

Retek[®] Merchandising System 10.0



Operations Guide – Volume 4: Batch Designs



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.

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.

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

Retek[®] Merchandising System[™] is a trademark of Retek Inc.

Retek and the Retek logo are registered trademarks of Retek Inc.

©2002 Retek Inc. All rights reserved.

All other product names mentioned are trademarks or registered trademarks of their respective owners and should be treated as such.

Printed in the United States of America.

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

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



Customer Support

Customer Support hours:

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@retек.com

Internet	<u>www.retek.com/support</u> 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

Audit purge process [audprg].....	1
Audit logic information edits [auditsys].....	3
Cost event purge [ccprg].....	5
Customs entry download [cednld]	7
Competitive pricing purge [cmpprg]	13
Competitive shopping list upload [cmpupld]	15
Contract purge [cntrmain].....	25
Contract replenishment – type B [cntrordb].....	28
Contract replenishment – types A, B, C [cntrprss].....	32
Complex deals maintenance [costcalc]	35
Deal close [dealcls].....	44
Deals purge [dealprg]	46
Deal upload [dealupld].....	48
Distro price change publish [distrocpub].....	83
Deal item insert [ditinsrt].....	86
Daily record deletion [dlyprg]	96
Document close [docclose].....	98
Direct store delivery upload [dsdupld].....	100
Increment set system date [dtesys]	108
EDI location address to vendor download [edidladd]	110
EDI contract download [edidlcon]	114
EDI invoice matching debit and credit download [edidldeb]	118
EDI purchase order download [edidlord].....	125
EDI sales and stock on hand report download [edidlprd].....	134

EDI purge [ediprg]	139
EDI supplier order acknowledgements and changes [ediupack]	141
EDI supplier address upload [ediupadd]	149
Supplier availability for contracts upload [ediupavl]	152
New and changed upload from supplier [ediupcat]	155
EDI invoice upload [ediupinv]	167
Forecast sales rebuild [fcstrbld]	179
Forecasted unit sales upload [fcslupld]	181
Forecast by subclass upload [fcslupld_sbc]	183
Forecasting/replenishment [fcstprg]	186
Forecast item sales, issues download [fdaydnld]	188
Daily sales forecast upload [fdayupld]	193
Financials general ledger download 1 [fifgldn1]	195
Financials general ledger download 2 [fifgldn2]	199
Financials general ledger download 3 [fifgldn3]	204
Forecast item sales download [fsadnld]	213
Forecast subclass sales download [fsandnld_sbc]	215
Geocode upload [gcdupld]	218
Generate supplier order numbers [genpreiss]	223
Store grade upload [gradupld]	224
Group, department number download [grpdnld]	226
Sales history rollup by department, class, and subclass [hstbld]	229
Sales history rollup by diff IDs [hstbld_diff]	234
Purge sales history [hstprg]	238
Sales history rollup by diff [hstprg_diff]	240
Weekly stock on hand, retail and average cost values update [hstwkupd]	241

Upload customs tariff files [htsupld]	243
Investment buy calculation [ibcalc]	260
Investment buy explosion/pre-quality [ibexpl]	266
Inventory adjustment purge [invaprg]	270
Close unmatched shipments [invclshp]	272
Invoice post [invcpst]	273
Automatic invoice matching [invmatch]	278
Invoice purge [invprg]	280
Letter of credit download [lcadnld]	283
Location list batch rebuild [lclrbld]	302
Letter of credit amendment download [lcmdnld]	305
SWIFT file conversion [lcmt700.perl]	314
SWIFT file conversion [lcmt707.perl]	330
SWIFT file conversion [lcmt730.perl]	339
SWIFT file conversion [lcmt798.perl]	345
Letter of credit update [lcup798]	353
Letter of credit upload [lcupld]	360
Stock upload conversion [lifestkup]	367
On order download [onorddnld]	370
On order extract [onordext]	373
Order deal discount [orddscent.pc/h]	380
Order purge [ordprg]	382
Purchase order information written to order history tables [ordrev]	388
Order update [ordupd]	391
Outstanding order export download file [otbdlord]	393
Stock ledger data download [otbdlsal]	399

