Retek

Retek Merchandising System 9.0.5

Addendum to Operations Guide

rms-905-og-addendum

Retek Merchandising System™

The software described in this documentation is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Information in this documentation is subject to change without notice.

Trademarks

Policy on Retek End User Documentation

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

Printed in the United States of America.

Customer Support

Customer Support hours:

8 AM to 5 PM Central Standard Time (GMT-6), Monday through Friday, excluding Retek company holidays (in 2001: Jan. 1, May 28, July 4, Sept. 3, Nov. 22, Nov. 23, Dec. 24, and Dec. 25).

Customer Support emergency hours:

24 hours a day, 7 days a week.

Contact Method	Contact Information
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: + 1 612-630-5800
Fax	(+1) 612-630-5710
E-mail	support@retek.com
Internet	www.retek.com/support Retek's secure client Web site to update and view issues
Mail	Retek Customer Support Midwest Plaza 801 Nicollet Mall Suite 1100 Minneapolis, MN 55402

When contacting Customer Support:

- Always fill out an Issue Report Form before submitting issues to Retek (request forms from Customer Support if necessary).
- Provide a completely updated Customer Profile.
- Have a single resource per product responsible for coordination and screening of Retek issues.
- Respond to our requests for additional information in a timely manner.
- Use Retek Online Customer Support (ROCS) to submit and update your issues.
- Have a test system in place running base Retek code.

Contents

Chapter 1 – Introduction

This addendum to the Retek Merchandising System (RMS) 9.0 Operations Guide contains updates to the following batch designs:

- Batch Design - ordrev.doc
- Batch Design - allocupd.doc
- tsfdnld.doc
- dealcalc.doc
- rtvupld.doc
- lifrtvup.doc

Refer to the following chapters for that information, which supercedes all comparable information in the RMS 9.0 Operations Guide. Each chapter contains a subsection indicating what specific modifications have been made.

Chapter 2 – Purchase Order Information Written to Order History Tables [ordrev]

Modification

This program was modified by adding fields to bring shipping and billing addresses to five lines in the stock order file.

Design overview

Ordrev will write versions of approved order to order revision history tables. When orders are approved or when approved orders are modified, this program selects order numbers from the rev_orders table and writes current order information to the order/allocation revision tables. After the new version has been written to the order revision tables, all records will be deleted from the rev_orders table for that order_no.

This program processes order changes made by the client that may need to be sent to the vendor. The order changes should always be referred to as 'versions' and kept clearly distinct from order 'revisions' which are vendor changes uploaded via the ediupack program.

This program also allows Nautilus and Retek to interface, by sending the warehouse PO and allocation (ie. pre distribution) information to prepare the warehouse for incoming orders. The program will create two flat files, PO header and PO detail, based on approved orders found on the rev_orders table. The program will also create Pre Distribution Header and Pre Distribution Detail flat files, which will enable the warehouse to perform cross docking activities.

The last file produced by the ordrev batch program is a component ticketing file that holds retail and ticketing information for non sellable pack items. This file allows the warehouse to correctly ticket the components of the pack item, before distributing the items to the stores.

If an order is not in approved status at the time the batch program runs, then none of the above processing will occur. The record will stay on the rev_orders table until the PO is approved or deleted.

Scheduling constraints

Processing Cycle: After rplprg & before edidlord, and Ad Hoc. This program must be run, if interfacing with Nautilus

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart recovery

Restartabilty will be implied, because the records that are selected from this table will be deleted before the commit. Restart library functions will still be included to ensure that rollback segments are not exceeded (by committing at intervals) and to perform basic record keeping functionality.

```
SELECT ro.action_type,
           ro.order_no,
           ro.alloc_no,
           ro.location,
           ro.sku,
           ro.hdr_dtl_ind,
           oh.pre_mark_ind,
           ro.rowid
      FROM rev_orders ro,
           ordhead oh
     WHERE ro.order_no = oh.order_no
       AND oh.status = 'A'
       AND MOD(ro.order_no, :oi_restart_num_threads) + 1
= :oi_restart_thread_val
       AND ro.order_no > NVL(:ora_restart_order_no, -
9999)
  ORDER BY ro.order_no;
```

Program flow

N/A

Shared modules

PRICING_ATTRIB_SQL.GET_RETAIL(): get the unit retail from item_zone_pricing table for a SKU/store.

PROMOTION_ATTRIB_SQL.EVENT_DESC(): get the event's description

Function level description

Init()

Initialization of the restart Retek recovery process will be performed here.

Get system date.

Open output files. There will be a maximum of 4 files (ie. one header and detail for PO download and one header and detail for Pre-distribution download)

Write FHEAD to all files.

Call Init_buffers().

Process()

All orders that need to have order version records will be processed.

If the order number changes, then perform the following logic.

- The order number will be used to populate the revision history tables. The get_rev_no() function is called to determine the version number for the insert into the revision history tables.
- Check if order is customer order. If order is customer order set flag to 1 , else set to 0(for the customer order no allocation information will be download to the RLS logistic).
- If version 1was just inserted (ie. order was just approved for the first time, no previous versions existed), then
  - Call write_new_po function to write newly created orders and associated allocations to the po header, po detail, pre distribution header, and pre distribution detail files.
- Else
  - Call write_existing_po function to write the changed order information to the flat files. Some or all of the flat files may be written in this circumstance depending upon what information has changed since the order was last sent down to Nautilus.
- End if;
- The insert_header() function will be called from here to insert header level information, the insert_sku() function will insert order sku information, the insert_loc() will insert order sku/location information, and the insert_alloc() will insert order allocation information if the order's pre-mark indicator was set. This indicator will indicate whether cross-docked allocation information will be sent to the supplier along with the order. When all of the version information has been inserted into the revision history tables, all of the records with that order number should be deleted from the revord table by the delete_revord() function.
- If system_options.financial_ap equals 'P', then call ins_revord () to insert into the fif_ordhead table.

Else /* the order number remains the same */

- If order is not customer order. Call write_alloc_only().

End if;

Get_rev_no()

It is necessary to get the last version number that was written to the order revisions tables. The maximum version number is selected from the header revision table and then incremented by 1 to get the version number that will be inserted during processing. If no record exists in the order header revision history table, then the order is new and a version number of 1 is used.

Insert_header()

The current information on the order header table will be inserted into the header revision history table with the new version number

Insert_sku()

The current information on the order SKU table is inserted into the order SKU revision history table with the new version number

Insert_loc()

The current information on the order SKU/location table is inserted into the order SKU/location revision history table with the new version number.

Insert_alloc()

The ship-to warehouse on the allocation header table and the allocation information and quantity information from the allocation detail table is written to the allocation revision history table with the new version number.

Ins_revord()

Insert into the fif_ordhead table.

Write_new_po()

This function will write FDETL records to the appropriate PO and pre distribution output files.

Order information is retrieved from the ordhead and ordloc tables to populate the PO header and PO detail files. A record will be written to the PO download header and detail file for only orders with a warehouse destination. The warehouse number will be stored in the Location (DC) field on the file. If the order is going to other locations, such as stores, then do not write a record to the files. There will be one header for each order/wh location retrieved.

Check customer order flag. If it is not customer order, open a “for loop” to retrieve the allocation information for an order.

Write pre-distribution header and detail with action type = 'A'dd for the warehouse/allocation/sku/order_no. There will be one header for every alloc_no retrieved and a detail record for each to_location for that allocation. In other words, the first allocation number will be written to the pre-distribution header record. Write the pre-distribution detail records, until that allocation number changes. When the allocation number changes, then write a pre-distribution header record. The warehouse (from_loc) will be stored in the Location (DC) field on the file. Call promotion_attrib_sql.get_event_desc package for the event's description. Also, get the correct retail (pricing_attrib_sql.get_retail package) and ticketing information for the predistribution detail file. In the for loop, if the allocation location is a store, call comp_tckt () function to write the component ticketing file.

Write_existing_po()

Open a "for loop" to retrieve ordhead and ordloc fields for comparison. The comparison will be completed for each warehouse location the order is destined. In the for loop, compare ordhead/ordloc with previous version on ordhead_rev/ordloc_rev. If there are any changes to the Nautilus required fields, then write PO download header and/or detail records. This process only needs to be done for orders going to warehouse locations.

Fetch the header information from ordhead and ordhead_rev. Compare each field (ie. ordhead.buyer = ohr.buyer). If the fields do not equal, then set an indicator, which will indicate that the ordhead records have been modified and an action_type = 'M' will need to be sent down in the PO header file.

For the order number retrieved in the above cursor loop through the ordloc warehouse records. First, check the header indicator. If the ordhead record has changed, then a PO header record needs to be written for each warehouse on the order. For example, one PO (#123456) has been created to replenish the stock in warehouse 1, 2, and 3. The PO header download file produced by the ordrev.pc program will have 3 separate records. The first FDETL will have a location (DC) = 1 for PO #123456, the second record will have a location (DC) = 2 for PO #123456, and the third record will have a location (DC) = 3 for PO#123456. After the ordhead indicator check, compare the ordloc and the ordloc_rev fields. If one of the fields differ, then write a PO detail record for the warehouse/order_no. Once all warehouse locations are processed in that order, go fetch the next order.

- If ordloc.qty_ordered != 0, then action type = 'M'
- If ordloc.qty_ordered = 0, then action type = 'D'

Check customer order flag. If it is not customer order. Call write_alloc_only();

Write_alloc_only()

This function will write FDETL records to the appropriate pre-distribution output files.

If alloc_no is not NULL, then (alloc_no was retrieved from the main driving cursor on the rev_orders table)

- If location is NULL and action type = 'A' then
  - Write pre-distribution download header and detail with action type 'A'. If the action type = 'A', then loop through all of the "to locations" of the allocation on alloc_detail table. A detail record will need to be written for each alloc_detail location.
  - In the for loop, if the allocation location is a store, then call the comp_tckt() function.
- Elsif location is not NULL and action type = 'D' and hdr_dtl_ind = 'H'
  - Write pre-distribution header with action type = 'D'. The location field retrieved by the driving cursor will contain the from warehouse location (ie. alloc_header.wh) and should be used to populate the Location (DC) field on the output file.
- Elsif location is not NULL and action type = 'D' and hdr_dtl_ind = 'D'
  - Write pre-distribution detail with action type = 'D'. The location field on the rev_orders table will contain the to store/warehouse location (ie. alloc_detail.store or wh) and should be used to populate the destination id on the output file.
- Else /* location is not NULL and action type = 'A' or 'M' */
  - Write pre-distribution download detail with 'A', 'M', depending on the action type retrieved from the main cursor (ie. rev_orders). Get the detail file's information (from_loc, to_loc, qty) by selecting from the alloc_detail/alloc_header table for the alloc_no and location found in the main driving cursor. A detail record should be written for the location that was retrieved from the rev_orders table.
  - If the action type = 'A' and the allocation location is a store, then call the comp_tckt() function.
- End if;

End if;

Comp_tckt()

If the SKU on the allocation is a non sellable pack item going to a store location, then write all of the component SKUs, retail price, and ticket information to the component_ticketing file.

Del_revord()

Multiple order versions could exist on the revord table for the same order. This could happen if the batch program had not been run since the last time the order was modified. Since the processing has written the current order value to the revision history tables, all records with that order number must be deleted from the revord table to prevent double processing

I/O specification

The five output files should be specified at the command line when running the ordrev.pc program.

