Retek® Merchandising System 9.0.8

Addendum to Operations Guide

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

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., Retek on the Mall, 950 Nicollet Mall, Minneapolis, MN 55403, and the copyright notice may not be removed without the consent of Retek Inc.

Information in this documentation is subject to change without notice.

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.

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

888.61.RETEK (toll free US)
+1 612 587 5000

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Printed in the United States of America.

Customer Support

Customer Support hours:

8AM to 5PM Central Standard Time (GMT-6), Monday through Friday, excluding Retek company holidays (in 2002: Jan. 1, May 27, July 4, July 5, Sept. 2, Nov. 28, Nov. 29, 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: +1 612-587-5000
Fax	(+1) 612-587-5100
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 Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

Chapter 1 – Introduction 1

Chapter 2 – Transfer Shipments Upload [tsfoupld] 3

Modification 3

Design overview 3

Scheduling constraints 4

Restart recovery 4

Program flow 6

Shared modules 6

Function level description 7

I/O specification 13

Input file 13
Output file 17
Detail 18
Reject file 18
Error file 18

Technical issues 18

Chapter 3 – Monthly Stock Ledger Processing [salmth] ... 19

Modification 19

Design overview 19

Scheduling constraints 20

Restart recovery 21

Program flow 22

Shared modules 22

Function level description 23

I/O specification 24

Technical issues 24

Chapter 4 – Purchase Order Information Written to Order History Tables [ordrev] 25

Modification 25

Design overview 25

Scheduling constraints 25

Restart recovery 26
Program flow 26
Shared modules 26
Function level description 26
I/O specification 30
Order Header file 31
Order Detail file 32
Stock Order file 33
Stock Allocation file 36
Component Ticketing file layout: 38
Technical issues 39

Chapter 5 – Stock Ledger Extract [stlgdnld] 41

Modification 41
Design overview 41
Design assumptions 41
Performance considerations 42
Scheduling constraints 42
Restart recovery 42
Program flow 42
Shared modules 42
Function level description 42
I/O specification 44
Input specifications 44
Output specifications 50
Technical issues 53

Chapter 1 – Introduction

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

- Transfer Shipments Upload [tsfoupld]
- Monthly Stock Ledger Processing [salmth]
- Purchase Order Information Written to Order History Tables [ordrev]
- Stock Ledger [stlgdnld]

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 – Transfer Shipments Upload [tsfoupld]

Modification

Descriptions of THEAD --> Number of Boxes field was reduced from size 15 to 4

Design overview

The purpose of this batch module is to accept transfer shipment details from an external system. The transfer transactions will provide feedback to existing transfers within the Retek system or initiate manual transfers created in an external system. The following functions will be performed for each transferred item:

- create/update transfer and shipment header and detail records.
- create item/location relation for receiving location (if it doesn't exist)
- update perpetual inventory and in transit qtys for source location
- update the average cost of item and in transit qtys for receiving location
- write financial transactions for both the transfer out and the transfer in
- update stock count's snapshot on hand quantity for source location and snapshot in transit quantity for destination location if stock count is in progress
- create/update bill of lading
- create/update warehouse issues history (if transfer from a wh to a store)
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for source location
- update quantity transferred on allocation detail table if this transfer was created from standalone allocation

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	No	Yes	Yes	Yes	No
TSFDETAIL	No	Yes	Yes	Yes	No
SHIPMENT	No	Yes	Yes	Yes	No
SHIPSKU	No	Yes	Yes	Yes	No
POS_MODS	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_STORE	No	Yes	No	Yes	No
RAG_SKUS_ST	No	Yes	No	Yes	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
WIN_WH	No	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No
RAG_SKUS	No	Yes	No	No	No
RAG_STYLE_ST	No	Yes	No	No	No
RAG_STYLE_WH	No	Yes	No	No	No
INV_STATUS_QTY	No	Yes	No	Yes	Yes
INV_STATUS_TYPES	No	Yes	No	No	No

Scheduling constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program must run before the transfer in 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. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available. In a true DC flow through type of operation, this program should also be run after Carton Receiving Upload (ctniupld) module to ship the cross-dock carton transfers created in ctniupld so that the goods received into DC for a cross-dock PO are shipped out to the final destination within the same day.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE and WH

