Retek® Merchandising System™ 9.0.18

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

Customer Support

Customer Support hours

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

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

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

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

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

POS Upload [posupld] 1

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 sku/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
- 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 SKU/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_sales_export_date on the item store tables 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.

Scheduling Constraints

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 SKU sales transaction at a given store location. The location type will be inferred as a store type and the item can be passed as a SKU or UPC 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), 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. 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.

Program Flow

N/A

Shared Modules

validate_all_numeric: intrface library function.

validate_all_numeric_signed: intrface library function.

valid_date: intrface library function.

ORDER_ATTRIB_SQL.DELIVERY_MONTH: called from consignment_data(), returns order delivery month into the :invoices variable.

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

CURRENCY_SQL.CONVERT: returns the converted monetary amount from Currency to currency.

NEW_STAPLE_LOC: called from win_check(), creates a new staple SKU if one doesn’t already exist for the SKU/location passed in.

NEW_FASHION_LOC: called from rag_check(), creates a new fashion SKU if one doesn’t already exist for the SKU/location passed in.

NEW_PACK_LOC: called from pack_check(), creates a new packitem if one doesn’t already exist for the packitem/location passed in.

UPDATE_SNAPSHOT_SQL.EXECUTE: called from update_snapshot(), updates the stake_sku_loc table for late transactions.

NEXT_ORDER_NO: called from consignment_data(), returns a randomly generated order number.

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

Function Level Description

Declarations:

declare input structures: file header (only date and type) & detail (all fields)

init()

initialize restart recovery

open input file (posupld)

- file should be specified as input parameter to program

fetch system variables

Retrieve all valid promotion types

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), store, transaction_date

```
if fresh start (l_file_start = 0)
    read file header record (get_record)
    if (record type <> 'FHEAD') Fatal Error
    validate file type = 'POSU'
else fseek to l_file_start location
validate location and date are valid
```

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.

Call the sku_process() function to perform item specific processing. Once all records have been processed, call posting_and_restart to commit the final records processed since the last commit and exit the function.

sku_process()

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, the transaction will be written 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. Since we do not create price_hist records of type 9 (promotional retail change) when the system_options.multi_prom_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 system_options.multi_prom_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_sale field on the packstore table to the transaction date and the packstore_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 SKU to the packitem to the component SKU's regular retail found on the price_hist table. The retail contribution for a component SKU is calculated by taking the component SKU's unit retail from price_hist, divided by the total retail of all component SKUs in the packitem, and multiplying the packitem's unit retail. So if the retail contribution of a component SKU within packitem A is $10, and the same component SKU's price_hist record has a retail of $14, and there is only one packitem sold, and this component SKU 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 SKU'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), call update_snapshot() to update the stake_sku_loc table and write_edi_sales() to insert or update the edi_daily_sales table. If the transaction is current, insert or update the edi_daily_sales table (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 item's 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.

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 system_options.multi_prom_ind = 'N' and the item is on promotion, or if the system_options.multi_prom_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 sku 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 sku_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, system_options.multi_prom_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 system_options.multi_prom_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.

process_sales_and_returns()

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 will be called to write a tran_data record with a tran_type 1 (always written), a tran_type 2 (if the system_options.stkldgr_vat_incl_retl_ind = Y), 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.

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.

validate_FHEAD()

Do standard string validations on input fields. This includes NULL padding fields, left shifting 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.

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 UPC exists. If the upc_sup field is blank, set it to 00000.

If an UPC is passed in from the input file, retrieve the SKU for the UPC and UPC supplement. Once the item is a SKU, retrieve the system 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.

If the sales audit indicator is ‘Y’ on system_options, the item will be a SKU and the dept, class, subclass, and system_ind will be included in the file.

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.

Get_upc_wt()

Assumption: Given the structure of the var_upc_ean table, it is only possible to have 100 different records on this table. This function uses this assumption when allocating memory to hold all the records on the var_upc_ean table. The logic will be required enhancements to accommodate a larger number of different variable weight upc types if desired.