Order Header file

Record Name	Record	Default value	Field type	Description
File Header	Detail file identifier	FHEAD	Char(5)	Identifies the header line
	line number	Incremented internally	Number(10)	sequential line number
	Program descriptor	POHD	Char(5)	Identifies the program
	Create date	YYYYMMDDHH24MISS	Char(14)	File create date
File detail	File record descriptor	FDETL	Char(5)	Identifies the detail line
	Line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	Ordloc.location (wh only)	Number(4)	Location of item that was ordered
	Transaction day date/time	sysdate	Datetime(12)	system date
	Po number	'P' + ordhead.order_no	Char(9)	Unique identifier of the purchase order, prefixed with 'P'
	Vendor number	Ordhead.supplier	Number(7)	Supplier number of the order
	Preassigned flag	'N'	Char(1)	

Record Name	Record	Default value	Field type	Description
	Deliver_not_before_date	Not_before_date	Date(8)	Not_before_date of the order
	Deliver_not_after_date	Not_after_date	Date(8)	Not_after_date of the order
	Shipping terms	Ordhead.freight_terms	Char(3)	Freight Terms of the order
	Buyer code	Ordhead.buyer	Char(12)	Buyer of the PO.
File trailer	File record identification	FTAIL	Char(5)	File trailer identifier
	Line number	Internally incremented	Number(10)	Sequential line number of file
	Number of transaction lines	Internally determined	Number(10)	Total number of transactions (not including FHEAD and FTAIL)

Order Detail file

Record Name	Record	Default value	Field type	Description
File header	File line identifier	FHEAD	Char(5)	identifies file record type
	Line number	Begins at 0000000001	Number(10)	identifies file line number
	Program descriptor	PODT	Char(5)	identifies the program
	Create date	YYYYMMDDHH24MISS format	Char(14)	file create date
File Detail	Detail file identifier	FDETL	Char(5)	Identifies the Detail line
	line number	Incremented internally	Number(10)	sequential line number

Record Name	Record	Default value	Field type	Description
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	Ordloc.location (wh only)	Number(4)	This field contains the location to which the item will be ordered to.
	Transaction day date/time	sysdate	Datetime(12)	system date
	PO number	'P' + order number	char(9)	Identifies the unique PO number
	Item id	Ordloc.sku	Char(16)	Sku on the order
	Requested unit qty	Ordloc.qty_ordered	Number(12,4)	Contains the total number of items ordered to a specific location.
	Ordered case pack	Ordsku.case_pack_size	Number(12,4)	Contains the case pack size that the item was ordered in
	Hang/Flat/Shoe Indicator	Hanger attribute or default door type	Char(1)	F=Flat, H=Hang, S=Shoe, A=All
File Trailer	File Line identifier	FTAIL	Char(5)	Identifies the trailer line
	line number	Incremented internally	Number(10)	sequential line number

Record Name	Record	Default value	Field type	Description
	number of transaction lines	Total number of detail lines	Number(10)	total number of detail lines in file (not including FHEAD and FTAIL)

Stock Order file

Record Name	Record	Default value	Field type	Description
File Header	Detail file identifier	FHEAD	Char(5)	Identifies the header line
	line number	Incremented internally	Number(10)	sequential line number
	Program descriptor	STOR	Char(5)	Identifies the program
	Create date	YYYYMMDDHH24MISS	Char(14)	File create date
File detail	File record descriptor	FDETL	Char(5)	Identifies the detail line
	Line number	Incremented internally	Number(10)	sequential line number
	Action_type	‘A’, ‘M’, ‘D’	Char(1)	Add, modify, or delete action type
	Location	alloc_header.wh	Number(4)	From Warehouse location
	Transaction day date/time	sysdate	Datetime(12)	system date
	distribution number	‘A’ + alloc_no	char(9)	Allocation number. Prefix ‘A’ for alloc
	Download comment	NULL	Char(30)	Comment to be printed on the label (for future use)
	Pick_not_before_date	Not_before_date	Date(8)	Not_before_date of theorder
	Pick_not_after_date	Not_after_date	Date(8)	Not_after_date of the order
	Event code	Promotion or NULL	Char(6)	Promotion’s event number

Record Name	Record	Default value	Field type	Description
	Event description	Prom_desc or NULL	Char(25)	Event description
	priority	1	Char(4)	Priority
	Order Type	ALLOC_HEADER.ORDER_TYPE	Char(9)	Type of Order : 'PO' or 'PREDIST'
	Break by Distro	'N'	Char(1)	Controls the mixing of orders (distros) in a container
	Carrier Code	NULL	Char(4)	Code of the carrier for the order
	Carrier Service Code	NULL	Char(6)	Carrier's service code for the delivery, First Class, and son on (Future Use)
	Route	NULL	Char(10)	Route specified for the delivery
	Ship Address Description	NULL	Char(30)	The description (such as the store name)
	Ship Address Line 1	NULL	Char(30)	Shipping Address Line 1
	Ship Address Line 2	NULL	Char(30)	Shipping Address Line 2
	Ship AddressLine 3	NULL	Char(30)	ShippingAddressLine 3
	ShipAddressLine 4	NULL	Char(30)	ShippingAddressLine 4
	ShipAddressLine 5	NULL	Char(30)	ShippingAddressLine 5
	City	NULL	Char(25)	Shipping City
	State	NULL	Char(3)	Shipping State
	Zip	NULL	Char(10)	Shipping Zip
	Billing Address Description	NULL	Char(30)	The description (such as company name). This is the first line of the address block.
	Billing Address 1	NULL	Char(30)	Billing Address Line 1
	Billing Address 2	NULL	Char(30)	Billing Address Line 2

Record Name	Record	Default value	Field type	Description
	Billing Address 3	NULL	Char(30)	Billing Address Line 3
	Billing Address 4	NULL	Char(30)	Billing Address Line 4
	Billing Address 5	NULL	Char(30)	Billing Address Line 5
	Amount 1	NULL	Number(8,2)	Amount Charge 1
	Amount 2	NULL	Number(8,2)	Amount Charge 2
	Amount 3	NULL	Number(8,2)	Amount Charge 3
	PO Number	'P' + ALLOC_HEADER.ORDER_NO	Char(9)	Unique identifier of the purchase order, prefixed with 'P'.
File trailer	File record identification	TTAIL	Char(5)	File trailer identifier
	Line number	Internally incremented	Number(10)	Sequential line number of file
	Number of transaction lines	Internally determined	Number(6)	Total number of transactions (not including FHEAD and FTAIL)

Stock Allocation file

Record Name	Record	Default value	Field type	Description
File header	File line identifier	FHEAD	Char(5)	identifies file record type
	Line number	Begins at 0000000001	Number(10)	identifies file line number
	Program descriptor	STAL	Char(10)	identifies the program
	Create date	YYYYMMDDHH24MISS format	Char(14)	file create date
File Detail	Detail file identifier	FDETL	Char(5)	Identifies the Detail line
	line number	Incremented internally	Number(10)	sequential line number

Record Name	Record	Default value	Field type	Description
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	alloc_header.wh	Number(4)	From Warehouse location
	Transaction day date/time	sysdate	Datetime(12)	system date
	distribution number	'A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc
	Item Id	ALLOC_HEADER.SKU	Char(16)	Unique item identifier
	requested unit qty	Alloc_detail.qty_allocated	Number(12,4)	quantity allocated
	destination id	Alloc_detail.store or wh	Number(4)	Allocation location
	price	Item_zone_price.unit_retail	Number(5,2)	Retail price
	print upc flag	NULL	char(1)	Print upc flag
	ticket type	item_ticket.ticket_type	Number(4)	Receiving Ticket type of item.
	priority	1	Char(4)	Priority
	expedite flag	'N'	char(1)	Flag indicating whether the order should be shipped via normal or expedite carrier service.
File Trailer	File Line identifier	FTAIL	Char(5)	Identifies the trailer line
	line number	Incremented internally	Number(10)	sequential line number

Record Name	Record	Default value	Field type	Description
	number of transaction lines	Total number of detail lines	Number(6)	total number of detail lines in file (not including FHEAD and FTAIL)

Component Ticketing file layout

Record Name	Record	Default value	Field type	Description
File Header	File Line identifier	FHEAD	Char(5)	Identifies the trailer line
	Line number	0000000001	Number(10)	identifies file line number
	Program descriptor	CPTT	Char(4)	identifies the program
	Create date	YYYYMMDDHH24MISS	Char(14)	file create date
File detail	file record descriptor	FDETL	Char(5)	Detail line descriptor
	line number	Incremented internally	Number(10)	sequential line number
	Action_type	‘A’	Char(1)	'A'dd, 'M'odify, 'D'elete
	Location	alloc_header.wh	Number(4)	location that items will be allocated from
	Transaction date/time	vdate	Datetime(12)	date/time created in RMS
	distribution number	alloc_header.alloc_no	char(9)	Unique identifier of the distribution.
	Master item id	alloc_header.sku	Char(16)	Unique identifier of the pack item

Record Name	Record	Default value	Field type	Description
	Dest Id	alloc_detail.store	Number(4)	Identifier of the ship destination
	Component Item ID	v_packsku_qty.sku	Char (16)	item identifier of the component
	price	Item_zone_price.unit_retail	Number(7,2)	Price of the merchandise.
File Trailer	file record identification	FTAIL	Char(5)	File trailer
	line number	Incremented internally	Number(10)	sequential line number
	number of transaction lines	Total number of detail lines	Number(6)	total number of transaction lines in file (not including FHEAD and FTAIL)

Technical issues

Clients will have to determine how frequently to run this program. If order versions are only needed at the end of the business day, e.g. when orders are mailed or transmitted to suppliers, then it might be sufficient to run this program once a day (after the replenishment orders are built and before the EDI orders are transmitted to the supplier).

Potential future enhancement, write a report when multiple records for the same order are on the table. This might be used to indicate whether orders versions should be written more frequently.

Information is selected into arrays to improve performance.

This program must be run if interfacing with Nautilus.

Chapter 3 – Allocation Update Download [allocupd]

Modification

This program was modified by adding fields to bring shipping and billing addresses to five lines in the stock order file.

Design overview

This program will send updated pricing information to the warehouse for the items that will be allocated to stores. The allocupd.pc program will get price change information for any allocations, which have been created, and write the information to stock order and stock allocation flat files. This program will ensure that any SKU/store unit retail information that is changed after the allocation has been downloaded will be updated in Nautilus system.

The new batch program will loop through the price_hist table, selecting records whose unit retail will change tomorrow, and transaction type is in 4 or 11. Any allocations that have been created for the SKU/store combination will then be downloaded with the new retail.

The allocation update download program will also produce a file that contains the ticketing and retail information for non-sellable pack items that will be cross-docked to store locations. This will allow the warehouse to correctly ticket the component items before the merchandise leaves to its final store destination.

Scheduling constraints

Processing Cycle:	N/A
Scheduling Diagram:	This program should always be run after pcext.pc and after ordrev.pc
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	multi threading available

Restart recovery

The logical unit of work is a row from the price_hist table, selecting information for all SKU/store combinations that have a price change in effect for tomorrow. This program will contain restart recovery and multi threading based on store. The driving cursor is as follows:

select distinct 'S', ---"normal" SKU, which includes staple, fashion SKU, and pack item

```
r.unit_retail,
0 pack_no,
r.sku,
r.store
```

from price_hist r

where r.tran_type in (4, 11)

and r.action_date = TO_DATE(:os_tomorrow, 'YYYYMMDD')

UNION ALL /* this union selects to find if there were price changes for the SKUs in a pack item */

Select distinct 'P', ---- component pack item indicator

```
r.unit_retail,
vpq.pack_no,
vpq.sku,
r.store
```

from price_hist r,

```
v_packsku_qty vpq,
packhead ph
```

where r.sku = vpq.sku

```
and ph.pack_no  = vpq.pack_no
and ph.sellable_ind = 'N'
and r.tran_type  in (4, 11)
and r.action_date  = TO_DATE(:os_tomorrow, 'YYYYMMDD');
```

Retrieve all SKUs that have price changes in effect tomorrow. This information will be written to the Stock Order and Stock Allocation detail files. Also, get all component SKUs of the non-sellable packs that have price changes. The component retail changes will be written to the component ticketing file.