Threads driven by number of distinct files

Restart recovery

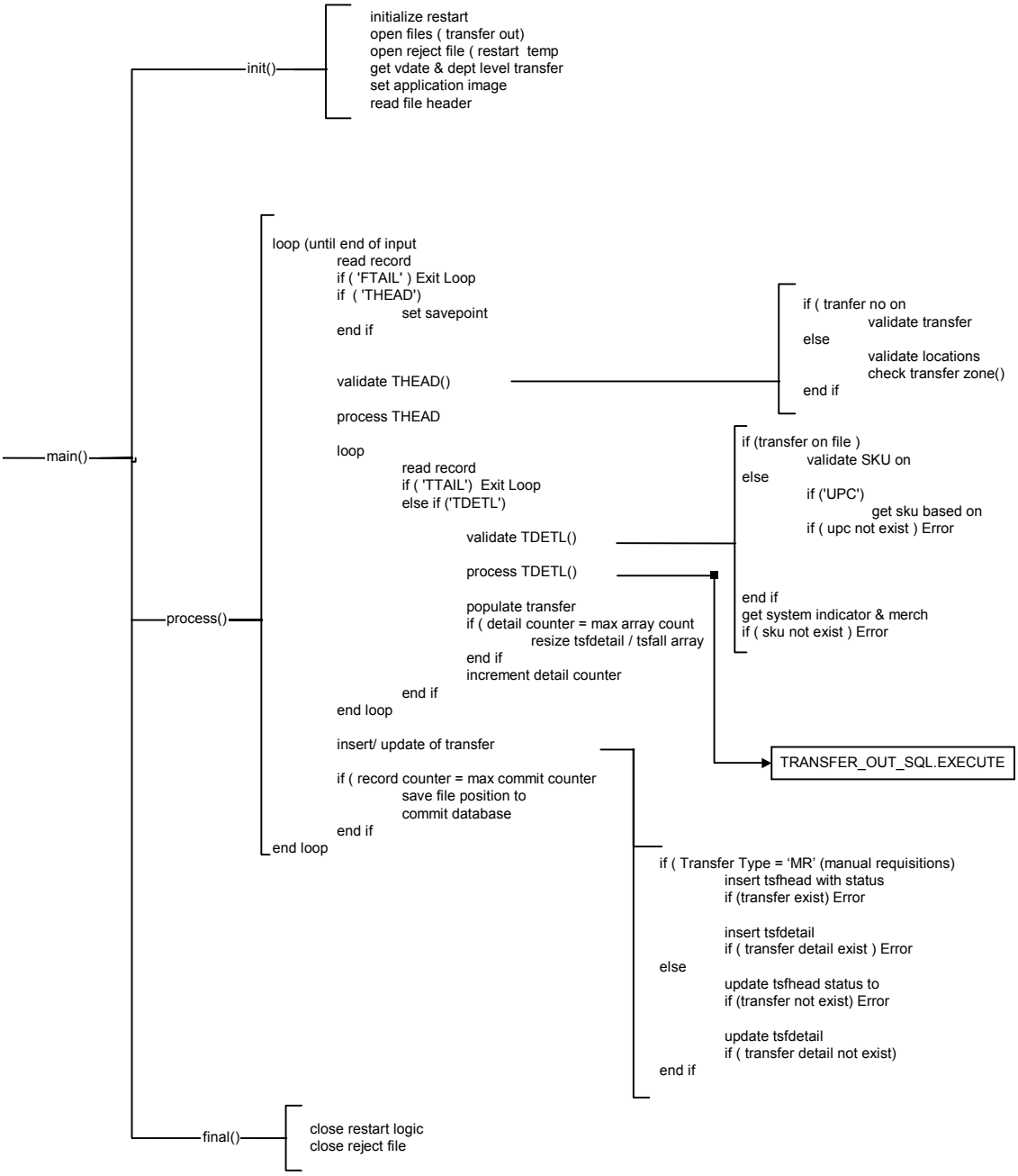
The logical unit of work for the transfer out module is the discrete transfer transaction. Each transfer will be identified by the transfer number (if it already exists in the Retek system) or an unique transaction set number generated by the external system. This transfer transaction will be defined as the logical unit of work. If any portion of the processing for the complete transfer transaction fails, the entire transfer must be re-processed.

A savepoint will be issued prior to processing a new transfer. If any record within the transaction fails, the whole transaction will be rolled back to the most recent savepoint. 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. A transfer 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 transfer transaction.

Error handling will be based on the logical unit of work also. If a given record within a transfer 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 transfer will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire transfer 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

TRANSFER_OUT_SQL.EXECUTE: Package referenced to perform transfer out logic, including:

- create item/location relation for receiving location (if it doesn't exist)
- update perpetual inventory for source location
- update the average cost of item for receiving location

- write financial transactions for both the transfer out and the transfer in
- update stock count's snapshot on hand quantity for source location and snapshot in transit quantity for destination location if stock count is in progress
- create/update bill of lading
- create/update warehouse issues history (if transfer from a wh to a store)
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for source location
- update quantity transferred on allocation detail table if this transfer was created from standalone allocation

TRANSFER_IN_SQL.EXECUTE: Package referenced to perform transfer in logic for customer order types of transfers where the delivery type for the transfer is 'Ship Direct':

- update perpetual inventory for destination location
- update stock count's snapshot on hand quantity for destination location if stock count is in progress
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for destination location
- update perpetual inventory with adjustments for detailed receipt discrepancies and create stock ledger stock adjustment transactions, if system_options.auto_close_tsf = 'Y'

The following are called from TRANSFER_OUT_SQL and/or TRANSFER_IN_SQL packages and are thus, indirect calls.

STOCK_LEDGER_SQL.TRAN_DATA_INSERT: Package referenced by TRANSFER_OUT_SQL.EXECUTE to perform the stock ledger transaction inserts for the transfer out of the goods from the source location and the transfer in of the goods at the destination location.

NEW_STAPLE_LOC, NEW_FASHION_LOC, NEW_PACK_LOC: These stored procedures are used to create item/location relationships for locations that are to receive goods on a transfer and have not yet stocked the given item.

INVADJ_SQL.ADJ_UNAVAILABLE : called to update the unavailable inventory status quantity

INVADJ_SQL.ADJ_TRAN_DATA : called to write tran_data record for unavailable inventory adjustment

Function level description

init()

declare structure arrays for tsfdetail

initialize restart recovery

open input file (transfer out)

- file should be specified as input parameter to program

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

get vdate and department level transfer indicator from period table and system options

set application image array - save the line counter

read file header record

if (record type <> 'FHEAD') Fatal Error

process()

loop

read record from input file

if ('FTAIL')

Exit Loop

end if

if ('THEAD')

set savepoint and save current file pointer position

validate_THEAD()

reset detail count

process_THEAD()

end if

loop

check carton flag to determine if tdetl records will be for a carton or not

read record from input file (different structure for carton or regular)

if ('TTAIL') Exit Loop

if ('TDETL')

validate_TDETL()

process_TDETL()

end if

if (detail count = max array count)

resize array structures for tsfdetail

increase max array count

end if

increment detail count

end loop

if (no errors)

post_transfers() (don't call this if doing a carton)

end if

if (non Fatal Error Encountered)

reject_record - call write error and pass file pointer as of last savepoint and current file pointer

Rollback transaction

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 transfer

-if external shipment number is 'CARTON', set carton flag and return from function

format_header_fields()

if (shipment number provided in transaction)

validate that the shipment number exists within Retek for a transfer. (check on shipment)

validate that the transfer within Retek has a status of 'A', 'E', 'S', 'C' (approved, extracted, shipped, closed) and is applicable to the

to/from locations specified (check on tsfhead) – also fetch transfer type

if shipment number provided does not exist on shipment in 'I', 'R' status for a transfer then

raise Non-Fatal Error

if transfer does not exist in Retek with the appropriate status and locations then

raise Non-fatal error

else if (no shipment number is provided)

if (external shipment number provided)