Assumption: Non-variable weight UPC's can't start with a prefix that exists on the var_upc_ean table.

Assumption: All values passed in through variable weight UPC's will be considered to be in the standard unit of measure.

This function will extract an amount from a variable weight UPC. The first time this function is called it will build an array containing the var_upc_ean table. It will then proceed to search that array for a record that has a prefix matching the first two values in the UPC from the file. If it finds a matching record, it will use the information in the var_upc_ean array to strip out the weight from the UPC. It will then replace the characters of the UPC that held the weight with zeros so the sku can be retrieved from the UPC.

Example:

The UPC 2712345000000 is stored in RMS as a variable weight UPC. The prefix of the UPC is 27. The item identifier is 12345. The next five digits are the variable weight portion of the UPC. And finally the last character is a check digit for data transmission use.

The UPC is sent down to the stores and thus the scales with a variable weight indicator. This indicator lets the scales know to insert the weight sold, of the item, into the variable weight portion of the UPC upon a sale (with 3 implied decimal places). Let's say 7.7 pounds of our UPC were sold. The UPC would be uploaded from the POS file as 2712345077001. This function would strip out the 7.7 and assign it to the total qty sold variable. It would then replace the variable weight portion and the check digit of the UPC with zeros: 2712345000000. The UPC with the weight and check digit stripped out then will be used with the upc_ean table to get the sku.

validate_TDETL

Assumption: Currently posupld.pc cannot interface with sales audit. This is due to the variable weight UPC logic. Sales audit currently doesn't recognize variable weight UPC's, and thus cannot process them. The code designed to interface with Sales Audit is commented out. It should be uncommented when sales audit is updated to deal with variable weight UPC's.

Assumption: Variable weight UPC's need to be sent in at the transaction level, not the rolled up level that posupld usually receives. This is due to the fact that TDETL lines need to source their qty's from the UPC and not from the specified qty input fields.

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.

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 a promotion number is passed in, validate it by checking the promhead table and set the promotional indicator 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.

New_staple_loc

This function creates a new store sku relationship for staple items.

Win_store_cursors

This function checks the win_store for the sku / store combination. It is called by the win_check function.

win_check

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

New_fashion_loc

This function creates a new store sku relationship for fashion items. It is called by rag_check.

rag_store_cursors

This function checks the win_store for the sku / store combination. It is called by the win_check function.

rag_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.

New_pack_loc

This function creates a new store sku relationship for fashion items. It is called by pack_check.

pack_check

This function verifies the pack item/location relationship exists and retrieves the component SKUs for the packitem. It is only called when the item being processed is a packitem. The component SKU, system indicator, department, class, subclass, cost, retail, price_hist retail, and component SKU 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 SKUs price ratios are also calculated. This indicates the retail contribution the component SKU 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 SKUs for the packitem. Below is an example of how this ratio is calculated:

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

get_unit_retail

This function retrieves the unit retail from price_hist for the item/location being processed. If the item being processed is not a component SKU to a packitem that is currently being processed and the transaction is a sale, call get_all_price. If that is not found, call get_base_price. If the price_hist record fetched is a 9 (promotional retail change), the item being processed is a component SKU to a packitem currently being processed, or the transaction is a return, call get_reg_price. (these are base retail changes). If a tran_code of 8 is returned, the item is on clearance.

Get_all_price

This function will get the retail from price_hist for tran_types in (0, 4, 8, 9, 11). It is called by get_unit_retail.

Get_reg_price

This function will get the retail from price_hist for tran_types in (0, 4, 8, 11). It is called by get_unit_retail.

Get_base_price

This function will get the retail from price_hist for tran_types of 0. It is called by get_unit_retail.

process_packitems

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

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_exist and daily_sales_insert_update.

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()

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.

Daily_sales_exist()

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

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 an update array for the daily_sales_discount table.

write_off_retail_markdown()

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_markdown()

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.