Program flow

N/A

Shared modules

N/A

Function level description

init()

- Declare restart variables
- Get system date.
- Open output files.
- Write FHEAD record to files.
- Call Init_buffers().

init_buffers()

- Set up format strings for outputting FHEAD, FDETL and FTAIL records.

process()

Declare a cursor to retrieve any allocations that contain that SKU/store combination.

```
select ad.alloc_no,
       ah.order_no
       ad.qty_allocated
 from alloc_header ah,
          alloc_detail ad
    where ah.alloc_no = ad.alloc_no
      and ad.store = rec.store  --- from price hist
      and ah.sku   = rec.sku    --- from price hist
      and ad.qty_allocated > NVL(ad.qty_transferred, 0)
order by order_no;
```

Declare a cursor to check for order revision.

Select 'X'

From ordhead_rev

Where order_no = :os_order_no

And origin_type = 'V';

For each row meeting our criteria from the price_hist table.

- Default action_type to 'M'
- Default priority to '1'
- Default expedite_flag to 'N'

Loop through price_hist records (fetch rows into array, equal to commit max counter).

For each SKU/store combination found on price_hist, get allocation details for the SKU/store.

If the order number on the allocation has changed (the order number will be initialized to 0) from the previous order number, then check if a revision exists for that order. If a revision does not exist for an order, then we will not send allocation information to Logistics, because the order has not been downloaded yet.

If order revision exists,

- If the SKU is not a component SKU (i.e. indicator = 'S')
  - Fetch the ticket type from the item_ticket table where the po_print_type = 'R' (ie. ticket at receiving location). If no ticket is found, then the ticket type will default to '0000'.
  - Write detail record (FDETL) to the Stock Order and Stock Allocation detail files for the new unit retail created in RMS.
- If indicator = 'P' (i.e. processing a pack item component)
  - Write detail record (FDETL) to the Component Ticketing file for the new unit retail created in RMS.

final()

- Write file trailer (FTAIL), copy temporary file to final file, close files.

I/O specification

All character variables are right-padded with blanks and left justified; all numerical variables are left-padded with zeroes and right-justified. Missing variables are blank.

Stock Order file

Record Name	Record	Default value	Field type	Description
File Header	Detail file identifier	FHEAD	Char(5)	Identifies the header line
	Line number	Incremented internally	Number(10)	sequential line number
	Program descriptor	STOR	Char(5)	Identifies the program
	Create date	YYYYMMDDHH24MI	Char(12)	File create date
File detail	File record descriptor	FDETL	Char(5)	Identifies the detail line
	Line number	Incremented internally	Number(10)	sequential line number
	Action_type	'M'	Char(1)	Add, modify, or delete action type

Record Name	Record	Default value	Field type	Description
	Location	alloc_header.wh	Number(4)	From Warehouse location
	Transaction day date/time	sysdate	Datetime(12)	system date
	distribution number	'A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc
	Downloadcomment	NULL	Char(30)	Comment to be printed on the label (for future use)
	Pick_not_before_date	Not_before_date	Date(8)	Not_before_date of the order
	Pick_not_after_date	Not_after_date	Date(8)	Not_after_date of the order
	Event code	Promotion or NULL	Char(6)	Promotion's event number
	Event description	Prom_desc or NULL	Char(25)	Event description
	priority	1	Char(4)	Priority
	Order Type	code_detail.code	Char(9)	Type of Order: PO or PREDIST.(Taken from alloc_header.order_type)
	Break byDistro	'N'	Char(1)	Contols the mixing of orders (distros) in a container
	Carrier Code	NULL	Char(4)	Code of the carrier for the order
	Carrier Service Code	NULL	Char(6)	Carrier's service code for the delivery, First Class, and son on (Future Use)
	Route	NULL	Char(10)	Route specified for the delievery
	Ship Address Description	NULL	Char(30)	The description (such as the store name). This is the first line of the address block
	Ship Address Line 1	NULL	Char(30)	Shipping Address Line 1
	Ship Address Line 2	NULL	Char(30)	Shipping Address Line 2
	Ship Address Line 3	NULL	Char(30)	Shipping Address Line 3

Record Name	Record	Default value	Field type	Description
	Ship Address Line 4	NULL	Char(30)	Shipping Address Line 4
	Ship Address Line 5	NULL	Char(30)	Shipping Address Line 5
	City	NULL	Char(25)	Shipping City
	State	NULL	Char(2)	Shipping State
	Zip	NULL	Char(9)	Shipping Zip
	Billing Address Description	NULL	Char(30)	The Description(such as company name). This is the first line of the address block.
	Billing Address Line 1	NULL	Char(30)	Billing Address Line 1
	Billing Address Line 2	NULL	Char(30)	Billing Address Line 2
	Billing Address Line 3	NULL	Char(30)	Billing Address Line 3
	Billing Address Line 4	NULL	Char(30)	Billing Address Line 4
	Billing Address Line 5	NULL	Char(30)	Billing Address Line 5
	Amount 1	NULL	Number(8,2)	Amount Charge 1
	Amount 2	NULL	Number(8,2)	Amount Charge 2
	Amount 3	NULL	Number(8,2)	Amount Charge 3
	Po number	'P' + Alloc_header.order_no	Char(9)	Unique identifier of the purchase order, prefixed with 'P'
File trailer	File record identification	TTAIL	Char(5)	File trailer identifier
	Line number	Internally incremented	Number(10)	Sequential line number of file
	Number of transaction lines	Internally determined	Number(10)	Total number of transactions (not including FHEAD and FTAIL)

Stock Allocation file

File Header

Record	Default value	Field type	Description
File line identifier	FHEAD	Char(5)	identifies file record type
Line number	Begins at 0000000001	Number(10)	identifies file line number
Program descriptor	STAL	Char(5)	identifies the program
Create date	YYYYMMDDHH24MI format	Char(12)	file create date

File Detail

Record	Default value	Field type	Description
Detail file identifier	FDETL	Char(5)	Identifies the Detail line
line number	Incremented internally	Number(10)	sequential line number
Action_type	'M'	Char(1)	type of record is Modify
Location	alloc_header.wh	Number(4)	From Warehouse location
Transaction day date/time	sysdate	Datetime(12)	system date
distribution number	A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc
Item id	alloc_header.sku	Char(16)	Unique item identifier
requested unit qty	qty_allocated	Number(12,4)	quantity allocated
destination id	price_hist.store	Number(4)	Allocation location
price	price_hist.unit_retail	Number(7,2)	Retail price
print upc flag	'N'	char(1)	Print upc flag
ticket type	item_ticket	Number(4)	Ticket type
priority	1	Char(4)	Priority

Record	Default value	Field type	Description
expedite flag	'N'	char(1)	Expedite flag

File Trailer

Record	Default value	Field type	Description
File Line identifier	FTAIL	Char(5)	Identifies the trailer line
line number	Incremented internally	Number(10)	sequential line number
number of transaction lines	Total number of detail lines	Number(6)	total number of detail lines in file (not including FHEAD and FTAIL)

Component Ticketing file layout

File Header

Record	Default value	Field type	Description
File Line identifier	FHEAD	Char(5)	Identifies the trailer line
Line number	0000000001	Number(10)	Identifies file line number
Program descriptor	CPTT	Char(4)	Identifies the program
Create date	YYYYMMDDHH24MISS	Char(14)	File create date

File Detail

Record	Default value	Field type	Description
file record descriptor	FDETL	Char(5)	Detail line descriptor
line number	Incremented internally	Number(10)	sequential line number
Action_type	'A'	Char(1)	'A'dd, 'M'odify, 'D'elete
Location	alloc_header.wh	Number(4)	Location that items will be allocated from

Record	Default value	Field type	Description
Transaction date/time	vdate	Datetime(12)	date/time created in RMS
distribution number	alloc_header.alloc_no	char(9)	Unique identifier of the distribution.
Master item id	alloc_header.sku	Char(16)	Unique identifier of the pack item
Dest Id	alloc_detail.store	Number(4)	Identifier of the ship destination
Component Item ID	v_packsku_qty.sku	Char (16)	item identifier of the component
price	price_hist.unit_retail	Number(7,2)	Price of the merchandise.

File Trailer

Record	Default value	Field type	Description
file record identification	FTAIL	Char(5)	File trailer
line number	Incremented internally	Number(10)	sequential line number
number of transaction lines	Total number of detail lines	Number(6)	total number of transaction lines in file (not including FHEAD and FTAIL)

Technical issues

N/A

Chapter 4 – Approved Warehouse Transfers Download [tsfdnld]

Modification

In this program, the download file layout for THEAD and TDETL transaction type was modified to add a space between each column according to RLS flat file format. The program was also modified by changing the length of carrier code to 4 and order number to 8.

Design overview

Function

This program processes all warehouse transfers that are approved, with a freight code of Normal or Expedite and have a release date equal to or less than tomorrow. If the destination location is a store, the store must be on the ship schedule to be shipped tomorrow. Shipments are created for these transfers and the shipment information is downloaded into a file to be used by an external WMS. Transfer status will be updated to 'E' (Extracted).

This program will produce two additional files. The first file contains component ticket and retail information, for non sellable pack items. This will provide the correct ticketing information for the warehouse to ticket the components of non sellable pack items. The second file contains outbound work order processing information for stock allocations. The work order information is found on the work order tables, wo_wip, wo_head, and wo_sku_loc.

When interfacing with Nautilus all three files will need to be converted into the proper flat file format, so that Nautilus can process.

Note: Transfers that are supposed to be combined into Combined Transfer (CT transfer type) will not be downloaded by this program. Transfers with a freight type = 'E' (Expedite) and a release date <= today will ignore the shipping schedule and be downloaded tonight. Transfers with a freight type = 'H' (Hold) will be ignored by this program.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFALLOC	Yes	Yes	No	No	No
TSFHEAD	Yes	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	No	No	No
SHIPMENT	Yes	Yes	Yes	No	No
STORE_SHIP_DATE	Yes	Yes	No	No	No
WO_HEAD	Yes	Yes	No	No	No
WO_SKU_LOC	Yes	Yes	No	No	No
WO_WIP	Yes	Yes	No	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDCUST	Yes	Yes	No	No	No
CUSTOMER	Yes	Yes	No	No	No
ITEM_TICKET	No	Yes	No	No	No
V_PACKSKU_QTY	No	Yes	No	No	No

The tsfdnld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(6).