- validate to and from locations

if (loc_type = 'S')
check for existence on store table
else (loc_type = 'W')
check for existence on wh table
end if
if any location not exist, write non-Fatal error
- validate common transfer zone for store to store transfers
if (to_loc type = 'S' and from_loc = 'S')
check transfer zone - select transfer zone of the from location and the to location.
if (from_loc transfer zone <> to _loc transfer zone)
write non-Fatal Error (transfer zones incompatible)
end if
end if
else (no external shipment number)
All detail records must have an allocation number.
end if
end if

process_THEAD()

check for a bill of lading in 0 - open status for the destination location
retrieve the bill of lading number if one exists
if (bill of lading does not exist)
get next bill of lading number
insert bill of lading header (lad_head) record
end if
if bol number passed in ensure it is valid.
If it is not valid get next bol number.
if transfer type = 'CO'
retrieve delivery type from the ORDCUST table
end if

validate_TDETL()

format_detail_fields()
if inventory status field is not blank, validate it against inv_status_types table
if no shipment / ext shipment in file

every detail line must have an allocation.

if (shipment number in file)

validate item exists on the transfer

else

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)

end if

else if (Item Type = 'SKU')

SKU = item value from the input file

case ID = ' '

end if

end if

if the store rcv type is 'C' the carton field must be populated

- get item system indicator, department, class and subclass

if (system indicator does not exist)

write non-Fatal Error (sku not found)

end if

process_TDETL()

The upd_resv_ind and the upd_intran_ind should be set up in the following way before calling transfer_out_sql.execute.

if :oi_new_tsf_flag = 1 then

if :os_store_rcv_type = 'A' then

L_upd_resv_ind := 'N';

L_upd_intran_ind := 'N';

else

L_upd_resv_ind := 'N';

L_upd_intran_ind := 'Y';

end if;

elsif :ora_tsf_type = 'CO' and :ora_deliver_type = 'S' or

:os_store_rcv_type = 'A' then

L_upd_resv_ind := 'Y';

L_upd_intran_ind := 'N';

else

if :os_tsf_status = 'C' then

L_upd_resv_ind := 'N';

else

L_upd_resv_ind := 'Y';

end if;

L_upd_intran_ind := 'Y';

end if;

call TRANSFER_OUT_SQL.EXECUTE package function

(see design specification for TRANSFER_OUT_SQL)

if transfer type = ‘CO’ and delivery type = ‘S’ or store receive type is ‘A’

call TRANSFER_IN_SQL.EXECUTE package function

(see design specification for TRANSFER_IN_SQL)

write_recs_to_struct()

post_transfers()

if (shipment number was not passed in on the input file)

insert TSFHEAD (transfer_type = ‘MR’ or PO in an allocation is passed in, ext_ref_no = external shipment number)

insert SHIPMENT (ext_ref_no_out should be the transaction control number, ship date should be the transaction date)

perform array insert of TSFDETAIL

perform array insert of SHIPSKU

else (for all other Retek initiated transfer transactions)

try to update shipsku record if no data is found

perform array update of TSFDETAIL, set ship_qty – if transfer type = ‘SA’, set tsf_qty = 0

perform array insert of SHIPSKU

- The this transfer is a customer order (tsf_type = ‘CO’) with a delivery type of direct ship to customer, then this transfer must also be closed when it is sent.

if transfer type = ‘CO’ and delivery type = ‘S’ or store rcv type is ‘A’

call TRANSFER_IN_SQL.CLOSE

(see design specification for TRANSFER_IN_SQL)

else if transfer type = ‘SA’ then

update TSFHEAD status to ‘A’ - approved

else

update TSFHEAD status to ‘S’ - shipped

end if

end if

format_header_fields()

assign input file fields to variables

if from location type = 'ST'

set ora_from_type = 'S'

else if from location type = 'WH'

set ora_from_type = 'W'

end if

if to location type = 'ST'

set ora_to_type = 'S'

else if to location type = 'WH'

set ora_to_type = 'W'

end if

format_detail_fields()

assign input file fields to variables

- transfer quantity has an implied 4 decimal places

transfer qty = transfer qty / 10000

process_carton()

Select details from transfer tables for the carton number; for each sku in the carton, call process_TDETL.

ON Fatal Error

rollback to last physical commit point

Exit Program

ON Non-Fatal Error

rollback to last savepoint

write out complete transfer transaction to the reject file, pass file pointer at last savepoint and current file pointer

I/O specification

Input file

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

Important:

The structure of the TDETL line will vary, depending on whether cartons are included or not. If cartons are included, the line will end after the item value field.

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)	TSFO	Identifies file as 'Transfer OUT'
	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	Number(14)	specified by external system	Used to force unique transaction check
	Transaction Date	Date	specified by external system	Date the transfer was created in external system
	From Location Type	Char(2)	ST - storeWH - warehouse	Specifies the type of location sending items
	From Location Value	Number(4)	location identifier	Specifies the sending location id number
	To Location Type	Char(2)	ST - storeWH - warehouse	Specifies the type of location receiving items
	To Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Shipment Number	Number(10)	Retek shipment number	Specifies the Retek shipment cross-reference

Record Name	Field Name	Field Type	Default Value	Description
	External shipment	Char(15)	External shipment number	Specifies external shipment number; will be CARTON when transferring cartons
	Courier	Char (20)	Courier used to ship order	
	Arrival date	Date	Arrival date	
	Number of boxes	Number(4)		Number of boxes in this transfer
	BOL number	Number(13)	Bill of lading	
Transaction Detail (Item)	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
	Detail Sequence Number	Number(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 a UPC or SKU
	Item Value	Number(13)	item identifier	The ID number of a SKU or UPC
	Supplement	Number(5)	supplemental identifier	Used to further specify the id of an UPC item
	Allocation Number	Char(6) or char(10) if the allocation_ind is = 'Y'.	allocation identifier	Retek allocation number attached to the transfer

Record Name	Field Name	Field Type	Default Value	Description
	Inventory Status	Number(2)	inventory status of item	Used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	carton	Char(20)	carton identifier	UCC – 122 carton code
	Transfer Quantity	Number(12)		Number of units to be transferred of the given item (*10000—4 implied decimal places)
Transaction Detail (Carton)	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
	Detail Sequence Number	Number(6)	specified by external system	Sequential number assigned to detail records within a transaction
	Item Type	Char(3)	CTN	Item type will be represented as a CTN when transferring a carton
	Item Value	Char(20)	carton identifier	UCC – 122 carton code
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