Open to buy download [otbdnld]	408
Open to buy purge [otbprg].....	414
Accept forward limit percentages upload [otbupfwd]	416
New budget data and budget adjustments [otbupld]	422
Clearance pricing POS download [pccdnlld].....	428
Retek clearance pricing update [pccext]	430
Clearance event deletions [pccprg].....	434
Clearance reset pricing POS extract [pccrdnlld].....	436
Retek clearance reset price update [pccrext].....	439
Price change POS extract [pcdnld]	442
Retek price change update [pcext].....	445
Price change impact [pcimpc]	450
Price change overlap [pcovrl]	451
Promotion overlap analysis [pcovrlpq].....	452
Purge price changes [pcprg].....	454
Price transactions download for store zone changes [pctrandn]	456
Price transactions extract for store zone changes [pctranex].....	461
Class level planned sales units acceptance [plncupld]	465
Department level planned sales units acceptance [plndupld].....	470
Subclass level planned sales units acceptance [plnsupld]	475
POS configuration download [poscdnlld].....	480
Point of sale download [posdnld].....	496
Group and department POS download [posgpdld].....	508
Point of sale upload [posupld].....	511
Pre-cost calculation [precostcalc].....	526
Pre- and post-functions for programs [prepost]	533

Promotion impact [primpdep]	542
Promotion impact item [primpsku]	544
Promotion price extract and download [prmext]	546
Promotion price update [prmpcupd]	550
Promotions purge [prmprg]	555
Item level promotions explode [prmxpld]	557
Promotions download to output file [promdnld]	560
Reclassification of item [reclsdly]	565
Replenishment adjustment [repladj]	569
Item requisition extraction [reqext]	571
Replenishment item-location maintenance [rilmaint]	580
Automatic replenishment order approval [rplapprv]	586
Replenishment attribute update [rplatupd]	589
Replenishment build [rplbld]	597
Vendor replenishment extraction [rplext]	602
Replenishment purge [rplprg]	608
Replenishment order [rplspllt]	610
RPM moving average [rpmmovavg]	614
Purge RTV [rtvprg]	617
Sales audit escheated vouchers [saescheat]	619
Sales audit ACH download [saexpach]	624
Sales audit export to ReIM [saexpim]	641
Sales audit export to RDW [saexprdw]	656
Sales audit export to RMS [saexprms]	676
Sales audit export to UAR [saexpuar.pc]	688
Sales audit import adjustment [saimpadj]	694

Sales audit import [saimptlog]	698
Stock ledger append [salapnd]	726
Sales daily [saldly]	728
End of half stock ledger processing [saleoh]	730
Stock ledger & budget tables insert [salins]	732
Sales monthly [salmth]	735
Purge stock ledger transactions [salprg]	739
Stock ledger stage [salstage]	740
Sales weekly [salweek]	742
Sales audit pre-export [sapreexp]	751
Sales audit purge [sapurge]	755
Sales audit rules [sarules]	761
Sales audit store day create [sastdyocr]	763
Sales audit totals [satotals]	765
Retek supplier cost changes update [sccect]	767
Store ship schedule purge [schedprg]	770
Scheduled item maintenance main [sitmain]	772
Location security rebuild [slocrbld]	775
Stock on hand download [sohdnld]	781
Store orders upload [soupld]	783
Product security rebuild [sprdrbld]	788
Stock count shrinkage update [stkdy]	794
Purge stock count [stkprg]	796
Stock count snapshots update [stkupd]	798
Upload stock count results [stkupld]	800
Stock count stock on hand updates [stkvar]	806

Stock count explode [stkxpld]	809
Stock ledger extract [stlgdnld.pc]	811
Store add [storeadd]	817
Supplier constraint scaling [supcnstr.pc]	820
Supplier data amount repository [supmth].....	823
Zone security rebuild [szonrbld].....	825
Ticket output file [tcktdnld]	828
Tax rate POS download [tifposdn].....	833
Transportation upload [tranupld]	836
Transfer purge [tsfprg]	846
Tax rate POS download [txrposdn]	852
Tax rate upload [txrtupld]	856
VAT rate maintenance [vatdlxpl].....	860
Vendor replenished order build [vrplbld].....	863
Wastage adjustment [wasteadj].....	871
Warehouse store assignment [whstrasg]	873