Scheduling constraints

Processing Cycle: N/A

Scheduling Diagram: Phase 3. Constraints: after TSFCOMB.PC

Pre-Processing: N/A

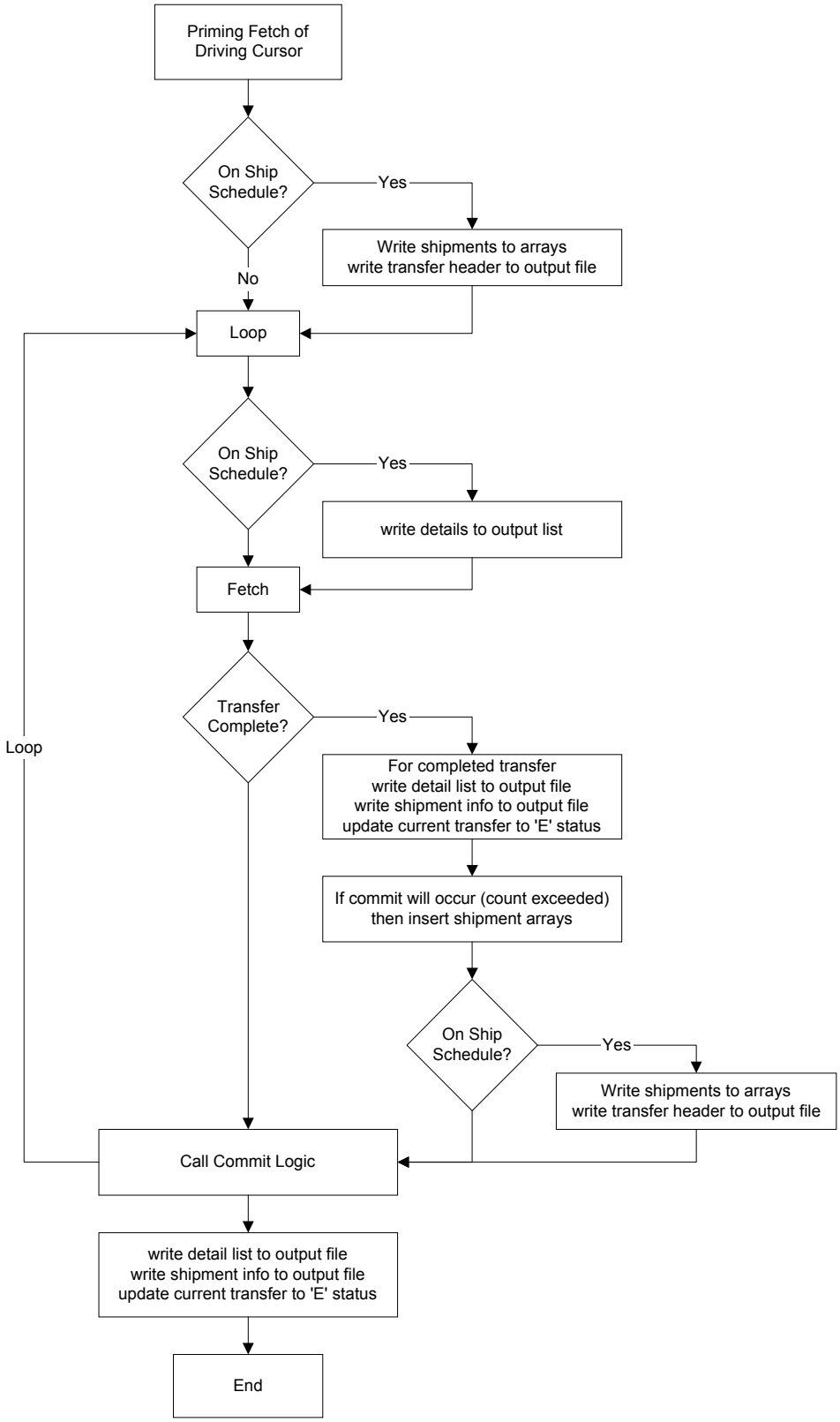
Post-Processing: N/A

Threading Scheme: N/A

Restart recovery

```
SELECT tsfhead.tsf_no,
           tsfhead.from_loc_type,
           tsfhead.from_loc,
           tsfhead.to_loc_type,
           tsfhead.to_loc,
           tsfhead.tsf_type,
           tsfhead.freight_code,
           ROWIDTOCHAR(tsfhead.rowid),
           ';'||to_char(tsfhead.tsf_no),
           tsfdetail.sku,
           (tsfdetail.tsf_qty)*1000,
           nvl(tsfdetail.inv_status, 0)
     FROM  tsfhead,
           tsfalloc,
           tsfdetail
    WHERE  tsfhead.status = 'A'
      AND  tsfhead.freight_code in ('N','E')
      AND  tsfhead.from_loc_type = 'W'
      AND  tsfhead.tsf_type not in ('PO','SR')
      AND  nvl(tsfalloc.merge_ind,'N') = 'N'
      AND  tsfhead.tsfalloc_no = tsfalloc.tsfalloc_no
(+)
      AND  nvl(tsfalloc.release_date,
to_date(:ps_tomorrow,'YYYYMMDD'))
                <= to_date(:ov_tomorrow,'YYYYMMDD')
      AND  tsfdetail.tsf_no = tsfhead.tsf_no
      AND  nvl(tsfdetail.tsf_qty,0) > 0
      AND  tsfhead.tsf_no > nvl(:ora_restart_tsf_no, -
999)
 ORDER BY  tsfhead.tsf_no;
```

Program flow



Shared modules

NEXT_SHIPMENT_SQL used to get the next shipment number.

PRICING_ATTRIB_SQL.GET_RETAIL(): get the unit retail from item_zone_pricing table for a sku/store.

Function level description

Init()

Initialize restart recovery.

Open output file.

Format header, detail, and shipment buffers (for writing output).

Determine tomorrow's date

Determine order type

Call function get_order_type to determine order type

Call function to write output file header information, write_std_header()

Process()

This function should select all transfer details and quantities for transfers that are ready to ship from a warehouse tomorrow. Each transfer (header, detail information, and shipment information) should be written to an output file for the WMS to upload with transfer requirements. When a transfer has been completed, that is all information has been written to a file and the shipment information has been created, its status will be updated to Extracted ('E').

The flow of logic is as follows:

- Fetch the first transfer record from the driving cursor.
- Get_ship_flag (determines if current transfer is due to ship tomorrow)
- if the transfer should be shipped then
  - call get_thead_info() to get the customer address information if it is a customer order type of transfer.
  - Call write_recs_to_struct() to create shipment number and write records to structure
  - Call write_head_to_str() to write to the THEAD structure.
- End if;
- Main processing loop through the transfer tables
  - If transfer should be shipped then
    - Call Get_detail_info() to get the ticketing and retail information. Also, decode the expedite flag.
    - Call write_detail_to_list() write TDETL to link list
    - Call Process_wo() to process the work order information

        - End if;
        - Fetch next transfer record
        - If the transfer number just changed, then
            - If the transfer should be shipped write into from the previous transfer to the file
                - Call Write_list_to_file() write link list of details to flat file.
                - Call Write_wo_to_file()
                - Call write_pack_to_file()
                - Call write_tail_to_file()
            - End if;
            - Call update_records() to update the appropriate tables
            - Now start working on the newly fetched transfer
            - Call get_ship_flag() to see if new transfer should be shipped
            - If transfer should be shipped, then
                - Call Get_thead_info()
                - Call write_recs_to_struct()
                - Call write_head_to_str()
            - End if;
        - End if;
        - Commit records and updates.
- End of transfer loop
- If the last transfer fetched should be shipped, then write final to file
    - Call write_list_to_file()
    - Call write_wo_to_file()
    - Call write_pack_to_file()
    - Call write_tail_to_file()
- End if;
- Call update_records()

Get_ship_flag()

This function calls validate_ship_schedule() to determine if transfer will be shipped tomorrow. If the transfer is set to expedite status, then the shipping schedule is ignored and the transfer is processed.

validate_ship_schedule()

This function validates that a ship date exists between today and tomorrow for the from warehouse and the to store combination (held on STORE_SHIP_DATE table).

get_thead_info()

This function retrieves the customer address from the customer table for the customer order transfer. If the customer is going to pick up the merchandise, then a message, "customer order for: < customer name > " will be displayed in the event description. This will indicate to the warehouse that it is a customer order, pick up.

- If customer order and ship direct
  - set break by distro value = 'Y'.
  - populate billing and shipping addresses with customer address info.
  - Set dest. Id = courier value from tsfhead
  - Set Courier/route/service codes = NULL
- If not customer order
  - set break by distro value = 'N'
  - do not populate billing and shipping address
  - set dest. Id = store or warehouse
  - set courier/route/service codes = NULL

get_detail_info()

This function decodes the freight code

- if freight_code = 'E' then
  - expedite_flag = 'Y';
- else
  - expedite_flag = 'N';
- end if;

Get the ticket type for the item from item_ticket table where the po_print_type = 'R' (i.e. print at the time of receipt). There may be several ticket types for the item with 'R' print type.Ttherefore, get the first ticket type in the fetch.

Get Unit retail for the item/location from the item_zone_price tables by calling the package PRICING_ATTRIB_SQL.GET_RETAIL.

If item is going to a store location call function comp_tckt() to write component ticketing file

process_wo ()

This function retrieves all the work order information for the selected stock allocation and Calls write_wo_to_list()

Write_wo_to_list()

This function writes the work order information to the structures

Write_wo_to_file()

This function prints out the work order structure to flat file

Comp_tckt()

This function selects from pack_head for the item and sellable_ind = 'N'.

- If non Sellable 'P'ack item is found
  - loop through component items that make up the pack item on the v_packsku_qty table.
  - Call pricing_attrib_sql.get_retail package to get the retail for the component SKU.
  - Call write_pack_to_list() Write FDETL record for component SKU, retail, and ticket type to file
  - End loop;
- end if;

write_pack_to_list()

This function writes the component ticketing and retail information to the structure

write_pack_to_file()

This function prints component ticketing and retail information structure to flat file

Write_std_header()

This function Increment counters and writes FHEAD record to file

Write_std_trailer()

This function increments counters and writes FTAIL record to file

write_tail_to_file()

This function writes the TTAIL structure to the output file

write_detail_to_list()

This function makes detail record string (TDETL) and add to linked list and calls add_dtl_to_list() function.

add_dtl_to_list()

This function will add ps_temp_dtl string to linked list

get_order_type()

This function gets order type from code_detail

write_head_to_str()

This function gets order header string (THEAD) and write structure

Write_recs_to_struct()

This function will be called when a new transfer number is encountered. Transfer header information is written to arrays that will update the status. A new shipment number is created and shipment information is written to arrays that will insert new shipment records into the shipment table.

write_list_to_file()

This function writes linked list detail records to file

update_records()

- perform array update of tsfhead using rowid, set status = 'E'
- perform array insert of newly created shipments

Final()

Call function to write output file trailer information, write_std_trailer().

The tsfdnld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(6).

I/O specification

File I/O

Output files should be specified on the command line.

