery

Select ALL FIELDS from store_add.

After a record on store_add has been processed successfully, it is immediately deleted. Thus, restart recovery is implicit in storeadd.pc.

### Program Flow

N/A

### Function Level Description

main()

Check command line for required arguments.

Call LOGON to connect to the database.

Call Init to initialize the program.

Call process to fetch records from the store_add table.

Call final to cleanup.

init()

Declare restart variables

Get system variables (ELC indicator and pricing rule)

process()

Loop through store_add table

Set “new” variable indicators

Insert into store table

If elc_ind = ‘Y’

  Call Insert_Cost_Zones

end if;

If repl_ind = ‘Y’

  Call Copy_Repl_info

end if;

If copy_close_ind = ‘Y’

  Call Copy_Close_Sched

End if;

If copy_dlvry_ind = ‘Y’

  Call Copy_Dlvry_Sched

End if;

Call Insert_Stock_Loc_Traits

Delete from store_add

Insert_Pricing_Zone()

This function inserts records into pricing zone tables as is appropriate to the store being created:

insert corporate pricing zone information

insert store pricing zone information

if new_price_zone_ind = ‘N’

  insert zone info for existing currency

else

  insert new zone info

Insert_Cost_Zones()

This function inserts records into cost zone table as is appropriate to the store being created:

If there is a corporate cost zone group, insert corporate cost zone information to cost_zone_group_loc.

If there is a location cost zone group, insert appropriate information into the cost_zone and cost_zone_group_loc tables.

if new_cost_zone_ind = ‘N’

  insert cost zone detail records

else

  insert new zone

Copy_Store_Items()

This function calls the like_store_execute_sql.copy_store_items package function, which copies all item/store records from the like_store and inserts them for the new store.

Copy_Repl_Info()

This function copies all replenishment information for items from the selected like_store and copies them into replenishment tables for the new store.

Copy_Close_Sched()

This function copies all the location closed information from the selected like_store which the close_date are greater or equal to current and copies them into location_closed and company_closed_excep tables for the new store.

Copy_Dlvry_Sched()

This function copies all the location delivery schedules from the selected like_store and copies them into the loc_dlvry_sched, loc_dlvery_sched_days, and loc_dlvry_sched_exc tables for the new store.

Insert_Stock_Loc_Traits()

This function calls the stkledgr_sql.stock_ledger_insert and loc_traits_sql.new_org_hier package functions, which insert records into the stock ledger and hierarchy tables.

Update_regionality_matrix()

This function will insert records to the store_hierarchy and regionality_temp tables.

Insert_pos_store()

This function will insert records into the pos_store table.

final()

This function stops restart recovery.

### I/O Specification

N/A

### Technical Issues

N/A

# Ticket output file [TCKTDNLD]

### Design Overview

This program will create an output file containing all of the information to be printed on a ticket or label for a particular ITEM/location. This program is driven by the "requests" for tickets that exist on the TICKET_REQUEST table. Information to be printed on the ticket is then retrieved based on the ITEM, location and the ticket type requested. The details, which should be printed on each type of ticket, are kept on the TICKET_TYPE_DETAIL table. Specific details, which will be written to the output file, are taken from the various item tables (i.e. ITEM short description from ITEM_MASTER, retail price from ITEM_ZONE_PRICE).

### Scheduling Constraints

| | |
|---|---|
| Processing Cycle: | Ad Hoc (Daily) |
| Scheduling Diagram: | N/A |
| Pre-Processing: | N/A |
| Post-Processing: | N/A |
| Threading Scheme: | N/A |

### Restart Recovery

Restartability will exist implicitly within this program. Because records will be deleted after they are selected, no explicit code is needed to restart in the event of a failure.

The lack volume of data processed by this program, in addition to the lack of an appropriate threading mechanism, negates the need for Retek multi-threading capabilities.