Audit purge process [audprg]

Design Overview

The audit purge process truncates auditing tables based on the purge frequency code specified on the audit table. Valid values for purge frequency include:

- 'D' Daily
- 'W' Weekly
- 'M' Monthly
- 'H' Every six months
- 'Y' Yearly

This program requires special permissions. It must be run by an oracle user that has been granted the following privileges, or be run by a DBA.

- 'drop any table' AND
- 'drop any trigger'

To run this program, a single parameter is required: username/password to log on to Oracle.

For example:

auditsys smithja/pw@sid

This program accesses the following database tables:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
AUDIT_TBL	No	Yes	No	No	No
CALANDAR	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (daily)

Scheduling Diagram: This job will be scheduled depending on the type of information to be audited in the system.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

Logging only. Since has ddl changes, rollbacks are not possible.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

This program requires special permissions. It must be run by an oracle user that has been granted the following privileges, or be run by a DBA.

- 'drop any table' AND
- 'drop any trigger'

Audit logic information edits [auditsys]

Design Overview

This program adds or removes audit logic from a given table. A table that has an audit request raised against it will have an audit table created to hold all audit details for any inserts, updates or deletes of data. This audit table will hold the key values of the master table and the username and date of the audited transaction. Additional fields to be audited may be added or removed at the user's request when the request is made prior to running auditsys. Once the table is created, the auditprg program must be run to remove the table to be removed, and then added to the RMS audit trail again for any changes to the columns being tracked. The audit module will create a database trigger to be applied to the master table.

The audit table and the database trigger are automatically promoted to the database and so this program must be executed by a user that has been granted the following special privileges, or a user that has DBA privileges:

- 'create any table' AND
- 'create any trigger'

This program requires only a username/password to run:

For example:

```
auditsys smithj/rettek@dev8i
```

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
AUDIT_TBL	No	Yes	No	No	No
AUDIT_FLD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
ALL_TABLES	No	Yes	No	No	No
ALL_INDEXES	No	Yes	No	No	No
ALL_TAB_COLUMNS	No	Yes	No	No	No
ALL_IND_COLUMNS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (daily)

Scheduling Diagram: Needs to be scheduled only when new audit info has been requested.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

Used for logging purposes only. Since this program is applying ddl, that ddl cannot be rolled back. If needed, tables and/or triggers should be dropped manually, or use the auditprg.pc program to remove them.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

The audit table and the database trigger are automatically promoted to the database and so this program must be executed by a user that has been granted the following special privileges, or a user that has DBA privileges:

- 'create any table' AND
- 'create any trigger'

Cost event purge [ccprg]

Design Overview

The final pricing batch program is the Cost Change Purge (ccprg) module. This program is responsible for removing old cost changes from the system. Cost changes are removed from the system using the following criteria:

- the status of the cost change is Delete, Canceled, or Extracted
- the status of the price change is Rejected and the effective date of the cost change has met the requirement for the number of days that rejected cost changes are held.

Note: The number of days that rejected price changes are held is determined by the system option, number_days_rejects_held on the UNIT_OPTIONS table.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
COST_SUSP_SUP_DE TAIL	No	No	No	No	Yes
COST_SUSP_SUP_HE AD	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
PRICE_EVENT_SKU	No	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)
 Scheduling Diagram: N/A
 Pre-Processing: N/A
 Post-Processing: N/A
 Threading Scheme: N/A (single threaded)