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Detail Line Count	Number(6)	sum of detail lines	Sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Current line number
	Number of transaction lines	Number(10)	specified by external system	Total number of lines in file, excluding FHEAD and FTAIL

Output file

Record Name	Field Name	Field Type	Default Value	Description
	Record Type	Char (1)	H	Specifies file record type
	Store Order Number	Number (10)	Order No	Specifies shipment number
	Division Type	Char (2)	Division Type	Specifies division type
	Warehouse	Number (6)	WH Loc	Specifies WH location value
	Store	Number (6)	Store Loc	Specifies ST location value
	Store Order Type	Number (4)	Store order type	Specifies transfer type
	Store order comment	Char (255)	Comment	Specifies store order comment (from shipment or transfer or both)
	Ship Date	Number (14)	Ship date	Specifies date shipped (date when file was processed + 1)

Detail

Record Name	Field Name	Field Type	Default Value	Description
	Record Type	Char (1)	D	Specifies record type
	Store Order number	Number (10)	Order No	Specifies Shipment Number
	Division type	Char (2)	SA, PO, MR, CO, AD	Specifies Division Type
	Xref Div Item	Number (8)		RMS SKU
	UPC	Number (13)	UPC value	Specifies UPC Value
	UPC supplement	Number (5)	UPC supplement	Specifies UPC supplement value
	Unit of Measure	Char (2)	Unit of Measure	Specifies unit of measure
	SKU Deck Cost	Number (10)	Deck cost	Average unit cost
	Quantity Shipped	Number (6)	Quantity Shipped	Specifies quantity shipped value

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 also 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 3 – Monthly Stock Ledger Processing [salmth]

Modification

Modified the document to indicate that the system_variables table is not updated by salmth.

Design overview

The purpose of this program is to sum up the monthly transaction totals from DAILY_DATA and calculate the closing stock and gross margin for the current month on MONTH_DATA. The procedure varies depending on the following factors:

1 Whether the retail or cost method of accounting is used. Depending on the setting of DEPS.profit_calc_type -- 1 = cost, 2 = retail

2 Whether a stock count of Unit & Dollar type has occurred during the month -- Determined by the presence or absence of a STAKE_PROD_LOC row by dept/class/subclass/store/wh.

Certain checks are made to ensure that the program is being run at an appropriate point in time.

1 The current date (period.vdate) must not be earlier than the next due eom_date (SYSTEM_VARIABLES.next_eom_date)

2 If a stocktake has been done during the month, all stocktake results must have been processed.

Once the timing is verified each subclass/location record on month_data is processed for the current month. For each record fetched, profit calculation type and purchase type are retrieved from deps table, and budgeted shrinkage percent are retreived from half_data_budget table.

If a stock count occurs during the current month, stkdly.pc would have already updated the stock count's book stock and actual stock fields on month_data (i.e. stocktake_bookstk_cost (& retail) and stocktake_actstk_cost (& retail) on month_data). The difference between the book stock and actual stock will be used by this program to adjust the closing stock value for the current month.

In addition, this program calculates a shrinkage amount as follows :

If budget shrinkage indicator = 'Y' :

shrinkage amount = budgeted shrinkage percent * sales amount for the month

else

shrinkage amount = - (stock_adj_cost or retail)

depending on cost or retail method is used

If stock count did not occur during the month,

the above calculated shrinkage amount will be used to reduce the closing stock for this month.

At the same time, this program adds the above calculated shrinkage amount and sales amount for this month into inter_stocktake_shrink_amt and inter_stocktake_sales_amt fields on month_data, which have been accumulated since the last stock count.

Else

inter_stocktake_shrink_amt and inter_stocktake_sales_amt fields will be reset by this program and re-start again to acumulate towards the next stock count.

Note: inter_stocktake_shrink_amt and inter_stocktake_sales_amt fields on month_data are used by stkdly.pc to calculate the book stock value and the actual shrinkage amount for a stock count.

After all threads for this program have finished processing, the prepost module must be launched by the scheduler to update the various end-of-month columns on SYSTEM_VARIABLES.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_DATA	Yes	Yes	No	No	No
DEPS	Yes	Yes	No	No	No
HALF_DATA_BUDGET	Yes	Yes	No	No	No
MONTH_DATA	Yes	Yes	Yes	Yes	No
PERIOD	No	Yes	No	No	No
STAKE_HEAD	Yes	Yes	No	No	No
STAKE_PROD_LOC	Yes	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
V_RESTART	No	Yes	No	No	No

Scheduling constraints

Processing Cycle: PHASE 3 (monthly)

Scheduling Diagram: Can run any time after end-of-month date

Must run salweek first before running salmth

Pre-Processing: N/A

Post-Processing: salmth_post()

Updates system variables to set the stock ledger calendar ahead to the next month for processing. All stock ledger calendar dates are moved forward to indicate that the current month's stock ledger processing has completed.

Threading Scheme: STORE_WH

V_restart_store_wh

Restart recovery

```
SELECT month_data.dept,
       month_data.class,
       month_data.subclass,
       month_data.store,
       month_data.wh,
       month_data.currency_ind,
       NVL(month_data.opn_stk_cost,0),
       NVL(month_data.opn_stk_retail,0),
       NVL(month_data.inter_stocktake_sales_amt,0),
       NVL(month_data.inter_stocktake_shrink_amt,0),
       NVL(month_data.stocktake_mtd_sales_amt,0),
       NVL(month_data.stocktake_mtd_shrink_amt,0),
       NVL(month_data.htd_gafs_cost,0),
       NVL(month_data.htd_gafs_retail,0),
       NVL(month_data.stocktake_bookstk_cost, 0),
       NVL(month_data.stocktake_bookstk_retail, 0),
       NVL(month_data.stocktake_actstk_cost, 0),
       NVL(month_data.stocktake_actstk_retail, 0),
       ';' || TO_CHAR(month_data.dept)     ||
       ';' || TO_CHAR(month_data.class)    ||
       ';' || TO_CHAR(month_data.subclass) ||
       ';' || TO_CHAR(month_data.store)
FROM month_data,
       v_restart_store_wh rv
WHERE month_data.half_no  = :half_no
AND month_data.month_no = :month_in_half
AND rv.driver_value     = month_data.store
AND rv.driver_name      = :ora_restart_driver_name
AND rv.num_threads      = :ora_restart_num_threads
AND rv.thread_val       = :ora_restart_thread_val
AND (month_data.dept>NVL(:ora_restart_dept,month_data.dept- 1)OR
       (month_data.dept = :ora_restart_dept AND
```

```
                (month_data.class > :ora_restart_class OR
                  (month_data.class = :ora_restart_class AND
                    (month_data.subclass  > :ora_restart_subclass OR
                      (month_data.subclass = :ora_restart_subclass AND
                        (month_data.store > :ora_restart_store)))))))
        ORDER BY month_data.dept,
              month_data.class,
              month_data.subclass,
              month_data.store,
              month_data.currency_ind;
```