Driving Cursor:

```
SELECT tr.ticket_type_id,
       tr.item,
       tr.qty*POWER(10, :pi_amount_implied_digits),
       tr.loc_type,
       tr.location,
       tr.country_of_origin,
       tr.unit_retail,
       tr.multi_units,
       tr.multi_unit_retail,
       tr.order_no,
       th.sel_ind,
       nvl(s.lang, :ps_primary_lang),
       ROWIDTOCHAR(tr.rowid)
  FROM ticket_request tr,
       store s,
       ticket_type_head th
```

```
         WHERE tr.ticket_type_id = th.ticket_type_id
           AND tr.print_online_ind = :ps_print_online_ind
           AND tr.location = NVL(:ps_print_location, tr.location)
           AND tr.location = s.store(+)
           AND (tr.price_change_id IS NULL OR
                (TRUNC(tr.price_change_eff_date) -
TO_NUMBER(:ps_days_in_advance) =
TRUNC(TO_DATE(:ps_vdate,'YYYYMMDDHH24MISS'))))
    ORDER BY item, location, tr.ticket_type_id;
```

### Program Flow

N/A

### Shared Modules

N/A

### Function Level Description

init()

Functional details:

This function should initialize the restart/recovery process. The output file should be opened, and if it is not a “restart”, then file header information should be written. The system vdate is selected for the file create date used in the output file header. The format_buffer function should be called to format output strings. The size_arrays function should be called to size the fetch and delete arrays.

Technical details:

The output file should be written in the style necessary for Retek restart/recovery. That is, a temp file should be opened and initialized, and the final file should only be written to when the restart/recovery commit logic is called (using restart_file_write).

format_buffer()

This function creates format strings for the file output that will be done later.

size_arrays()

This function allocates space for the read array.

process()

Functional Details:

This function should write transaction records to the output file for each item/ticket type/location combination on the ticket_request table. For each record a transaction header should be written to the output file and the ticket_item function should be called to write the detail items for the details associated with the ticket type. If the item is a pack item, however, the ticket_pack function should be called to first write component item records (the ticket_item function will be called within the ticket_pack function for each component item.). After each record from the driving cursor is processed, it will be deleted from the ticket_request table. Finally, when all of the records have been processed from the table, a file trailer should be written to the output file.

Technical Details:

The function should fetch records from the driving cursor into arrays. The arrays should be sized to match the value of the maximum commit counter on the restart_control table. Once the records are fetched, each record should be processed in a for-loop. After all of the records have been processed in the for loop, the records should be array deleted from the ticket_request table by rowid, and the restart_commit logic should be called. Output file line counters, transaction counters, etc. should be saved into the application image array string that is passed to the restart_control function.

write_THEAD()

Called by process(), this function writes transaction header line the output file.

get_item_master()

Called by process(), this function gets the attributes of a specific item.

ticket_pack()

This function will be called from process if the item on the ticket_request table is a pack item. This function should fetch all of the component items in the pack, along with pack quantity information, and write a pack record for each component. Further the ticket_item function should be called for each component.

ticket_item()

This function should select all of the records from the ticket_type_detail table with the ticket_type from the ticket_request record. Detail records should be written out to the output file for each detail record retrieved. Either item information or attribute information should be written to the output file. If the ticket item is to be written (fetched ticket item is not null) get_ticket_item is called to retrieve this appropriate intormation. If the attribute information is to be written (fetched attribute column is not null) then a function should be called to get the appropriate attribute information (get_UDA).

get_UDA()

This function should fetch the user defined attribute (UDA) value and description assocated with the attribute value selected from the ticket detail table. The UDA description will be selected for the UDA and the ITEM from either the UDA_item_lov and the UDA_value tables, the UDA_item_ff table or the UDA_item_date table. The UDA value will be written to the output file in the “value” location of the detail line.

get_ticket_item()

This function retrieves the database information which corresponds to the requested ticket item, according to the table below.

| TICKET ITEM | OUTPUT FILE VALUE |
|---|---|
| UOM | Price per unit of measure from item_master. |
| ITEM | Retek ITEM value |
| ITDS | ITEM description (from item_master) |
| ITSD | ITEM short description from the item_master table |
| VAR | The primary variant (ref_item) from the item_master table |
| DIF1 | Diff_1 value from item_master |
| DIF2 | Diff_2 value from item_master |
| WGHT | Case weight from item_supp_country_dim table |
| DEPT | Department from item_master & department name from deps table |
| CLAS | Class value from item_master table & class name from class table |
| SBCL | Subclass from item_master table & subclass name from subclass table |
| RTPC | Selling retail price from driving cursor (if available), otherwise from item_zone_price for item/store (use base zone value for warehouses). |
| SRTP | Suggested retail price (mfg_rec_retail) from item_master |
| MUPC | Multi-units and multi-unit retail from driving cursor (if available), otherwise from item_zone_price for item/store (use base zone value for warehouses) |
| SUPR | Supplier from ordhead for most recent PO for the SKU. |
| SUP1 | Supplier diff_1, from item_supplier |
| SUP2 | Supplier dif_2, from item_supplier. |
| STRE | Store from driving cursor |
| WHSE | Warehouse from driving cursor |
| COOG | Country of origin from driving cursor if available, else from the last PO (see supplier). |
| DPST | Deposit Item Return Amount. |
| DTOT | Total Deposit Item Unit Retail. |
| NETV | Unit Retail Net of VAT |

get_price_uom()