Restart Recovery

```
SELECT cssh.cost_change
FROM cost_susp_sup_head cssh
WHERE cssh.status in ('D','C','E') OR (cssh.status = 'R'
AND ((TO_DATE(:tomorrow) - cssh.active_date) > :days_reject_held));
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

6 Retek Merchandising System

I/O Specification

N/A

Technical Issues

N/A

Customs entry download [cednld]

Design Overview

This program is used to download custom entry information from the RMS database to brokers. Each night, this program will read all Custom Entry (CE) transactions that are in a Sent status for a broker id. These transactions will be written to a flat file and the status will be changed to Downloaded. One process will run and one flat file will be written per broker.

Scheduling Constraints

Processing Cycle: 2
 Scheduling Diagram: This program must run after cefinal.pc.
 Pre-Processing: N/A
 Post-Processing: N/A
 Threading Scheme: Broker

Restart Recovery

The Logical Unit of Work for the program will be a single row from the customs entry tables. Restart/Recovery will be used for init and commit.

```
SELECT LPAD(ce_id,:oi_len_ce_id,'0'),
       NVL(entry_no,' '),
       NVL(to_char(entry_date,'YYYYMMDDHH24MISS'),' '),
       entry_status,
       NVL(entry_type,' '),
       NVL(entry_port,' '),
       NVL(to_char(summary_date,'YYYYMMDDHH24MISS'),' '),
       NVL(broker_id,' '),
       NVL(broker_ref_id,' '),
       NVL(file_no,' '),
       importer_id,
       import_country_id,
       currency_code,
       LPAD(exchange_rate * 10000000000,:oi_len_exchange_rate,'0'),
       NVL(bond_no,' '),
       NVL(bond_type,' '),
       NVL(surety_code,' '),
       NVL(consignee_id,' '),
       live_ind,
       NVL(batch_no,' '),
       NVL(entry_team,' '),
       NVL(to_char(liquidation_amt * 10000),' '),
       NVL(to_char(liquidation_date,'YYYYMMDDHH24MISS'),' '),
       NVL(to_char(reliquidation_amt * 10000),' '),
       NVL(to_char(reliquidation_date,'YYYYMMDDHH24MISS'),' '),
       NVL(merchandise_loc,' '),
       NVL(location_code,' '),
       ROWIDTOCHAR(rowid),
       ';' || to_char(ce_id)
FROM ce_head
WHERE status = 'S'
      AND broker_id = :os_broker_id
      AND ce_id > NVL(:os_restart_ce_id, -999)
ORDER BY ce_id;
```

Program Flow

N/A

Function Level Description

init

This function will perform standard Retek init() function logic (restart/recovery initialization, opening files, etc.). In addition, this function should select system_options.vdate, call the size_arrays() function to allocate memory for SQL fetch arrays, call the init_buffers() function to format the record strings that are written to the output file.

process

Within a loop, the driving cursor should fetch ce_head records into an array. For each ce_id that is fetched from the driving cursor, functions to retrieve records from ce_shipment. All records do not need to retrieve the comments field. The records for the output file that will be written will have the following hierarchy:

FHEAD

ce_head (THEAD)

ce_shipment (TSHIP)

ce_ord_item (TORDI)

transportation (TBLAW)

transportation (TCONT)

ce_lic_visa (TLICV)

ce_charges (TCHRG)

missing_doc (TMDOC)

FTAIL

After all records have been written to the output file for the CE being processed, write the rowid (retrieved from the driving cursor) to an update array. If the transaction count is equal to or greater than the restart_max_ctr, call the updated_ce_head() function to update the ce_head table. Also, call restart_commit() and restart_file_write().

Process_shipments

This function will perform an array fetch to retrieve information from the ce_shipment table associated to the ce_head record being processed and call write_file() to write the records to the output file. This function should call the process_order_items() function to retrieve all order items associated with the shipment.

Process_order_items

This function will perform an array fetch to retrieve information from the ce_ord_items table associated to the ce_head, ce_shipment record being processed and call write_file() to write the records to the output file. This function should call the process_bl_awb_id() (only if the ce_ord_item.bl_awb_id = 'MULTI'), process_container(), process_license_visa(), process_charges(), and process_missing_docs functions to retrieve all detail records associated with the shipment/order/item.

