Oracle® Retail Merchandising System Operations Guide Addendum

Release 11.0.9

September 2006

Contents

Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's 'behind-the-scenes' processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise

This Operations Guide Addendum should be used in conjunction with previously released Oracle Retail Merchandising System 11.x documentation.

Audience

Anyone with an interest in developing a deeper understanding of the underlying processes and architecture supporting Oracle Retail Merchandising System functionality will find valuable information in this guide. There are three audiences in general for whom this guide is written:

- Business analysts looking for information about processes and interfaces to validate the support for business scenarios within and other systems across the enterprise.
- System analysts and system operations personnel:
  - Who are looking for information about Oracle Retail Merchandising System's processes internally or in relation to the systems across the enterprise.
  - Who operate Oracle Retail Merchandising System regularly.
- Integrators and implementation staff with overall responsibility for implementing Oracle Retail Merchandising System.

Related Documents

You can find more information about this product in these resources:

- Oracle Retail Merchandising System Installation Guide
- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Data Model
- Oracle Retail Merchandising System Batch Schedule

Customer Support

- https://metalink.oracle.com

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.

1 Introduction

The information in this document reflects modifications and updates to the latest Oracle Retail Merchandising System Operations Guide. Using this document in conjunction with that guide provides retailers with a complete overview of the application.

Note: RSL for RMS is not new functionality. However, this addendum is the first time RSL for RMS documentation has been published for RMS 11.x.

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the *Oracle Retail Merchandising System 11.0.9 Release Notes*.

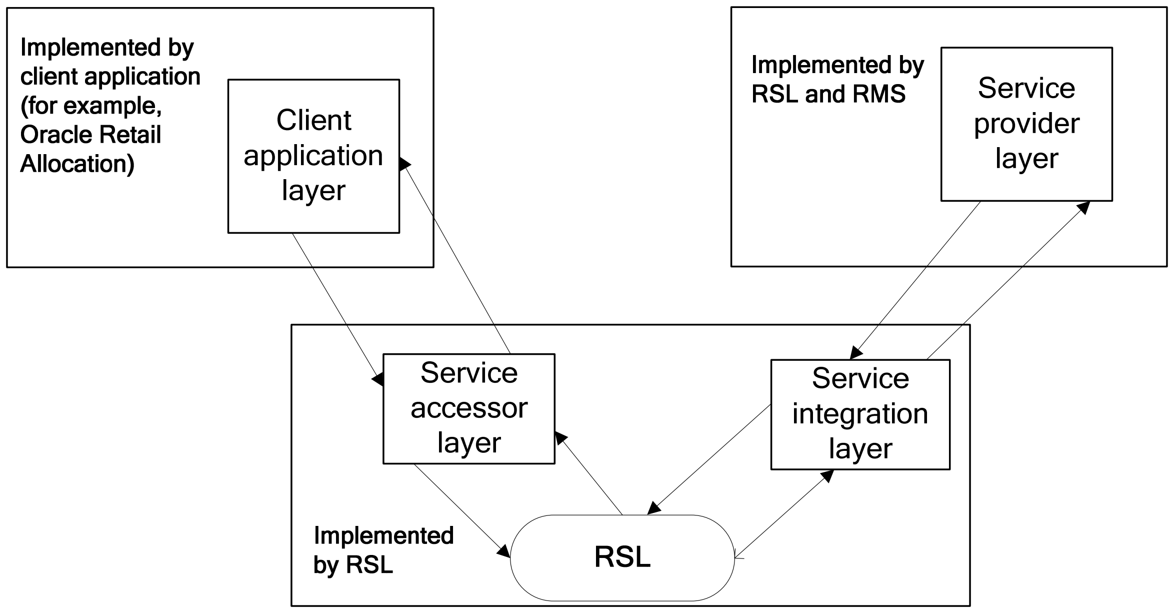
2 RSL for RMS

RMS and the Oracle Retail Service Layer (RSL)

RSL is a framework that allows Oracle Retail applications to expose APIs to other Oracle Retail applications. As shown in the diagram below, in RSL terms, there is a ‘client application layer’ and a ‘service provider layer’. RMS includes the ‘service provider layer’ that owns the business logic.