Called out by get_ticket_item , this function gets the retail price of container item.

get_item_supplier()

This function gets the supplier diffs of a particular item.

get_item_master()

This function gets the description, dept, pack_ind, item_level and tran_level of a particular item.

get_ref_item()

This function gets the reference item of a particular item.

get_item_diffs()

This function gets the diffs and diffs description of a particular item.

get_deps()

This function gets the department name of the department on to which a particular item belongs.

get_class()

This function gets the class name of the class on to which a particular item belongs.

get_subclass()

This function gets the subclass name of the subclass on to which a particular item belongs.

get_rec_retail()

This function gets the suggested unit retail of a particular item.

get_ordhead()

This function gets the order attributes of a particular item.

get_nonpack_info()

Called out by get_class() and get_subclass(). Gets the item info and retail price of non-pack items.

get_item_info()

Called out by get_ticket_item().This function gets the short description, class, subclass and item_parent of a particular item.

get_item_zone_price()

Called out by various functions. Gets the item zone price of a particular item.

get_eu_retail()

Called out by get_ticket_item if ticket_type_id = ‘EURO’.

get_item_parent_desc()

Called out by get_ticket_item if ticket_type_id = ‘IPDS’. Gets the item_parent description.

get_container_item()

Called out by get_ticket_item , this function will check if the item is a contents item and get the corresponding container item.  Also, this function should also retrieve the unit retail of the container item if applicable.

get_unit_retail_net_vat()

Called out by get_ticket_item, this function will compute the unit retail net of Vat for the location of the ticket, less the standard vat set-up for the item.

write_FHEAD(void);

Called by init(), this function will write the FHEAD line of the output file.

write_TTAIL(void);

Called by init(), this function will write the transaction tail line to output file.

write_TDETL(void);

Called by ticket_item(), this function will write the TDETL line of the output file.

write_FTAIL(void);

Called by final(), this function will write the FTAIL line of the output file.

write_TCOMP(void);

Called by process(), this function will write the TCOMP line of the output file.

final()

Retek restart/recovery process will be closed by calling the internal API function, and all appropriate output files will be close and temp files will be removed.

#### 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 Sequence | Number(10) | | Line number of the current file |
| | File Type Definition | Char(4) | TCKT | Identifies file as ‘Print Ticket Requests’ |
| | File Create Date | Date | create date | date file was written by external system |
| Transaction Header | File Type Record Descriptor | Char(5) | THEAD | Identifies file record type |
| | File Line Sequence | Number(10) | | Line number of the current file |
| | ITEM | Char(25) | | ID number of the transaction level, non-pack item or the pack item |
| | Ticket Type | Char(4) | | ID which indicates the ticket type to be printed |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Location Type | Char(1) | S - Store<br>W - Warehouse | Identifies the type of location for which tickets will be printed |
| | Location | Char (10) | | number of the store or warehouse for which tickets will be printed |
| | Quantity | Number(12,4) | | the quantity of tickets to be printed |
| Transaction Component | File Type Record Descriptor | Char(5) | TCOMP | Identifies file record type |
| | File Line Sequence | Number(10) | | Line number of the current file |
| | ITEM | Char(25) | | ID number of the ITEM |
| | Quantity | Number(12,4) | | Quantity of the component ITEM as part of the whole; if ITEM on the header record is a transaction level ITEM, the value in this field will be 1. |
| Transaction Detail | File Type Record Descriptor | Char(5) | TDETL | Identifies file record type |
| | File Line Sequence | Number(10) | | Line number of the current file |
| | Detail Sequence Number | Number(10) | | Sequential number assigned to the detail records |
| | Ticket Item | Char(4) | | ID indicating the detail to be printed on the ticket |
| | Attribute Description | Char(40) | | Description of the attribute (from the UDA Table) |
| | Value | Char(100) | | Detail to be printed on the ticket (i.e. REF_ITEM, Department Number, ITEM description) |
| | Supplement | Char(300) | | Supplemental description to the Value (i.e. Department Name) |
| Transaction Trailer | File Type Record Descriptor | Char(5) | TTAIL | Identifies file record type |
| | File Line Sequence | Number(10) | | Line number of the current file |