Program flow

N/A

Shared modules

STKLEDGR_ACCTING_SQL.RETAIL_METHOD_CALC:	performs stock ledger stock and gross margin calculations using the retail accounting method.
STKLEDGR_ACCTING_SQL.COST_METHOD_CALC:	performs stock ledger stock and gross margin calculations using the cost accounting method.
CAL_TO_454_LDOM:	determines the 454 last-day-of-month from current calendar date.
CAL_TO_CAL_HALF:	determines the half number based on current date.
CAL_TO_CAL_LDOM:	determines last-day-of-month based on current date.
CAL_TO_454_WEEKNO:	determines week number in 454 half from current date.
CAL_TO_CAL_WEEKNO:	determines calendar week number from current date.
CAL_TO_454:	determines 454 calendar date from current date.
HALF_TO_CAL_FDOH:	determines the first-day-of-half from the current half number.
HALF_TO_CAL_LDOH:	determines the last-day-of-half from the current half number.
HALF_TO_454_FDOH:	determines the 454 first-day-of-half from current half number.

HALF_TO_454_LDOH:	determines the 454 last-day-of-half from current half number.

Function level description

First check if there are unprocessed "Unit & Dollar" type of stock count, if there are any, stop processing and give user an error message

Main process :

Loop through all subclass/location on month_data for current month (month to be processed). For each month_data record fetched :

read profit_calc_type and purchase_type from deps table

read shrinkage_pct from half_data_budget table

check if Unit & Dollar type of stock count occurs during the month

sums the DAILY_DATA records by transaction type for all records with date in the current month

if purchase_type = 1 (consignment department)

gross_margin_amt = purch_retail - purch_cost

else

if cost method (profit_calc_type = 1)

call the package function stkledgr_accting_sql.cost_method_calc to calculate week's closing stock at cost and gross margin

else

call the package function stkledgr_accting_sql.cost_method_calc to calculate week's closing stock at retail and cost and gross margin

call function update_month_data

Update month_data for current month

Insert a row for next month on month_data

copy current month's closing stock to be opening stock for next month

copy inter_stocktake_shrink_amt and inter_stocktake_sales_amt from current month to next month

if current month_no = 6 (last month of the half) reset GAFS cost and retail :

htd_gafs_cost and retail of next month = cls_stk_cost and retail of current month, respectively

else

copy htd_gafs_cost and retail from current month to next month

I/O specification

N/A

Technical issues

N/A

Chapter 4 – Purchase Order Information Written to Order History Tables [ordrev]

Modification

A note was added to each of the 5 file layouts to specify that the flat files that are created will contain a space between record fields.

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.

Get Allocation Indicator from system_options table.

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

Note: The flat files will contain a space between record fields.

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)	
	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.

Record Name	Record	Default value	Field type	Description
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

Note: The flat files will contain a space between record fields.

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
	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

Record Name	Record	Default value	Field type	Description
	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
	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

Note: The flat files will contain a space between record fields.

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

Record Name	Record	Default value	Field type	Description
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) or char(11) if the allocation_ind is = 'Y'	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_dat e 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
	Event description	Prom_desc or NULL	Char(25)	Event description
	priority	1	Char(4)	Priority
	Order Type	ALLOC_HEADER.ORDE R_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

Record Name	Record	Default value	Field type	Description
	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)	Shipping Address Line 3
	ShipAddressLine 4	NULL	Char(30)	Shipping Address Line 4
	ShipAddressLine 5	NULL	Char(30)	Shipping Address Line 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
	Billing Address 3	NULL	Char(30)	Billing Address Line 3
	Billing Address 4	NULL	Char(30)	Billing Address Line 4

Record Name	Record	Default value	Field type	Description
	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

Note: The flat files will contain a space between record fields.

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) or char(11) if the allocation_ind is = 'Y'.	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:

Note: The flat files will contain a space between record fields.

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.

Record Name	Record	Default value	Field type	Description
	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	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 5 – Stock Ledger Extract [stlgdnld]

Modification

Changed output file due to changes from SIR 29894.

Design overview

This program extracts stock ledger data at a SKU/location/week level from the TRAN_DATA_HISTORY table. The program can extract data for a specified historic period or for the most current complete week. Therefore, if the most current complete week ends on March 10th, running the program on any day between March 11th and 16th will download the week ending March 10th. An historic download will download all the complete weeks between the from and to dates supplied in the input file.

This program will extract the following information at a SKU/location/week level:

Sales Value (retail & cost for regular, promotional and clearance), Sales Units (regular, promotional, and clearance), RTV Value (retail & cost), RTV units, Customer Returns Value (retail & cost), Customer Returns Units, Reclass In (retail & cost), Reclass In Units, Reclass Out (retail & cost), Reclass Out units, Permanent Markdown Value (retail), Promotional Markdown Value (retail), Clearance Markdown Value (retail), Markdown Cancel (retail), Markup Value (retail), Markup Cancel Value (retail), Received Value (retail & cost), Received Units, Transfer In Value (retail & cost), Transfer In Units, Transfer Out Value (retail & cost), Transfer Out Units, Stock Adjustment Value (retail & cost), Stock Adjustment Units, Employee Discount Value, Freight Cost, Cost Variance, Workroom/Other Cost of Sales Value (retail), and Cash Discount value (retail).

Back posted transactions will be downloaded in the week in which the actual transaction occurred. Since the weeks with back posted transactions will only contain additions to the week and not the full week's value – downloaded in a previous extract for the week in which the transaction occurred – the record will be extracted with a 'U' to signify an update for the week. Records for the most current week will be extracted with a 'I' to indicate an insert or overlay for the week (i.e. full weekly data).