The RMS implementation of RSL exposes a *synchronous* method to communicate with other applications (RIB-facilitated processing is asynchronous). All RSL services are contained within an interface offered by a Stateless Session Bean (SSB). To a client application, each service appears to be merely a method call.

For information about RSL-related configuration within the RMS application, see RSL documentation.


Implemented by client application (for example, Oracle Retail Allocation)
Client application layer
Implemented by RSL and RMS
Service provider layer
Service accessor layer
Service integration layer
Implemented by RSL
RSL

Client application and service provider processing through RSL

Functional Description of the Packages Used by RSL

The table below offers a functional description of the packages used by RSL.

Package	Description
RMSSVC_XLOCPOTSF	Through RSL, this call to RMS allows Oracle Retail Allocation to create/update a purchase order in RMS from a ‘what if” allocation.

3 Custom Post Processing

RMS has an optional method of handling unwanded cartons for customer post processing. This only applies to stock order receiving. An unwanded carton occurs when a carton was not scanned when the stock order was shipped, but is scanned at the time of the receipt. These cartons do not contain any shipment records in RMS.

Since the carton contains items that did not go through the appropriate transfer out procedure, the inventory for those items will not be accurate. As a result, the message which contains the unwanded (unscanned) carton is rejected by RMS to the RIB error hospital at the time of receiving. RMS will then publish to the warehouse management system via the RIB of the unwanded cartons in the RcptAdjustDesc message. The warehouse management system will then send RMS a shipment message containing the appropriate BOL and the carton ID. RMS will process the message and create or update the shipment records. The next time RMS tries to process the rejected receipt message with the unwanded carton, RMS will be able to process it.

The client’s warehouse management system must be able to support the processing of the RcptAdjustDesc message above in order for this functionality of unwanded carton to work successfully.

4 RMS Internationalization and Localization

The technical infrastructure of RMS supports German localization. Please note that this does not have any functional impact on the RMS product, nor does it allow the user to switch to different languages, as the user interface does not support this capability.

Key RMS Tables

Several tables hold user-interface displayed text.

If the retailer creates a new form, a new menu, or a new object on a form, then the retailer will need to populate these tables with the corresponding information. If the retailer customizes the information in any of the tables FORM_ELEMENTS, FORM_ELEMENTS_LANGS, MENU_ELEMENTS, or MENU_ELEMENTS_LANGS, the BASE_IND field in customized records must contain ‘N’. Any record with BASE_IND=N will be preserved in a temp table during future patches. It is the responsibility of the retailer to move the customized data bank from the temp table to the primary table (e.g., FORM_ELEMENTS_LANGS) after applying patches.

FORM_ELEMENTS

This table is used for screen display and holds the master list of items for all forms whose labels/prompts are translated. This information will always be in English. The BASE_IND=Y means that the item is part of the base Oracle Retail code set. BASE_IND =N indicates that the item was added as part of retailer customization. Anything with the BASE_IND =N will be preserved at upgrade time on the FORM_ELEMENTS_TEMP, but the retailer is responsible for moving the data back to FORM_ELEMENTS.

FORM_ELEMENTS_LANGS

This table is used for screen display. This table holds translated values for labels/prompts on forms. This information will be in a language that is defined on the lang column of the user_attrib table. All users see data from this table, as the retailer may customize the text of a given field. The access key for a button is defined by filling in the DEFAULT_ACCESS_KEY field. At runtime, that character will be marked in the string, and function as the access key. Any time the retailer changes the DEFAULT_LABEL_PROMPT or DEFAULT_ACCESS_KEY, the BASE_IND should be updated to N because it is not part of the base language translations provided by Oracle Retail. Anything with the BASE_IND=N will be preserved at upgrade time on the FORM_ELEMENTS_LANGS_TEMP, but the retailer is responsible for moving the data back to FORM_ELEMENTS_LANGS.