| Record Name | Field Name | Field Type | Default Value | Description |
|---|---|---|---|---|
| | Transaction Detail Line Count | Number(6) | sum of detail lines | sum of the detail lines within a transaction |
| File Trailer | File Type Record Descriptor | Char(5) | FTAIL | Identifies file record type |
| | File Line Sequence | Number(10) | | Line number of the current file |

### Technical Issues

The program could be sped up by outer joining ticket_type_detail into the driving cursor, and avoiding the c_ttd cursor which must be opened for each record in our fetch array.

# Warehouse retail [WHADD]

### Design Overview

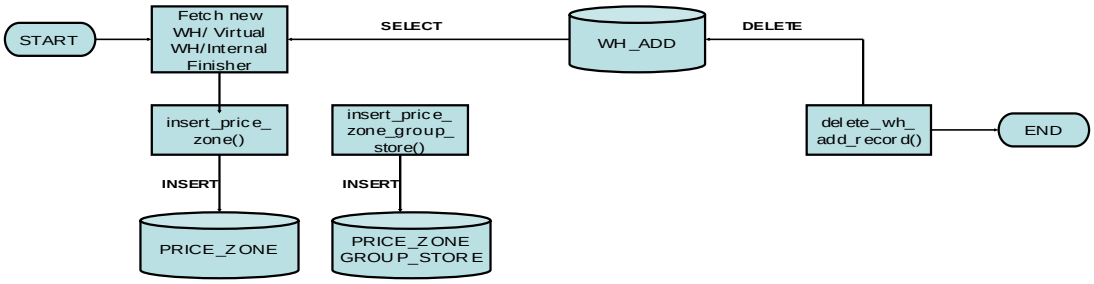
The whadd.pc batch program reads new warehouses, virtual warehouses and/or internal finishers from the WH_ADD table. Records will be inserted to the PRICE_ZONE and PRICE_ZONE_GROUP_STORE for each retrieved record.

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|---|
| WH_ADD | | Yes | No | No | Yes |
| PRICE_ZONE | | No | Yes | No | No |
| PRICE_ZONE_GROUP_STORE | | No | Yes | No | No |

### Stored Procedures / Shared Modules (Maintainability)

CURRENCY_SQL.CONVERT_VALUE – converts values based on a given source and target currencies.

### Program Flow



### Function Level Description

main()

The standard Retek main function that calls init(), process(), and final().

init()

This function initializes the restart/recovery logic. It also retrieves the system of record pricing indicator the pricing level from the system tables.

process()

This is the main control function of the batch program. The function retrieves new warehouses, virtual warehouses and/or internal finishers from the WH_ADD table. Each record processed will have corresponding records inserted to the price_zone and price_zone_group_store tables.

Every record processed will be deleted from the WH_ADD table.

insert_price_zone()

This function accepts data as parameters to the function for the insert to the price_zone table.

insert_price_zone_group_store()

This function accepts data as parameters to the function for the insert to the price_zone_group_store table.

delete_wh_add_record()

This function deletes wh_add table record based on the rowid passed to the function as a parameter.

final()

This function calls retek_close() to update restart recovery tables and performs commit.

### Input Specifications

Driving Cursor:

```
EXEC SQL DECLARE c_add_wh_driver CURSOR for
    SELECT wa.wh,
                   wa.wh_currency,
                   wh.wh_name,
                   rowidtochar(wa.rowid)
      FROM wh_add wa,
                  wh
     WHERE wa.wh = wh.wh (+)
ORDER BY wa.wh;
```

### Output Specifications

'Table-To-Table'

Selected and deleted records from:

- WH_ADD

Inserted records into:

- PRICE_ZONE
- PRICE_ZONE_GROUP_STORE

### Scheduling Considerations

Processing Cycle: Daily, Ad Hoc Phase

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Locking Strategy

N/A

### Restart/Recovery

None.

### Performance Considerations

N/A

### Security Considerations

N/A