Transfer download file

Record Name	Field Name	Field Type	FieldValue	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFD	Identified file as 'Inventory Adjustments'
	File Create Date	Date	Sysdate	Date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	Specified by external system	Line number of the current file

Record Name	Field Name	Field Type	FieldValue	Description
	Transaction Set Control Number	Number(14)	Specified by external system	Used to force unique transaction check
	Action Type	Char(1)	'A' (hardcode)	'A'dd, 'D'elete, 'M'odify
	Delimeter *	Char(1)	<Space>	
	Location (DC)	Number (4)	Tsfhead.from_loc	Code for the DC.
	Delimeter *	Char(1)	<Space>	
	Transaction Date/Time	YYYYMMDDHHMI	Period.vdate	Date/Time created in RMS
	Delimeter *	Char(1)	<Space>	
	Distribution Number	Char(9)	Shipment.shipment	Unique identifier of the distribution.
	Delimeter *	Char(1)	<Space>	
	Download Comment	Char (30)	NULL	Comment to be printed on the label (for future use)
	Delimeter *	Char(1)	<Space>	
	Pick-Not-Before-date	YYYYMMDD	Period.vdate	Date before which merchandise will not be distributed
	Delimeter *	Char(1)	<Space>	
	Pick-Not-After-Date	YYYYMMDD	Period.vdate + (specified time from codes table)	Date by which merchandise must be distributed. Extra days will be determined by a code type = 'DATE'
	Delimeter *	Char(1)	<Space>	
	Event Code	Char(6)	NULL or tsfalloc.tsfalloc_no	Identifier of event. Only used for stock allocations
	Delimeter *	Char(1)	<Space>	
	Event Description	Char(25)	NULL or tsfalloc.alloc_desc	Description of event. Only used for stock allocations

Record Name	Field Name	Field Type	FieldValue	Description
	Delimeter *	Char(1)	<Space>	
	Priority	Char(4)	Default to 1	Priority 1=highest
	Delimeter *	Char(1)	<Space>	
	Order Type	Char(9)	Default from system optionTables	Order type (Automatic,Manual orWave)
	Delimeter *	Char(1)	<Space>	
	Break by Distro	Char(1)	Default from codes tables	Controls the mixing of orders (distros) in a container
	Delimeter *	Char(1)	<Space>	
	Carrier Code	Char(4)	NULL	Code of the carrier for the order
	Delimeter *	Char(1)	<Space>	
	Carrier Service Code	Char(6)	NULL	Carrier's service code for the delivery, First Class, etc.
	Delimeter *	Char(1)	<Space>	
	Route	Char(10)	NULL	Route specified for the delivery
	Delimeter *	Char(1)	<Space>	
	Ship Address Description	Char(30)	NULL or customer address	Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Ship Address line 1	Char(30)	NULL or customer address	Shipping address line 1. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	

Record Name	Field Name	Field Type	FieldValue	Description
	Ship Address line 2	Char(30)	NULL or customer address	Shipping address line 2. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Ship Address line 3	Char(30)	NULL or customer address	Shipping address line 3. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Ship Address line 4	Char(30)	NULL or customer address	Shipping address line 4. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Ship Address line 5	Char(30)	NULL or customer address	Shipping address line 5. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	City	Char(25)	NULL or customer address	Shipping city. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	State	Char(3)	NULL or customer address	Shipping state. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	

Record Name	Field Name	Field Type	FieldValue	Description
	Zip	Char(10)	NULL or customer address	Shipping zip. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Billing Address Description	Char(30)	NULL or customer address	The description (such as company name, etc.). This is the first line of the address block. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Billing Address line 1	Char(30)	NULL or customer address	Billing address line 1. Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Billing Address line 2	Char(30)	NULL or customer address	Billing address line 2, Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Billing Address line 3	Char(30)	NULL or customer address	Billing address line 3, Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Billing Address line 4	Char(30)	NULL or customer address	Billing address line 4, Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	

Record Name	Field Name	Field Type	FieldValue	Description
	Billing Address line 5	Char(30)	NULL or customer address	Billing address line 5, Used to store only customer order (ship direct) addresses.
	Delimeter *	Char(1)	<Space>	
	Amount 1	Number(8, 2)	NULL	Amount charge 1
	Delimeter *	Char(1)	<Space>	
	Amount 2	Number(8, 2)	NULL	Amount charge 2
	Delimeter *	Char(1)	<Space>	
	Amount 3	Number(8, 2)	NULL	Amount charge 3
	Delimeter *	Char(1)	<Space>	
	Order No.	Char(8)	NULL	Purchase OrderIdentifier
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	Specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	Specified by external system	used to force unique transaction check
	Action Type	Char(1)	‘A’ (hardcode)	‘A’dd, ‘D’elete, ‘M’odify
	Delimeter *	Char(1)	<Space>	
	Location (DC)	Number (4)	NULL	Code for the DC (future use)
	Delimeter *	Char(1)	<Space>	
	Transaction Date/Time	YYYYMMDDHHMI	Period.vdate	Date/Time created in RMS
	Delimeter *	Char(1)	<Space>	
	Distribution Number	Char(9)	Shipment.shipment	Unique identifier of the distribution.
	Delimeter *	Char(1)	<Space>	
	Item ID	Char(16)	Tsfdetail.sku	Item identifier
	Delimeter *	Char(1)	<Space>	

Record Name	Field Name	Field Type	FieldValue	Description
	Requested Unit Qty	Num(12,4)	Tsfdetail.tsf_qty	Number of units to distribute to the destination
	Delimeter *	Char(1)	<Space>	
	Destination ID	Number (4)	Tsfhead.routing_code (if ship direct to Customer order)Tsfhead.to_loc (if store or wh)	Identifier of shipping destination. If customer order and ship direct, then field contains a carrier value. If it is direct to store or warehouse, then populate with the store or warehouse location.
	Delimeter *	Char(1)	<Space>	
	Price	Number (7,2)	Item_zone_price.unit_retail	Price of merchandise
	Delimeter *	Char(1)	<Space>	
	Print UPC Flag (‘Y’,’N’)	Char(1)	‘N’ (hardcode)	Whether to print UPC on tickets (Future use)
	Delimeter *	Char(1)	<Space>	
	Ticket Type	Number (4)	Item_ticket.ticket_type	Type of ticket refers to ticket type table. This field will be populate with the “ticket at receipt”.
	Delimeter *	Char(1)	<Space>	
	Priority	NUMBER (4)	1 (hardcode)	Priority 1 = highest
	Delimeter *	Char(1)	<Space>	
	Expedite Flag	VARCHAR(1)‘Y’ or ‘N’	Tsfhead.freight_code (translate value to ‘Y’ or ‘N’)	Flag indicating whether the order should be shipped via normal or expedited carrier service

Record Name	Field Name	Field Type	FieldValue	Description
Transaction Trailer	File type record descriptor	Char(5)	TTAIL	Identifies file record type
	File Line sequence	Number(10)	Specified by external system	Line number of the current file
	Transaction detail line count	Number(6)	Sum of detail lines	Sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Line Count	Number(10)	total detail + transaction head lines	sum of all transaction lines, not including file header and trailer

Work Order Download File

RecordName	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes:0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	OWOD	Identifies file as ‘Outbound Work Order Download’
	File Create Date	Date	Create date	date file was written by external system
Trans-action Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.

RecordName	Field Name	Field Type	Default Value	Description
	Action Type	Char(1)	'A'	The action being performed on the work order. This will always be 'A' since transfer work orders can't be modified once they've been extracted.
	Location (DC)	Char(4)	Wo_sku_loc.wh	When an item is crossdocked, this field holds the value of the flow-through warehouse. Otherwise it holds the value of the final destination.
	Transaction Date/Time	Char (12) format: YYYYMMDDHHMI	Vdate	sysdate without the seconds
	Distribution Number	Char(9)	Shipment	This field will hold the shipment number of the transfer the work order is associated with.
	Item ID	Char (16)	Wo_sku_loc.sku	Valid item identifier for a staple SKU, fashion SKU, or Pack Item
	Dest ID	Number(4)	Wo_sku_loc.location	Unique identifier of the final shipping destination.
	WIP Sequence No.	Number(7)	Wo_wip.seq_no	Work in Progress Sequence Number
	WO WIP Code	Char (6)	Wo_wip.code	WWIP code from codes table
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	DeterminedInternally	Number of records/transactions processed in current file (only records between head & tail)

Component ticketing file

Record Name	Record	Default value	Field type	Description
File Header	File Line identifier	FHEAD	Char(5)	Identifies the trailer line
	Line number	0000000001	Number(10)	identifies file line number
	Program descriptor	CPTT	Char(4)	identifies the program
	Create date	YYYYMMDDHH24MISS	Char(14)	file create date
File detail	file record descriptor	FDETL	Char(5)	Detail line descriptor
	line number	Incremented internally	Number(10)	sequential line number
	Action_type	‘A’	Char(1)	'A'dd, 'M'odify, 'D'elete
	Location	Tsfhead.from_loc	Number(4)	location that items will be transferred from
	Transaction date/time	vdate	Datetime(12)	date/time created in RMS
	distribution number	Shipment.shipment	char(9)	Unique identifier of the distribution.
	Master item id	Tsfdetail.sku	Char(16)	Unique identifier of the pack item
	Dest Id	Tsfdetail.to_loc	Number(4)	Identifier of the ship destination
	Component Item ID	v_packsku_qty.sku	Char (16)	item identifier of the component
	price	Item_zone_price.unit_retail	Number(7,2)	Price of the merchandise.

Record Name	Record	Default value	Field type	Description
File Trailer	file record identification	FTAIL	Char(5)	File trailer
	line number	Incremented internally	Number(10)	sequential line number
	number of transaction lines	Total number of detail lines	Number(6)	total number of transaction lines in file (not including FHEAD and FTAIL)

Note: There is a space between fields in the RLS flat file format, except for the standard Retek flat file information, such as file type descriptor, file line identifier, file record counts.

Technical issues

N/A

Chapter 5 – Deals – cost calculations [dealcalc]

Modification

This program was modified by changing the order-by clause in the driving cursor.

Design overview

This new batch program will calculate the net cost, net net cost, and dead net net cost for all items that are on the deal_sku_temp table (which should contain all items or items in hierarchies on deals that are on the deal_queue table, which will contain deals that are about to be approved, unapproved, or closed—any action that would potentially change which deals affect an item). All active deals for each item will be used in the calculation. Once calculated, the costs will be inserted into the deal_sku_cost table.

Scheduling constraints

Processing Cycle:	Phase II (daily)
Scheduling Diagram: deal_sku_temp table	Must be run after ditinsrt.pc, which populates the
Pre-Processing:	
Post-Processing:	Call prepost to delete all records from deal_sku_temp.
Threading Scheme:	SUPPLIER