MENU_ELEMENTS

This table is used for screen display. This table holds the master list for all menus whose items are translated. This information will always be in English. The access key for a menu option is defined by using the ampersand (&) before the character that is the access key in the default description. The BASE_IND=Y means that the item is part of the base Oracle Retail code set. BASE_IND=N indicates that the item was added as part of retailer customization. Anything with the BASE_IND=N will be preserved at upgrade time on the MENU_ELEMENTS_TEMP, but the retailer is responsible for moving the data back to MENU_ELEMENTS.

MENU_ELEMENTS_LANGS

This table is used for screen display. This table holds the values for all menus whose items are translated. This information will be in a language that is defined on the lang table. Even English language users see data from this table, as the retailer may customize the text of a given menu option. Any time the retailer changes the LANG_LABEL, the BASE_IND should be updated to N because it is not part of the base language translations provided by Oracle Retail. Anything with the BASE_IND=N will be preserved at upgrade time on the MENU_ELEMENTS_LANGS_TEMP, but the retailer is responsible form moving the data back to MENU_ELEMENTS_LANGS.

FORM_MENU_LINK

This table is used for screen display. This table holds the intersection of form and menu files, mapping each form to the menu that it displays.

5
Batch Designs

POS Upload [posupld] Batch Design

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

The combinations of system variables and the resulting scenarios				
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), M (Monthly), or D (Daily). This controls the date used for the 1 to 1 matching which is vdate, eow_date, eom_date or vdate 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), M (Monthly), or D (Daily). This controls the date used for the 1 to 1 matching which is vdate, eow_date, eom_date, or vdate 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
	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

Record Name	Field Name	Field Type	Default Value	Description
	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 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
	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

Record Name	Field Name	Field Type	Default Value	Description
	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
	Transaction Type	Char(1)	‘S’ – sales ‘R’ - return	Transaction type code to specify whether transaction is a sale or a return
	Drop Shipment Indicator	Char(1)	'Y' '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.

Record Name	Field Name	Field Type	Default Value	Description
	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 ‘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)	**‘N’**	Indicates if item is a catchweight item. Valid values are ‘Y’ or ‘**N’**
	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 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.

Record Name	Field Name	Field Type	Default Value	Description
	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
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.

Record Name	Field Name	Field Type	Default Value	Description
	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 available.
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.

RPM Moving Average [rpmmovavg] Batch Design

Design Overview

This batch module takes the number of units sold from TRAN_DATA table for all items designated for a particular store within a specified store/day, and maintains a smoothed average in the IF_RPM_SMOOTHED_AVG table.

Only the sales, which have a sales type of regular, are included. If the item is on promotion or clearance, then no updating is required. The units under normal sales will be considered as unadjusted units and will be taken for smoothed average. The threshold percent will be maintained at the department level. This percent will be compared to the existing smoothed average value and used to limit the upper and lower boundaries for regular sales received. If the unadjusted units amount is outside of the boundaries, then the appropriate boundary amount will be substituted and become the adjusted units amount. If no threshold percent is defined for the department, it will be defaulted to 50%.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Phase 3 (Daily)
Scheduling Considerations	This program has to be run after all of the data for the store/day has been uploaded into RMS and before the days data is removed from the TRAN_DATA table. Run before SALSTAGE.PC.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

The logical unit of work for this program is set at store/item level.

Restart ability is implied based on item and store combination. Records will be committed to the database when commit_max_ctr defined in the RESTART_CONTROL table is reached.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_LOC	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
LOCATION_CLOSED	Yes	No	No	No
TRAN_DATA	Yes	No	No	No
DEPS	Yes	No	No	No
IF_RPM_SMOOTHED_AVG	Yes	Yes	Yes	No

Program Flow

N/A

I/O Specification

N/A

Outstanding Issues

N/A

Sales Daily (saldly) Batch Design

Functional Area

Stock Ledger

Module Affected

SALDLY.PC

Design Overview

This module rolls up transaction data on IF_TRAN_DATA to the dept/class/subclass/location/transaction date/currency level.

The rolled-up transactions are used to update applicable records on DAILY_DATA based on the transaction type. A new record is inserted if no record exists for the transaction.