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.

vat_convert()

This function will either add or remove vat from a retail value.

process_win()

Update the stock on hand on the win_store table for Sales and Returns unless the item is on consignment. Also, update the win_store_hist table for Sale transactions. Do not update for returns.

process_rag()

Update the stock on hand on the rag_skus_st table for Sales and Returns unless the item is on consignment. Also, update the rag_skus_st_hist table for Sale transactions. Do not update for returns.

process_pack()

Update the stock on hand on the packstore table for Sales and Returns. Also, update the rag_skus_st_hist table for Sale transactions. Do not update for returns.

write_tran_data()

Writes a record to the tran_data insert array.

Write_edi_sales()

Writes or updates a record to the edi_daily_sales table for both current and late transactions.

update_snapshot()

Calls the UPDATE_SNAPSHOT_SQL.EXECUTE function to update the stake_sku_loc table for late transactions.

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.

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

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.

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’.

fill_packitem_array()

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

write_sku_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.

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

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 sales_audit_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 sales_audit_ind is 'N' these values will not be used and retrieved from the system.

When the system_options field sales_audit_ind is 'Y' the following THEAD fields will be populated and already validated: System_ind, Dept, Class, and Subclass. When the sales_audit_ind is 'N' these values will not be used and retrieved from the system. When the sales_audit_ind is on, all items are assumed to be SKUs.

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(4)	specified by external system	Store or warehouse 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. Required if the sales audit indicator is 'Y' on system_options.
	Vat region	Number(4)		Vat region the given location is in. Required if the sales audit indicator is 'Y' on system_options.
	Currency code	Char(3)		Currency of the given location. Required if the sales audit indicator is 'Y' on system_options.
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Required if the sales audit indicator is 'Y' on system_options.
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.
	Business Date	Char(14)	business date to process	business date of transactions

Record Name	Field Name	Field Type	Default Value	Description
	Item Type	Char(3)	UPC SKU	item type will be represented as an UPC, an SKU
	Item Value	Char(13)	item identifier	the id number of a SKU or UPC
	Supplement	Char(5)	supplemental identifier	used to further specify the id of an UPC item, or the pre-pack id reference
	System_ind	Char(1)	'S'- staple sku 'f'- fashion sku 'P'- pack item	The type of item sold or returned. Required if the sales audit indicator is 'Y' on system_options.
	Dept	Number(4)	Item's dept	Dept of item sold or returned. Required if the sales audit indicator is 'Y' on system_options.
	Class	Number(4)	Item's class	Class of item sold or returned. Required if the sales audit indicator is 'Y' on system_options.
	Subclass	Number(4)	Item's subclass	Subclass of item sold or returned. Required if the sales audit indicator is 'Y' on system_options.

Record Name	Field Name	Field Type	Default Value	Description
	Wastage Type	Char(6)	Item’s wastage type	Wastage type of item sold or returned. Required if the sales audit indicator is ‘Y’ on system_options.
	Wastage Percent	Number(12)	Item’s wastage percent	Wastage percent of item sold or returned. Required if the sales audit indicator is ‘Y’ on system_options.
	Transaction Type	Char(1)	‘S’ – sales ‘R’ - return	Transaction type code to specify whether transaction is a sale or a return
	Total Sales Quantity	Number(12)		Number of units sold at a particular location with 4 implied decimal places.
	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

Record Name	Field Name	Field Type	Default Value	Description
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(4)	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.
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

Record Name	Field Name	Field Type	Default Value	Description
	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 UPC’s need to be sent in at the transaction level, not the rolled up level that posupld usually receives. This is due to the fact that TDETL lines need to source their qty’s from the UPC and not from the specified qty input fields.

Assumption: Given the structure of the var_upc_ean table, it is only possible to have 100 different records on this table. This function uses this assumption when allocating memory to hold all the records on the var_upc_ean table. The logic will be required enhancements to accommodate a larger number of different variable weight upc types if desired.

Assumption: Non-variable weight UPC’s can’t start with a prefix that exists on the var_upc_ean table.

Assumption: All values passed in through variable weight UPC’s will be considered to be in the standard unit of measure.