Restart recovery

```
SELECT dst.sku,
   dst.supplier,
   dst.origin_country_id,  /* DST country not DI
country—if no country given, DO expand out */
   TO_CHAR(dst.start_date,'YYYYMMDD'),
    NVL(TO_CHAR(dh.close_date,'YYYYMMDD'),'-1'),
                NVL(TO_CHAR(dh.close_date +
1,'YYYYMMDD'),'-1'),
              sups.currency_code,
   isc.unit_cost,
   dh.deal_id,
   dd.deal_detail_id,
   dh.currency_code,
NVL(dst.location,  -1) /* DST loc not DI loc—expand out
location unless loc-independent  */
NVL(dst.loc_type,'N')
DECODE(dd.cost_appl_ind,'N',1,'NN',2,'DNN',3)
cost_appl_num,
dd.deal_class,
dd.threshold_value_type,
NVL(dd.qty_thresh_buy_item, -9999),
NVL(dd.qty_thresh_buy_qty, 0),
NVL(dd.qty_thresh_recur_ind,'N'),
NVL(dd.qty_thresh_buy_target, 0),
NVL(dd.qty_thresh_get_item, -9999),
NVL(dd.qty_thresh_get_qty, 0),
NVL(dd.qty_thresh_free_item_unit_cost, 0),
NVL(dd.qty_thresh_get_type, 'Z'),
NVL(dd.qty_thresh_get_value, 0),
   TO_NUMBER(di.merch_level, 0),
   TO_NUMBER(NVL(di.org_level, 99)
  FROM deal_sku_temp dst,
deal_head dh,
deal_detail dd,
deal_itemloc di,
sups,
item_sup_country isc,
```

```
v_restart_supplier vrs
WHERE     dd.deal_id  = dh.deal_id
AND di.deal_id = dd.deal_id
AND di.deal_detail_id = dd.deal_detail_id
AND dh.status = 'A'
AND dh.type in ('A','P')  /* only use
promotional/annual, not PO specific or vendor funded */
AND di.excl_ind  = 'N'
AND sups.supplier = dst.supplier
AND isc.item = dst.sku
AND isc.supplier = dst.supplier
AND isc.origin_country_id = dst.origin_country_id
AND ((dh.close_date is NOT NULL
      AND dst.start_date BETWEEN DECODE(rebate_ind, 'Y',
NVL(dd.rebate_active_date, dh.active_date),
dh.active_date)
                              AND dh.close_date)
     OR (dh.close_date is NULL
         AND dst.start_date >= DECODE(rebate_ind, 'Y',
NVL(dd.rebate_active_date, dh.active_date),
dh.active_date)))
AND ((dh.supplier is NOT NULL AND dst.supplier =
dh.supplier)   /* supplier hierarchy match */
           OR(dh.partner_type = 'S1' AND
isc.supp_hier_lvl_1 = dh.partner_id)
           OR(dh.partner_type = 'S2' AND
isc.supp_hier_lvl_2 = dh.partner_id)
           OR(dh.partner_type = 'S3' AND
isc.supp_hier_lvl_3 = dh.partner_id) )
AND ( (di.merch_level = 1)
            OR (di.merch_level = 2 AND di.division =
dst.division
            OR (di.merch_level = 3 AND di.group_no  =
dst.group_no)
            OR (di.merch_level = 4 AND di.dept =
dst.dept )
            OR (di.merch_level = 5 AND (di.dept =
dst.dept AND di.class = dst.class))
            OR (di.merch_level = 6 AND (di.dept =
dst.dept AND di.class = dst.class AND di.subclass =
dst.subclass))
            OR (di.merch_level = 7 AND di.style =
dst.style)           --style/color hierarchy
```

```
            OR (di.merch_level = 8 AND (di.style =
dst.style AND di.color = dst.color)
            OR (di.merch_level = 9 AND (di.style =
dst.style AND ((di.size1 = dst.size1 OR di.size1 is
NULL)                          AND (di.size2 =
dst.size2 OR di.size2 is NULL)))
            OR (di.merch_level = 10 AND di.sku =
dst.sku))
AND  (di.org_level is NULL AND dst.chain is NULL
            AND dst.area is NULL AND dst.region is NULL
            AND dst.district is NULL AND dst.location is
NULL
     OR (di.org_level = 1 AND di.chain = dst.chain)
     OR (di.org_level = 2 AND di.area = dst.area)
     OR (di.org_level = 3 AND di.region = dst.region)
     OR (di.org_level = 4 AND di.district =
dst.district)
     OR (di.org_level = 5 AND di.location =
dst.location))
AND (di.country_id = dst.country_id OR di.country_id is
NULL)
/* exclude clause here –don't fetch excluded skus */
AND (NOT EXISTS
   SELECT 'x'
     FROM deal_itemloc di1
   WHERE di1.deal_id  = di.deal_id
    AND di1.deal_detail_id = di.deal_detail_id
                AND di1.excl_ind  = 'Y'
AND ( (di1.merch_level = 1)
                    OR (di1.merch_level = 2 AND
di1.division = dst.division
                    OR (di1.merch_level = 3 AND
di1.group_no  = dst.group_no)
                    OR (di1.merch_level = 4 AND
di1.dept = dst.dept )
                    OR (di1.merch_level = 5 AND
(di1.dept = dst.dept AND di1.class = dst.class))
OR (di1.merch_level = 6 AND (di1.dept = dst.dept AND
di1.class = dst.class
                                        AND
di1.subclass = dst.subclass))
            OR (di1.merch_level = 7 AND di1.style =
dst.style)          --style/color hierarchy
```

```
            OR (di1.merch_level = 8 AND (di1.style =
dst.style AND di1.color = dst.color)
            OR (di1.merch_level = 9 AND (di1.style =
dst.style
                                                    AND
((di1.size1 = dst.size1 OR di1.size1 is NULL)
                                                    AND
(di1.size2 = dst.size2 OR di1.size2 is NULL)))
            OR (di1.merch_level = 10 AND di1.sku =
dst.sku))
   AND  (di1.org_level is NULL AND dil.chain is NULL
              AND di1.area is NULL AND di1.region is
NULL
              AND di1.district is NULL AND di1.location
is NULL
     OR (di1.org_level = 1 AND di1.chain = dst.chain)
     OR (di1.org_level = 2 AND di1.area = dst.area)
     OR (di1.org_level = 3 AND di1.region = dst.region)
     OR (di1.org_level = 4 AND di1.district =
dst.district)
     OR (di1.org_level = 5 AND di1.location =
dst.location))
AND (di1.origin_country_id = dst.origin_country_id OR
di1.origin_country_id is NULL))
AND (dst.sku > NVL(:ps_restart_sku, -999) OR  /* restart
on item/supplier/country/start_date */
              (dst.sku = :ps_restart_sku AND
                 (dst.supplier > :ps_restart_supplier OR
         (dst.supplier = :ps_restart_supplier AND
                       (dst.origin_country_id >
:ps_restart_country  OR
            (dst.orign_country_id = :ps_restart_country
AND
   dst.start_date > :ps_restart_date)))))
AND vrs.num_threads  = :pi_num_threads
AND vrs.thread_val  = :pi_thread_val
AND vrs.driver_value = dst.supplier
ORDER BY  dst.sku,
dst.supplier,
dst.origin_country_id,
dst.start_date,
loc,
```

```
dh.close_date,
cost_appl_num,
dh.type,
dh.create_date,
dd.application_order
```

The ORDER BY dh.type's and dh.create_date's asc/desc following rules:

1 create date asc, annual before promotional (dh.type asc)

2 create date desc, annual before promotional

3 create date asc, promotional before annual (dh.type desc)

4 create date desc, promotional before annual

Logical unit of work

SKU/supplier/origin country/start date

Driving cursor

The driving cursor will be dynamically created depending on ordering requirements, which will be determined by deal_type_priority and deal_age_priority of system_options.

Program flow

Tables used:

Table	Select	Insert	Update	Delete
Period	X			
system_options	X			
deal_sku_temp	X			X
deal_head	X			
deal_detail	X			
deal_itemloc	X			
deal_threshold	X			
deal_sku_cost		X		
item_supp_country	X			
Sups	X			

Shared modules

CURRENCY_SQL.CONVERT –convert an amount in deal currency to the equivalent amount in supplier currency if necessary, or vice versa

Function level description

init:

- Retrieve the vdate from the period table (use as calculation date for inserts into deal_cost table).
- Get priority indicators (deal_type_priority, deal_age_priority—these determine annual first vs. promotional first, and oldest first vs. newest first ordering for the driving cursor) from system_options.
- Allocate memory for the deal fetch and cost arrays (call size_arrays) and initialize the linked list for deal target values.
- Restart/recovery initialization.

process:

- Call prepare_driving_cursor to create driving cursor statement based on the system options.
- Use the driving cursor to get all active deals for each item/supplier/origin country/start date on the deal_sku_temp table (use an array fetch).
- For each deal/deal detail, call get_target_threshold_value to find the threshold value to be used in cost calculations.
- Call calculate_cost_driver to get the net, net net, and dead net net cost (initially for location-independent deals and then for the location-specific deals, starting form the costs already calculated for location-independent deals), and create an insert array that includes the net/net net/dead net net cost information AND the location information.
- If commit point reached, call post_insert_delete_records to insert the costs into the deal_sku_cost table FOR EACH LOCATION of the same LUW (including a record with no location if there are location-independent deals), and to delete processed records from the deal_sku_temp table.
- After each set of deals has been processed, call the restart commit logic.

prepare_driving_cursor:

Create driving cursor statement based on the system options deal_type_priority and deal_age_priority, which only affect the ORDER BY clause.

calculate_cost_driver:

This function will drive the process of calculating the net, net net, and dead net net cost, given information on all the deals that apply to a particular SKU/supplier/origin country/start date (pass in array structs which include the target threshold value). Each deal/deal detail record is passed on to the calculate_costs function to do the actual calculation for each LUW + loc, that is, SKU/supplier/origin country/start date/loc.

For each set of deals for a unique item/supplier/country id/start date, the desired end result is to have one record on deal_sku_cost with no location that will hold the item's costs with all location-independent deals accounted for, and additional records on deal_sku_cost for each location, with location-specific discounts applied on top of the location-independent discounts.

- For each new LUW + loc, reset the flag for 'F'ixed Amt value type discount. 'F'ixed Amt value type discount should only be applied once for each LUW + loc.
- For each new LUW, reset the flag and merchandise level for 'EX'clusive deal class discount; for each LUW + loc, reset the merchandise/organization level for 'EX'clusive deal class discount (merchandise level needed to be reset back to before any loc-specific applied). 'EX'clusive deal class discount should only be applied once for each LUW + loc.
- Reset the net/net net/dead net net costs according to the following rules:
  - If new LUW, set to supplier's original unit cost
  - If the same LUW, check if location changed:
    - If new loc:
      - Check if just change from loc-independent to loc-specific. If yes, applied merch level (for 'EX'clusive discount) of loc-independent discounts
      - Check if the flag for 'EX'clusive deal class discount is set (previous 'EX'clusive discount applied)
        - If NO previous 'EX'clusive discount applied, check if this is an 'EX'clusive discount:
          - If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
          - If no, set net/net net/dead net net costs to costs of loc-independent discounts
        - If previous 'EX'clusive discount applied check if this is an 'EX'clusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level 'EX' discount):
          - If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
          - If no, skip this discount.
    - If the same loc, check if the flag for 'EX'clusive deal class discount is set (previous 'EX'clusive discount applied)
      - If NO previous 'EX'clusive discount applied, check if this is an 'EX'clusive discount:
        - If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
        - If no, set net/net net/dead net net costs to latest calculated costs
      - If previous 'EX'clusive discount applied check if this is an 'EX'clusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level 'EX' discount):

  - If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
  - If no, skip this discount.

- Call calculate_costs to calculate net/net net/dead net net costs. For the same LUW + loc, the driving cursor has sorted the discounts by cost_appl_ind: 'N' first, 'NN' later, 'DNN' last. For each cost application level, the same business rules are followed.
- If the new LUW is not in the array, increment the writing index of the cost array (we always write a record into the cost array to keep track of last calculated costs, but change to a new record only if the LUW is changed)
- Prepare an insert record into the deal_sku_cost table by writing costs into the current indexed record of the cost array. There are two dates to consider, start and ending (close_date from deal_head). When inserting the start_date as the active_date, set a flag in the array so we know that's which date it is, and insert the unit_cost from item_supp_country as the base_cost. The location and location type fields should be left NULL if no location was given on deal_sku_temp. Vdate should be used for the calc_date.
- If the start_date is found in the array, calculate the change for each cost field and subtract that change from the net fields in the array. If there is no close date, subtract the change amounts from the net fields of each close date in the array. If we have a close_date and the date found originally in the array was a start_date, subtract the change amounts from the corresponding close_date entry in the array. Find the close_date by looking for the same LUW with the date indicator set to close_date.
- After updating with the start_date, add one to the close date see if that reset_date is already in the array. If not, add it to the array setting the net costs to the base_cost.
- If the reset date is found in the array, set the net costs to the base cost and exit.
- Save current processed LUW and loc.

calculate_costs:

- Inputs: index of fetch array, target threshold value, current net/net net/dead net net costs
- Outputs: calculated net/net net/dead net net costs

The definition of different net costs are:

- net cost = unit cost – components whose cost_appl_ind is 'N'
- net net cost = net cost – components whose cost_appl_ind is 'NN'
- dead net cost = net net cost – components whose cost_appl_ind is 'DNN'

Use the cost_appl_ind on deal_detail to figure out whether a deal component contributes to the net, net net, or dead net net cost (the records should already be sorted by cost_appl_ind) and what the initial costs are (initial cost are need to process 'CU'mulative deal class discounts with 'P'ercentage value type):

- If 'N', the initial net cost is the supplier's original unit cost, and need to update all 3 net costs with the calculated discount
- If 'NN', the initial net net cost is the current net cost, and need to update both net net cost and dead net net cost with the calculated discount.
- If 'DNN', the initial dead net net cost is the current net net cost, and need to update only the dead net net cost with the calculated discount.