If open stock count exists for the closed month and there are back-posted sales transactions then the program will rolls up transaction data on IF_TRAN_DATA to the dept/class/subclass/location/transaction date into a new table DAILY_DATA_BACKPOST.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 3 (daily)
Scheduling Considerations	N/A
Pre-Processing	Run SALSTAGE to move records from TRAN_DATA to IF_TRAN_DATA.
Post-Processing	N/A
Threading Scheme	Threaded by department

Restart/Recovery

The logical unit of work is department/class/subclass. This batch program is multithreaded using the v_restart_dept view.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
PERIOD	Yes	No	No	No
SYSTEM_VARIABLES	Yes	No	No	No
IF_TRAN_DATA	Yes	No	No	No
DAILY_DATA	Yes	Yes	Yes	No
DAILY_DATA_TEMP	No	Yes	No	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
PARTNER	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
DAILY_DATA_BACKPOST	No	Yes	No	No

I/O Specification

N/A

Stock Count Shrinkage Update (stkdly) Batch Design

Functional Area

Stock Ledger

Module Affected

STKDLY.PC

Design Overview

This program processes the ‘Unit & Dollar’ type of stock count that the user has submitted for processing for the stock ledger. The main functions are to calculate actual shrinkage amount that will be used to correct the book stock value on the stock ledger and to calculate a shrinkage rate. A system option indicator (CLOSE_MTH_WITH_OPN_CNT_IND) is used to determine whether or not the current fiscal month is allowed to be closed while containing an open Unit and Dollar stock count.

If the indicator is No (i.e., fiscal month may not be closed with existing open Unit and Dollar stock counts), the program raises a fatal error if open stock counts are found within the current fiscal month. If no open stock counts are found within the current fiscal month, the program calculates the book stock value for the current months scheduled stock counts. It then compares the book stock value to the actual stock value as reported on the stock count. These values and their difference are used to update month data records. Values such as shrinkage, book stock, and actual stock are modified as a consequence. Week data are similarly updated; since it is always the current month being processed, current half-year data records for inter-stock-take and sales can be updated with these values as well.

If the indicator is Yes and open stock count exists for the closed month then the program gets the data from daily_data table and also from daily_data_backpost table for the back-posted sales transactions. It then calculates and compares the book stock value to the actual stock value as reported on the stock count. These values and their difference are used to update month data records. Values such as shrinkage, book stock, and actual stock are modified as a consequence. Week data are similarly updated.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 3 (daily)
Scheduling Considerations	Run before SALWEEK.PC and SALMTH.PC
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	Threaded by department

Restart/Recovery

This batch program is multithreaded using the v_restart_dept view. The logical unit of work for this program is dept/class/location/loc_type.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
SYSTEM_VARIABLES	Yes	No	No	No
STAKE_PROD_LOC	Yes	No	Yes	No
STAKE_HEAD	Yes	No	No	No
DEPS	Yes	No	No	No
HALF_DATA_BUDGET	Yes	No	No	No
DAILY_DATA	Yes	No	No	No
WEEK_DATA	No	No	Yes	No
MONTH_DATA	Yes	No	Yes	No
HALF_DATA	No	No	Yes	No
DAILY_DATA_TEMP	No	Yes	No	No
DAILY_DATA_BACKPOST	Yes	No	No	No

I/O Specification

N/A

Tampered Carton (tamperctn) Batch Design

Functional Area

Store Receiving

Module Affected

TAMPERCTN.PC

Design Overview

The Tampered Carton module (tamperctn.pc) is a batch program that matches the tampered carton information in the staging table to existing shipment records. If the shipment records contain a prepack, then the batch program uses the prepack components to compare with the items on the staging table.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	AD-HOC
Scheduling Considerations	This batch program should only run when the store_pack_comp_rcv_ind system option is set to ‘Y”.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

N/A

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
SYSTEM_OPTIONS	Yes	No	No	No
DUMMY_CARTON_STAGE	Yes	No	No	Yes
PERIOD	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No
SHIPMENT	Yes	No	No	No
SHIPSKU	Yes	No	No	No
SHIPSKU_TEMP	Yes	Yes	No	Yes
PACKITEM	Yes	No	No	No

I/O Specification

N/A