Design assumptions

Unit shrinkage will be calculated in RPP. RMS will pass the necessary inventory adjustment records.

Unit BOP will be calculated as the previous period's unit BOP + unit receipts + unit rtvs + unit transfer ins - unit transfer outs + unit reclass in - unit reclass out - net sales - shrinkage. Since the interface will be providing all of these metrics, RPP will calculate the actual unit BOP.

Unit numbers interfaced from RMS to RPP will be in “eaches”.

Performance considerations

Since the data is being extracted from the Transaction Data History table, which is a very large table, performance may be a concern. RMS can not determine at this time the actual performance of this process. See Technical Design for possible performance enhancements.

Scheduling constraints

This program can be run weekly as well as ad hoc (for historic data). This program runs in phase 4.

Restart recovery

This program will use restart recovery. The logical unit of work is each unique SKU.

Program flow

N/A

Shared modules

DATES_SQL.GET_EOW_DATE – Retrieves the end of week date for a specific input date.

Function level description

main()

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

init()

Initialize restart recovery by calling retek_init() and set up the output file.

Fetch the multi_currency_ind, stkldgr_vat_incl_retl_inc, vat_ind, last_eow_date_unit and vdate from system_options, system_variables, and period tables.

format_buffer()

Formats the string that will be used to write to the output file.

process()

Will read the input file and call either fetch_w_process or fetch_h_process according to the input value from command line.

fetch_w_process()

This function will call the driving cursor that fetches the stock ledger data for the most current end of week information from TRAN_DATA_HISTORY. This function will loop through records returned by the driving cursor and write to the output file by calling write_file(). Within the FOR loop, calls the conversion function if the multi_currency_ind = 'Y'. Conversion function will convert the values to primary currency. Eow_date is checked to determined whether the update indicator needs to be set to 'U' or 'I'. Records are written to the output file by using write_file()

fetch_h_process()

This function will call the driving cursor that fetches the stock ledger data for a specific range of historic data from TRAN_DATA_HISTORY. It will also call get_eow_date to determine the end of week date's for the dates that were passed in the input file. This function will loop through records returned by the driving cursor and write to the output file by calling write_file(). A separate record should be written for each SKU/location/eow_date combination. Within the FOR loop, call the conversion function if the multi_currency_ind = 'Y'. This function will convert the values to primary currency. Update indicator will always be set to 'I' for historic runs.

get_eow_date()

Call DATES_SQL.GET_EOW_DATE to determine the end of week dates associated with the from_date and to_date passed in the input file. End of week dates will be used in the fetch_h_process driving cursor to bring back all records on tran_data_history that fall within the historic date range.

write_file()

This function will call rtk_print to write the information fetched from the driving cursor to the output file.

conversion()

Since tran_data_history stores information in the local currency, the values need to be converted to the primary currency. This function will call a C function, CONVERT_TO_PRIMARY, from utils.h and convert the amount values to primary currency if the multi-currency-indicator from system_options is 'Y'. This function will be called within the for loop of both fetch_process functions.

size_array()

Dynamically allocates memory to arrays.

free_array()

Frees memory allocated to arrays.

final()

Take care of file clean up and complete the restart recovery process by calling retek_close() and free_array().

I/O specification

Input specifications

Input file format

H[date in the 'YYYYMMDD' format][date in the 'YYYYMMDD' format] or

W

Ex: H1996010120000222 or W

Complete week driving cursor

If a 'W' is passed in the input file, this driving cursor should be called to bring back all tran_data_history records that fall within the last complete week. Records should be brought back at a SKU/location/week level. The cursor needs to sum the totals for each tran_code based on these variables. Since tran_data_history is kept at a daily level, all days within the given completed week need to be summed up. There are three types of sales that could exist on tran_data_history: regular, promotional, and clearance. Sales need to be broken out by type for each SKU and written to separate fields in the output file for that SKU.

```
EXEC SQL DECLARE c_week_data_w CURSOR FOR
  SELECT tdh.sku,
      DECODE(tdh.store,-1,'W','S'),      /* This is required for currency conversion */
      DECODE(tdh.store,-1,tdh.wh,tdh.store),
      to_date(:ps_eow_date,'YYYYMMDD')-
(7*trunc((to_date(:ps_eow_date,'YYYYMMDD')-
      (DECODE(SIGN(TO_DATE(:ps_eow_date,'YYYYMMDD')-
tran_date),
                  -1,to_date(:ps_eow_date,'YYYYMMDD'),
                  0,to_date(:ps_eow_date,'YYYYMMDD'),
                  TO_DATE(tran_date,'YYYYMMDD'))))/7)),


SUM(DECODE(tran_code,1,DECODE(sales_type,'R',NVL(total_retail,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'R',NVL(total_cost,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'R',NVL(units,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'P',NVL(total_retail,0),0),0)),
```

```
SUM(DECODE(tran_code,1,DECODE(sales_type,'P',NVL(total_cost,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'P',NVL(units,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'C',NVL(total_retail,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'C',NVL(total_cost,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'C',NVL(units,0),0),0)),
        SUM(DECODE(:pi_stkldgr_vat_incl_retl_ind,2,0,
                        DECODE(tran_code,1,NVL(total_retail,0),0))+
                         DECODE(tran_code,2,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,4,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,4,NVL(total_cost,0),0)),
        SUM(DECODE(tran_code,4,NVL(units,0),0)),
        SUM(DECODE(tran_code,24,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,24,NVL(total_cost,0),0)),
        SUM(DECODE(tran_code,24,NVL(units,0),0)),
        SUM(DECODE(tran_code,34,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,34,NVL(total_cost,0),0)),
        SUM(DECODE(tran_code,34,NVL(units,0),0)),
        SUM(DECODE(tran_code,36,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,36,NVL(total_cost,0),0)),
        SUM(DECODE(tran_code,36,NVL(units,0),0)),
        SUM(DECODE(tran_code,13,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,15,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,16,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,14,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,11,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,12,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,22,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,22,NVL(total_cost,0),0)),
        SUM(DECODE(tran_code,22,NVL(units,0),0)),
        SUM(DECODE(tran_code,20,NVL(total_retail,0),0)),
        SUM(DECODE(tran_code,20,NVL(total_cost,0),0)),
```

```
            SUM(DECODE(tran_code,20,NVL(units,0),0)),
            SUM(DECODE(tran_code,30,NVL(total_retail,0),0)),
            SUM(DECODE(tran_code,30,NVL(total_cost,0),0)),
            SUM(DECODE(tran_code,30,NVL(units,0),0)),
            SUM(DECODE(tran_code,32,NVL(total_retail,0),0)),
            SUM(DECODE(tran_code,32,NVL(total_cost,0),0)),
            SUM(DECODE(tran_code,32,NVL(units,0),0)),
            SUM(DECODE(tran_code,26,NVL(total_cost,0),0)),
            SUM(DECODE(tran_code,60,NVL(total_retail,0),0)),
            SUM(DECODE(tran_code,70,NVL(total_cost,0),0)),
            SUM(DECODE(tran_code,80,NVL(total_retail,0),0)),
            SUM(DECODE(tran_code,81,NVL(total_retail,0),0)),
            ':'||TO_CHAR(tdh.sku)
       FROM tran_data_history tdh,
            v_restart_dept rv
      WHERE tdh.post_date between (TO_DATE(:ps_eow_date,
'YYYYMMDD') - 6) and TO_DATE(:ps_eow_date, 'YYYYMMDD')
        AND tdh.sku > NVL(:ps_restart_sku, -999)
        AND rv.driver_value = tdh.dept
        AND rv.num_threads  = :pi_num_threads
        AND rv.thread_val   = :pi_thread_val
   GROUP BY sku,
            DECODE(tdh.store,-1,'W','S'),
            DECODE(tdh.store,-1,tdh.wh,tdh.store),
            to_date(:ps_eow_date,'YYYYMMDD')-
(7*trunc((to_date(:ps_eow_date,'YYYYMMDD')-
            (DECODE(SIGN(TO_DATE(:ps_eow_date,'YYYYMMDD')-
tran_date),
                        -1,to_date(:ps_eow_date,'YYYYMMDD'),
                        0,to_date(:ps_eow_date,'YYYYMMDD'),
                        TO_DATE(tran_date,'YYYYMMDD'))))/7))
   ORDER BY sku,
            to_date(:ps_eow_date,'YYYYMMDD')-
(7*trunc((to_date(:ps_eow_date,'YYYYMMDD')-
            (DECODE(SIGN(TO_DATE(:ps_eow_date,'YYYYMMDD')-
tran_date),
```

```
                    -1,to_date(:ps_eow_date,'YYYYMMDD'),
                    0,to_date(:ps_eow_date,'YYYYMMDD'),
                    TO_DATE(tran_date,'YYYYMMDD'))))/7));
```