Business rules that need to be followed when applying discounts:

- Deal classes:
  - If an exclusive deal was previously found for this SKU/supplier/origin country/start date: new cost should be calculated only if THIS deal is also exclusive and is for a lower merchandise hierarchy. If this is the first exclusive deal, process it and set a flag, saving the hierarchy levels.
  - Cumulative discounts need to be applied to the original unit cost (2% off + 3% off = 5 %off original unit cost)
  - Cascade discounts need to be applied on the result thus far ("current cost")---take2% off of the unit cost, then take 3% off of that price, for example
- Deal value types (take N cost calculation for example):
  - for a % discount
    - If 'CS'cade:
      - discount cost = unit cost – (unit cost *%/100)
    - If 'CU'mulative:
      - discount cost = unit cost – (initial unit cost *%/100)
  - for an amt discount (first convert amount to be in supplier currency if necessary)
    - discount cost = unit cost – amt (amount discounts are per unit cost already)
  - fixed amt: if have fixed amount discount must start with THAT amount rather than the unit cost (convert to supplier currency if necessary)
    - discount cost = fixed amt (converted to supplier's currency if necessary)
  - quantity discount ("buy some get some at discount") (these are not allowed on rebates)
    These are the most complicated. They affect the cost of the get item AND of the buy item, whose cost we also need to get. Both the get item and the buy item will be on deal_itemloc. You should only calculate the cost for whichever item you're presently on (if buy item, just calculate buy item cost; will get the free item separately later, or vice versa). The initial unit cost for the get item should be taken from deal_detail.qty_thresh_free_item_unit_cost (or, if that field is not populated, off of item_supp_country). Before any calculations are done, convert the unit costs into supplier currency if necessary. If a buy/get free discount is encountered, the following things need to happen:

- Call get_unit_cost to get the original unit cost for the buy item (from item_supp_country), if it's different from the free item. Use the supplier and origin country of the free item (free and buy items are required to come from the same supplier and country).
- Calculate the discount costs(for whichever is the current item, free or buy)
  - If qty_thresh_buy_target of the buy item < qty_thresh_buy_qty, stop; you didn't get any discount
  - Otherwise, figure out how many free items you actually get.
    - If the qty_thresh_recur_ind is 'N':
      - free qty = deal_detail.qty_thresh_free_qty
    - If the qty_thresh_recur_ind is 'Y':
      - If buy item = free item:
        free qty = qty_thresh_free_qty *
        FLOOR(qty_thresh_buy_target / (qty_thresh_buy_qty + qty_thresh_free_qty))
      - If buy item different from free item:
        free qty = FLOOR(qty_thresh_buy_target / qty_thresh_buy_qty) *qty_thresh_free_qty
  - If buy item = free item:
    - If qty_thresh_get_type is 'X', this is a "buy/free" discount:
      total discount = total get cost
    - If qty_thresh_get_type is 'P', this is a "buy/get % off" discount:
      total discount = (get item's unit_cost * qty_thresh_get_value / 100) * get qty
    - If qty_thresh_get_type is 'A', this is a "buy/get amt off" discount:
      total discount = qty_thresh_get_value * get qty
    - If qty_thresh_get_type is 'F', this is a "buy/get at fixed amt" discount:
      total discount = (get item's unit_cost - qty_thresh_get_value) * get qty
    - Discount rate = total discount / (buy item unit cost + buy target)
    - Discount = discount rate * get item unit cost
  - If the free item and the buy item are different:
    - If qty_thresh_get_type is 'X', this is a "buy/free" discount:
      total discount = total get cost
    - If qty_thresh_get_type is 'P', this is a "buy/get % off" discount:
      total discount = (get item's unit_cost * qty_thresh_get_value / 100) * get qty
    - If qty_thresh_get_type is 'A', this is a "buy/get amt off" discount:
      total discount = qty_thresh_get_value * get qty
    - If qty_thresh_get_type is 'F', this is a "buy/get at fixed amt" discount:
      total discount = (get item's unit_cost - qty_thresh_get_value) * get qty
    - Get discount rate = (get item cost * get qty) / total buy cost

- Buy get discount rate = 1 – get discount rate
- If current item is buy item
  Discount = total discount * buy discount rate / buy target
- If current item is get item
  Discount = total discount * get discount rate / get qty
- If the total cost of the buy item is less than that of total discount, stop; no discount is applied
- These discounts are the amount that needs to be subtracted from the original price to get the discounted price.

get_target_threshold_value:

Given a deal_id and deal_detail_id, fetch the target value from the deal_threshold table (the value where the target_id is 'Y'). Since this function is often called multiple times for the same input (multiple SKUs of the same deal/deal detail), a linked list is maintained to keep track of target threshold values for different deal/deal detail. The linked list is ordered by the deal/deal detail. This function first tries to get the value from the list (previously fetched from database). If yes, job is done. Otherwise, fetch the target value for this deal/deal detail from database and call convert_currency if the value is currency amount and the deal currency is different from the supplier's currency. The newly fetched value is then saved into the list by calling add_to_list. Other maintenance functions for the linked list are init_list (called in init) and free_list (called in final).

get_unit_cost:

For a given SKU/supplier/country id, get the unit cost from item_supp_country. Since usually the unit cost is fetched by the driving cursor, the function is only called for buy-get type discount when the buy item's unit cost is needed.

convert_currency:

Call CURRENCY_SQL package to convert an amount in deal currency to equivalent amount in supplier's currency. (This should only be called if the currencies are different—normally they will be the same).

post_insert_delete_records:

Array insert all records of the cost array into the deal_sku_cost table and array delete processed records, which are also all records of the cost array, from the deal_sku_temp table. This deletes all records from deal_sku_temp for a given sku/supplier/origin country/start date/location, the unique key of these five columns are part of the unique key on deal_sku_cost, which contains one more column (calc_date) to save the cost information for a system specified history month.

add_to_list:

Add a node made of deal/deal detail and the target value to the current position of the linked list.

init_list:

Initialize the linked list for target threshold values.

free_list:

Free the memory used by the linked list for target threshold values.

size_arrays:

Allocate memory for the fetch array used by the driving cursor and the cost array used to save the costs.

resize_array:

Allocate additional memory for the cost array.

free_arrays:

Free the memory used by the fetch array and cost array.

final:

- Call free_arrays and free_list.
- Restart/recovery close logic.

I/O specification

N/A

Technical issues

There are two rebate_calc_type's: linear and scalar. Currently the scalar type calculation is taken as the same as the linear type. The differentiation is left for future release.

Testing Scenarios

test with:

- item that has 1 active deal
- more than 1 active deal
- multiple deals including an exclusive deal
- different ordering parameters (promo vs annual, earliest vs latest)
- different types of deals

Chapter 6 – Upload RTV Transactions [rtvupld]

Modification

This program was modified by changing the length of the following fields in the upload file layout:

- freight char(20,4)
- Reason char (6)
- Return Quantity char (12,4)
- Unit Cost char (20,4)

Design overview

The RTV Transaction Upload (rtvupld) module processes RTV transactions captured by an external source according to the same logic as the online RTV functionality within the RMS. For each RTV transaction processed by this module, an RTV is created in Shipped status on the RMS database. The RTV transaction can involve any of the supported item types within the Retek system, from any location entity that stocks the transferred item back to the vendor that supplies the item.

The detail processing for an RTV transaction includes the following:

- create RTV header and detail records in shipped status
- update perpetual inventory
- update average cost of shipping location
- write financial transactions for return of merchandise
- update on hand snapshots for current cycle counts (in the case of late postings)
- update unavailable inventory status quantity

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
RAG_SKUS	No	Yes	No	No	No
RAG_SKUS_ST	No	Yes	No	Yes	No
RAG_STYLE_ST	No	Yes	No	No	No
RAG_STYLE_ST	No	Yes	No	No	No
RTV_HEAD	No	Yes	Yes	Yes	No
RTV_DETAIL	No	Yes	Yes	Yes	No
SUPS	No	Yes	No	No	No
ADDR	No	Yes	No	No	No
TRAN_DATA	No	No	Yes	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
V_SKU_INFO	No	Yes	No	No	No
WIN_STORE	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
INV_STATUS_QTY	No	Yes	No	Yes	No
INV_STATUS_TYPES	No	Yes	No	No	No
CODE_DETAIL	No	Yes	No	No	No

Scheduling constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program must run after the Transfer Out batch module and will likely be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of the batch run if pending warehouse transactions exist. It can also be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE and WH – additional threads can be added based on number of distinct input files

Restart recovery

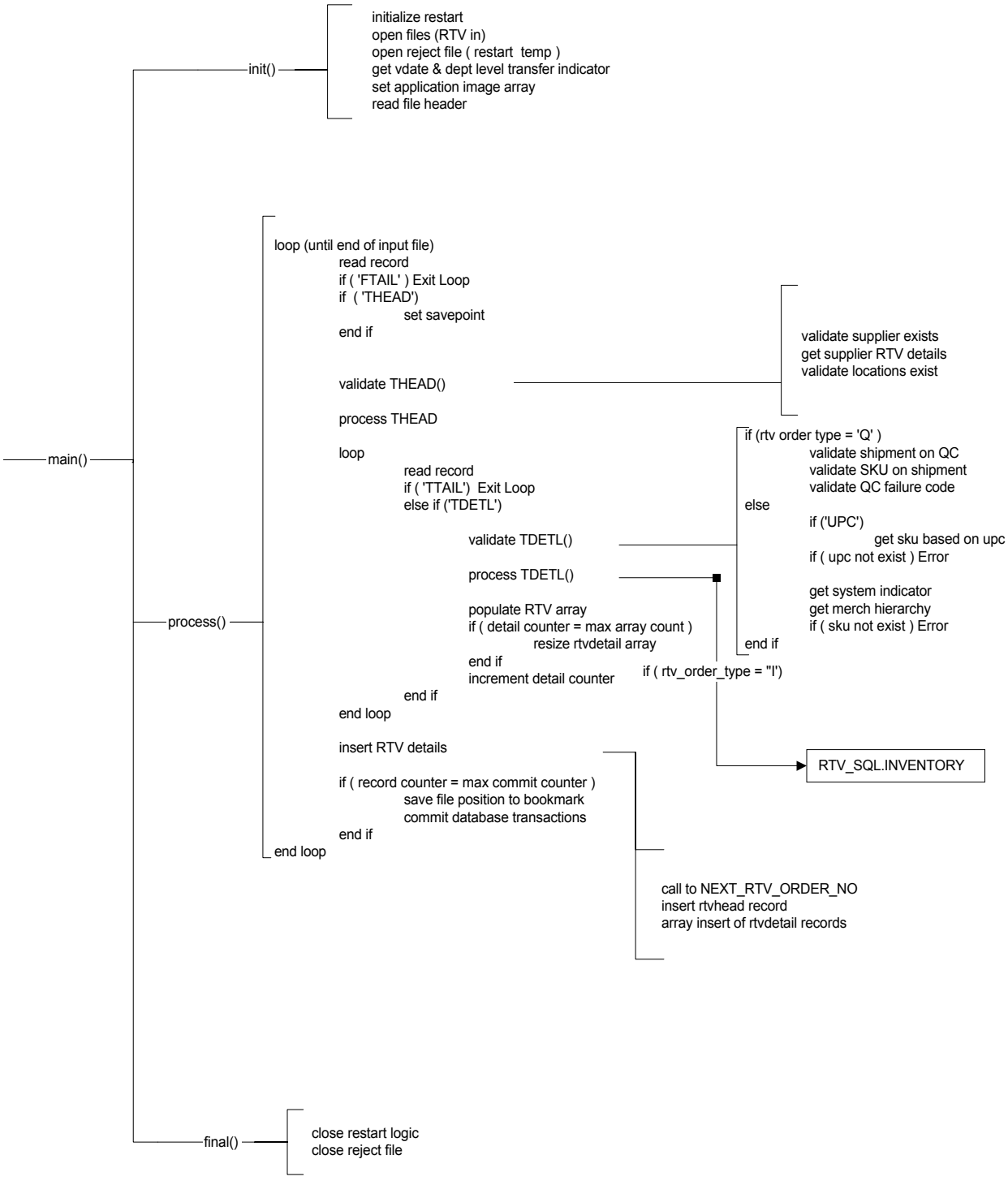
The logical unit of work for the RTV module is the creation of a shipped RTV order in the Retek system. An external reference number created by the external system will identify each RTV. The RTV transaction will be defined as the logical unit of work. If any portion of the processing for the complete RTV transaction fails, the entire RTV must be re-processed.