Driving cursor for historic data

```
   EXEC SQL DECLARE c_week_data_h CURSOR FOR
     SELECT tdh.sku,
          DECODE(tdh.store,-1,'W','S'),      /* This is required for currency
conversion */
          DECODE(tdh.store,-1,tdh.wh,tdh.store),
          to_date(:is_to_date,'YYYYMMDD')-
(7*trunc((to_date(:is_to_date,'YYYYMMDD')-
          (DECODE(SIGN(TO_DATE(:is_to_date,'YYYYMMDD')- tran_date),
                    -1,to_date(:is_to_date,'YYYYMMDD'),
                    0,to_date(:is_tp_date,'YYYYMMDD'),
                    TO_DATE(tran_date,'YYYYMMDD'))))/7)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'R',NVL(total_retail,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'R',NVL(total_cost,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'R',NVL(units,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'P',NVL(total_retail,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'P',NVL(total_cost,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'P',NVL(units,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'C',NVL(total_retail,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'C',NVL(total_cost,0),0),0)),

SUM(DECODE(tran_code,1,DECODE(sales_type,'C',NVL(units,0),0),0)),
          SUM(DECODE(:pi_stkldgr_vat_incl_retl_ind,2,0,
                    DECODE(tran_code,1,NVL(total_retail,0),0))+
                    DECODE(tran_code,2,NVL(total_retail,0),0)),
          SUM(DECODE(tran_code,4,NVL(total_retail,0),0)),
          SUM(DECODE(tran_code,4,NVL(total_cost,0),0)),
```

```
SUM(DECODE(tran_code,4,NVL(units,0),0)),
SUM(DECODE(tran_code,24,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,24,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,24,NVL(units,0),0)),
SUM(DECODE(tran_code,34,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,34,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,34,NVL(units,0),0)),
SUM(DECODE(tran_code,36,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,36,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,36,NVL(units,0),0)),
SUM(DECODE(tran_code,13,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,15,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,16,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,14,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,11,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,12,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,22,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,22,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,22,NVL(units,0),0)),
SUM(DECODE(tran_code,20,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,20,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,20,NVL(units,0),0)),
SUM(DECODE(tran_code,30,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,30,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,30,NVL(units,0),0)),
SUM(DECODE(tran_code,32,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,32,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,32,NVL(units,0),0)),
SUM(DECODE(tran_code,26,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,60,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,70,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,80,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,81,NVL(total_retail,0),0)),
':'||TO_CHAR(tdh.sku)
FROM tran_data_history tdh,
```

```
        v_restart_dept rv
    WHERE tdh.post_date between (to_date(:is_from_date,'YYYYMMDD')-6)
and (to_date(:is_to_date,'YYYYMMDD'))
      AND tdh.sku > NVL(:ps_restart_sku, -999)
      AND rv.driver_value = tdh.dept
      AND rv.num_threads  = :pi_num_threads
      AND rv.thread_val   = :pi_thread_val
  GROUP BY sku,
        to_date(:is_to_date,'YYYYMMDD')-
(7*trunc((to_date(:is_to_date,'YYYYMMDD')-
        (DECODE(SIGN(TO_DATE(:is_to_date,'YYYYMMDD')- tran_date),
                -1,to_date(:is_to_date,'YYYYMMDD'),
                0,to_date(:is_tp_date,'YYYYMMDD'),
                TO_DATE(tran_date,'YYYYMMDD'))))/7)),
        DECODE(tdh.store,-1,'W','S'),
        DECODE(tdh.store,-1,tdh.wh,tdh.store)
  ORDER BY sku;
```

Following is a list of the transaction codes that will be rolled up to a week level:

01 - Net Sales (retail & cost)

02 - net sales (retail & cost) - retail is always VAT exclusive, written only if system_options.stkldgr_vat_incl_retl_ind = Y

04 - Customer Returns (retail & cost)

11 - Markup (retail only)

12 - Markup cancel (retail only)

13 - Permanent Markdown (retail only)

14 - Markdown cancel (retail only)

15 - Promotional Markdown (retail only)

16 - Clearance Markdown (retail only)

20 - Purchases (retail & cost)

22 - Stock adjustment (retail & cost)

24 - RTV from inventory (retail & cost)

26 - Freight (cost only)

30 - Transfers IN (retail & cost)

32 - Transfers OUT (retail & cost)

34 - Reclassifications In

36 - Reclassifications Out

60 - Employee discount (retail only)

70 - Cost Variance

80 - Workroom/Other Cost of Sales (retail only)

81 - Cash discount (retail only).