A save-point will be issued prior to processing a new RTV. If any record within the transaction fails, the whole transaction will be rolled back to the most recent save-point. This way, the successfully processed transactions will remain posted to the database but not yet committed.

To prevent excessive rollback space usage, intermittent commits will be issued based on a commit counter. The recommended commit counter setting is 10000 records (subject to change based on experimentation). The commit counter is based on actual records processed, not overall transactions, nor the number of writes to the database, since the database interactions will be a constant multiplier of the commit counter. An RTV transaction cannot be committed to the database until it is complete so the commit counter is viewed as a minimum threshold, that once reached, will force a commit after the completion of the current RTV transaction.

Error handling will be based on the logical unit of work also. If a given record within an RTV transaction fails, that error will be posted to the standard error log for the batch module. If the error is of a non-fatal type, all subsequent detail records within the RTV will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire RTV will be rejected to a runtime specified rejection file. If a fatal error is encountered, the file pointer at the time of the last commit will have been posted to the bookmark and all transactions from the last commit will be rolled back. Processing will commence with from the saved file position.

Program flow



Shared modules

RTV_SQL.INVENTORY – package performing all RTV logic, including

- update perpetual inventory
- update average cost of returning location
- write financial transactions for the return of merchandise
- update the on hand snapshot for current cycle counts (for late postings)
- update unavailable inventory status quantity

Function level description

init()

- Declare structure array for RTV detail
- Initialize restart recovery
- Open input file (RTV in) – file should be specified as input parameter to the program
- Open reject file (as a temporary file for restart) – file should be Specified as input parameter to the program
- Get vdate from period table
- Set application image array - save the line counter
- Read file header record

process()

Loop

- Read record from input file
- If ('FTAIL')
  - Exit Loop
- Else if ('THEAD')
  - reset detail count
  - set savepoint and save current file pointer position
  - validate_THEAD()
  - process_THEAD()
  - increment line count
- End if
- Loop
  - read record from input file
  - if ('TTAIL')

        - Exit Loop
    - else if ('TDETL')
        - validate_TDETL()
        - process_TDETL()
    - end if
    - if (detail count = max array count)
        - resize array structures for rtvdetail
        - increase max array count
    - end if
    - increment detail count
- End loop
- If (no errors encountered)
    - post_RTV()
- End if
- If (non-Fatal Error encountered)
    - reject_record - call write error and pass file pointer as of last savepoint and current file pointer
- End if
- If (transaction count > max commit count)
    - restart file commit
        - save the current input file pointer position
        - save the line counter in restart image
- End if

End loop

Restart commit final

validate_THEAD()

Validate supplier

- Check for supplier existence on the sups table and ensure returns are allowed
- Select return authorization indicator, minimum dollar amount, and courier from SUPS table.

Validate locations

- If (loc_type = 'ST')
- check for existence on store table
- Else (loc_type = 'WH')
- check for existence on wh table

If the location does not exist, write non-Fatal error.

validate_THEAD()

Call NEXT_RTV_ORDER_NO to get next RTV order number

validate_TDETL()

format_ddetail_fields()

If (Item Type = 'UPC')

- select sku from upc_ean based on the upc and supplement
- if (UPC does not exist)
- Write non-Fatal Error (UPC not found)

If (rtv reason code = 'Q')

- if shipment number found
  - validate shipment exists in Retek and is a QC shipment
  - validate item exists on shipment and has 'Failed' QC
- else
  - write non-Fatal Error (no shipment)

Else if (rtv reason code = 'U')

- if inventory status is not NULL,
  - validate inventory status against INV_STATUS_TYPES table
  - if (inventory status is not found)
    - write non-Fatal Error (invalid inventory status)
  - else
    - validate return quantity <= inventory status quantity for the sku/location/inventory status
    - if (not true)
      - write non-Fatal Error (return qty greater than inventory status quantity)
- else (inventory status is NULL)
  - write non-Fatal Error (no inventory status)

Else (any other reason code or NULL reason code)

- validate reason code against code_detail.code where code_type = 'RTVR'
- if (reason code is not found)
  - write non-Fatal Error (invalid reason code)

Get sku system indicator

If (system indicator does not exist)

- write non-Fatal Error (sku not found)

valid_sku_loc() (validate sku/loc and check if return more than stock on hand)

Validate item/supplier and get item unit cost from item_supp_country table

If (item does not supplied by the supplier)

- write non-Fatal Error

If (unit cost not specified in input file)

- convert unit cost fetched from item_supp_country table into local currency

process_TDETL()

If (shipment number does not exist)

- Call RTV_SQL.INVENTORY package function (see design specification for RTV_SQL) to insert tran_data records and update inventory

Else (shipment number exists)

- Get merchandise hierarchy information
- Call ITEMLOC_ATTRIB_SQL.GET_COST_RETAIL to get unit_retail
- Call STKLEDGER_SQL.TRAN_DATA_INSERT

Convert unit cost back into supplier's currency

Write detail transaction into structure rtvdetail

Calculate total order amount in supplier's currency

post_RTV()

- Perform insert of RTVHEAD
- Perform array insert of RTVDETAIL

ON Fatal Error

- rollback to last physical commit point
- Exit Program

ON Non-Fatal Error

- rollback to last save-point
- write out complete transfer transaction to the reject file, pass file pointer at last save-point and current file pointer

I/O specification

Input File

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

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	RTV	Identifies file as 'Return to Vendor'
	File Create Date	Date	create date	date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	specified by external system	used to force unique transaction check
	Transaction Date	Char(14)	specified by external system	date the transfer was created in external system
	Supplier Identifier	Char(10)	Retek Identifier	Supplier reference identifier as identified within Retek
	Return Authorization	Char(12)	Supplier Specified	Supplier return authorization number

Record Name	Field Name	Field Type	Default Value	Description
	RTV Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location returning items
	RTV Location Value	Char(4)	location identifier	specifies the returning location id number
	Freight	Char(20,4)		Freight cost associated with the RTV in supplier's currency
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPCSKU	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

Record Name	Field Name	Field Type	Default Value	Description
	Shipment	Char(10)	Retek Shipment no	Cross reference to retek shipment number for RTVs associated with QC.
	Reason	Char(6)	Retek reason code	Reason for the return:Q - QC failedU - Unavailable inventory
	Return Quantity	Char(12,4)	return quantity	number of units returned of the given item
	Unit Cost	Char(20,4)	unit cost of return item	assigned cost value of inventory to be returned(in local currency)
	Inventory Status	Char(2)	Retek unavailable inventory status number	Cross reference to Retek Unavailable inventory status number.
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Detail Line Count	Number(6)	sum of detail lines	sum of the detail lines within a transaction

Record Name	Field Name	Field Type	Default Value	Description
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Transaction Line Count	Number(10)	sum of all transaction lines	total of all records less file head and tail records

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the transfer out module and a reject line counter will be 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. The reject filename should be specified as a runtime parameter.

Error File

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

Technical issues

N/A

Chapter 7 – Return to Vendor Upload [lifrtvup]

Modification

This program was modified by changing the length of the following fields in the upload file layout:

- freight char(20,4)
- Return Quantity char (12,4)
- Unit Cost char (20,4)

Design overview

This program will format information originating from the return to vendor file. The Nautilus file, rtv_upload.dat, is SQL Loaded into a staging table: lif_rtv. This program will read from the staging tables and create a standard formatted file for Retek's rtvupld.pc program to process.

Scheduling constraints

Processing Cycle: N/A

Scheduling Diagram: This program should be run after uploading the rtv information from Nautilus and after SQL Loading the files into the staging tables. It should run before rtvupld.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart recovery

No restart/recovery, except for error handling.

Main cursor:

```
SELECT location,
          rtv_order_no,
          SUBSTR(item_id, 1, 13),
          rtv_auth_id,
          unit_qty,
          supplier
     FROM lif_rtv
 ORDER BY location,
          rtv_order_no;
```

Program flow

N/A

Shared modules

N/A

Function level description

Init()

The output file should be opened, then file header information should be written.

The vdate is selected from the period table for the file create date used in the output file header.

Call restart init.

Write output file header (FHEAD) information

Process()

Initialize the RTV number variable to NULL.

Loop through the records found on the lif_rtv table. Array processing should be used to fetch the records from lif_rtv table.

- If RTV order number changes or if the location changes. (ie. one THEAD for each rtv number)
  - Write TTAIL for previous RTV
  - Set default values for THEAD record.
  - Write THEAD record.
  - Set default values for TDETL record.
  - Write TDETL record to the output file for each sku in the RTV.
- End If;

End loop;

final()

Close output file.

Delete from lif_rtv table if all no failures occurred.

Write output file trailer (FTAIL) information.

Call restart close.

I/O specification

Output File

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

RTV upload file

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	RTV	Identifies file as 'Return to Vendor'
	File Create Date	Date	create date	date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	specified by external system	used to force unique transaction check
	Transaction Date	Char(14)	specified by external system	date the transfer was created in external system
	Supplier Identifier	Char(10)	Vendor_nbr	Supplier reference identifier as identified within Retek

Record Name	Field Name	Field Type	Default Value	Description
	Return Authorization	Char(12)	Rtv_auth_nbr	Supplier return authorization number
	RTV Location Type	Char(2)	WH – warehouse	specifies the type of location returning items
	RTV Location Value	Char(4)	Location id	specifies the returning location id number
	Freight	Char(20,4)	NULL	Freight cost associated with the RTV in supplier's currency
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	SKU	item type will be represented as an UPC, an SKU

Record Name	Field Name	Field Type	Default Value	Description
	Item Value	Char(13)	item id	the id number of a SKU or UPC
	Supplement	Char(5)	NULL	used to further specify the id of an UPC item
	Shipment	Char(10)	NULL	Cross reference to retek shipment number for RTVs associated with QC.
	Reason	Char(6)	'W'	Reason for the return:Q - QC failedU - Unavailable inventoryW – warehouse initiated RTV from the RTVR code on codes table.
	Return Quantity	Char(12,4)	Unit qty	number of units returned of the given item
	Unit Cost	Char(20,4)	NULL	assigned cost value of inventory to be returned
	Inventory Status	Char(2)	NULL	Cross reference to Retek Unavailable inventory status number.

Record Name	Field Name	Field Type	Default Value	Description
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Detail Line Count	Number(6)	sum of detail lines	sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Transaction Line Count	Number(10)	sum of all transaction lines	total of all records less file head and tail records

Technical issues

N/A