Output specifications

Output files

The output will be named stckldgr.dat. A separate record should be written for each SKU/location/eow_date combination.

Output file format

Record Name	Field Name	Field Type	Default Value	Description
	SKU	Char(20)		SKU associated with stock ledger date
	Location Type	Char(1)		Type of Location – S or W
	Location	Char(20)		Store or warehouse of stock ledger data
	EOW Date	Char(8)		End of week date of week for which data was derived
	Update Indicator	Char(1)		It is set to 'U' if the tran_date in weekly run is between end of month date and last end of week date and in all other cases (including historic run) is set to 'I'
	Regular Sales Retail	Number(25)		Total regular sales value (retail) for SKU/location/week – tran_code = 1 and sales_type = 'R'
	Regular Sales Cost	Number(25)		Total regular sales value (cost) for SKU/location/week - tran_code = 1 and sales_type = 'R'
	Regular Sales Units	Number(17)		Total regular sales units for SKU/location/week- tran_code = 1 and sales_type = 'R'
	Promotional Sales Retail	Number(25)		Total promotional sales value (retail) for SKU/location/week - tran_code = 1 and sales_type = 'P'

Record Name	Field Name	Field Type	Default Value	Description
	Promotional Sales Cost	Number(25)		Total promotional sales value (cost) for SKU/location/week- tran_code = 1 and sales_type = 'P'
	Promotional Sales Units	Number(17)		Total promotional sales units for SKU/location/week - tran_code = 1 and sales_type = 'P'
	Clearance Sales Retail	Number(25)		Total clearance sales value (retail) for SKU/location/week - tran_code = 1 and sales_type = 'C'
	Clearance Sales Cost	Number(25)		Total clearance sales value (cost) for SKU/location/week - tran_code = 1 and sales_type = 'C'
	Clearance Sales Units	Number(17)		Total clearance sales units for SKU/location/week - tran_code = 1 and sales_type = 'C'
	Sales Retail Excluding VAT	Number(25)		Total sales value (retail) excluding VAT for SKU/location/week. If the VAT_IND and STKLDGR_VAT_INCL_RETL_IND on SYSTEM_OPTIONS = 'Y', then this value will come from the tran_code 2 records. It will hold the total retail excluding VAT. The tran_code 1 record will contain retail including VAT. If VAT_IND = 'Y' and STKLDGR_VAT_INCL_RETL_IND = 'N', then the tran_code 1 record will contain retail without VAT and no tran_code 2 record will be written, so this field should be 0. If VAT isn't being used, then this field should contain a 0.
	Customer Returns Retail	Number(25)		Total customer returns value (retail) for SKU/location/week
	Customer Returns Cost	Number(25)		Total customer returns value (cost) for SKU/location/week
	Customer Returns Units	Number(17)		Total customer returns units for SKU/location/week
	RTV Retail	Number(25)		Total Return to Vendor value (retail) for SKU/location/week

Record Name	Field Name	Field Type	Default Value	Description
	RTV Cost	Number(25)		Total Return to Vendor value (cost) for SKU/location/week
	RTV Units	Number(17)		Total Return to Vendor units for SKU/location/week
	Reclass In Retail	Number(25)		Reclass In value (retail) for SKU/location/week
	Reclass In Cost	Number(25)		Reclass In value (cost) for SKU/location/week
	Reclass In Units	Number(17)		Reclass In units for SKU/location/week
	Reclass Out Retail	Number(25)		Reclass Out value (retail) for SKU/location/week
	Reclass Out Cost	Number(25)		Reclass Out value (cost) for SKU/location/week
	Reclass Out Units	Number(17)		Reclass Out units for SKU/location/week
	Permanent Markdown Value	Number(25)		Permanent markdown value (retail)
	Promotional Markdown Value	Number(25)		Promotion markdown value (retail)
	Clearance Markdown Value	Number(25)		Clearance markdown value (retail)
	Markdown Cancel Value	Number(25)		Markdown cancel value (retail)
	Markup Value	Number(25)		Markup value (retail)
	Markup Cancel Value	Number(25)		Markup cancel value (retail)
	Stock Adjustment Retail	Number(25)		Stock Adjustment value (retail) for SKU/location/week
	Stock Adjustment Cost	Number(25)		Stock Adjustment value (cost) for SKU/location/week
	Stock Adjustment Units	Number(17)		Stock Adjustment units for SKU/location/week
	Received Retail	Number(25)		Received value (retail) for SKU/location/week
	Received Cost	Number(25)		Received value (cost) for SKU/location/week

Record Name	Field Name	Field Type	Default Value	Description
	Received Units	Number(17)		Received units for SKU/location/week
	Transfer In Retail	Number(25)		Transfer In value (retail)
	Transfer In Cost	Number(25)		Transfer In value (cost)
	Transfer In Units	Number(17)		Transfer In units
	Transfer Out Retail	Number(25)		Transfer Out value (retail)
	Transfer Out Cost	Number(25)		Transfer Out value (cost)
	Transfer Out Units	Number(17)		Transfer Out units
	Freight Cost	Number(25)		Freight value (cost) for SKU/location/week
	Employee Discount Retail	Number(25)		Employee discount value (retail) for SKU/location/week
	Cost Variance	Number(25)		Cost variance value for SKU/location/week
	Workroom/Other Cost of Sales Retail	Number(25)		Workroom/other costs value for SKU/location/week
	Cash Discount Retail	Number(25)		Cash discount value (retail) for SKU/location/week

Technical issues

N/A