Process_bl_awb_id

This function will perform an array fetch to retrieve information from the transportation table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_container

This function will perform an array fetch to retrieve information from the transportation table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_license_visa

This function will perform an array fetch to retrieve information from the ce_lic_visa table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_charges

This function will perform an array fetch to retrieve information from the ce_charges table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_missing_docs

This function will perform an array fetch to retrieve information from the missing_doc table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Update_ce_head

This function will perform an array update of the ce_head table, changing the status from 'S'ent to 'D'ownloaded for ce_head records that have been processed. The array size counter should be initialized to zero after the post to the database.

Size_arrays

Initially size all fetch and update arrays to the size of the restart_control.restart_max_ctr (using the calloc function). If the memory cannot be allocated, raise a Fatal error.

Init_buffers

This function will format all output strings to the output file. Every time a string is initialized, it should first be set to NULL to clear it out.

Write_line

This function will write a record to the output file for the given record type passed in as a parameter.

Final

This function should perform standard Retek batch final processing. The restart_final() function should be called, the final output file should be closed and the temporary output file should be closed.

I/O Specification

Output File

The output 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 Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
	File Type Definition	Char(4)	CEDN	Identifies file as 'Customs Entry download'
	File Create Date	Date	Create date	date file was written by external system
CE_HEAD	File Type Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	CE ID	Number(10)	ce_head.ce_id	
	Entry No	Char (15)	ce_head.entry_no	
	Entry Date	Char(14)	ce_head.entry_date	YYYYMMDDHH24MISS format
	Entry Status	Char(6)	ce_head.entry_status	
	Entry Type	Char(6)	ce_head.entry_type	
	Entry Port	Char(5)	ce_head.entry_port	
	Summary Date	Char(14)	ce_head.summary_date	YYYYMMDDHH24MISS format
	Broker ID	Char(10)	ce_head.broker_id	
	Broker Ref. ID	Char(18)	ce_head.broker_ref_id	
	File Number	Char(18)	ce_head.file_no	
	Importer ID	Char(10)	ce_head.importer_id	
	Import Country	Char(3)	ce_head.import_country_id	
	Currency Code	Char(3)	ce_head.currency_code	
	Exchange Rate	Number(20,10)	ce_head.exchange_rate	
	Bond Number	Char(18)	ce_head.bond_no	
	Bond Type	Char(6)	ce_head.bond_type	
	Surety Code	Char(6)	ce_head.surety_code	
	Consignee ID	Char(10)	ce_head.consignee_id	
	Live Indicator	Char(1)	ce_head.live_ind	
	Batch Number	Char(20)	ce_head.batch_no	
	Entry Team	Char(3)	ce_head.entry_team	
	Liquidation Amount	Number(20,4)	ce_head.liquidation_amt	
	Liquidation Date	Char(14)	ce_head.liquidation_date	YYYYMMDDHH24MISS format
	Reliquidation Amount	Number(20,4)	ce_head.reliquidation_amt	
	Reliquidation Date	Char(14)	ce_head.reliquidation_date	YYYYMMDDHH24MISS format
	Merchandise Loc	Char(40)	ce_head.merchandise_loc	
	Location Code	Char(4)	ce_head.location_code	
CE_SHIPMENT	File Type Descriptor	Char(5)	TSHIP	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Vessel ID	Char(20)	ce_shipment.vessel_id	
	Voyage Flt ID	Char(10)	ce_shipment.voyage_flt_id	
	Estimated Departure Date	Char(14)	ce_shipment.estimated_depart_date	YYYYMMDDHH24MISS format
	Vessel SCAC Code	Char(6)	ce_shipment.vessel_scac_code	
	Lading Port	Char(5)	ce_shipment.lading_port	
	Discharge Port	Char(5)	ce_shipment.discharge_port	
	Tran Mode ID	Char(6)	ce_shipment.tran_mode_id	
	Export Date	Char(14)	ce_shipment.export_date	YYYYMMDDHH24MISS

	Import Date	Char(14)	ce_shipment.import_date	YYYYMMDDHH24MISS
	Arrival Date	Char(14)	ce_shipment.arrival_date	YYYYMMDDHH24MISS
	Export Country	Char(3)	ce_shipment.export_country_id	
	Shipment Number	Number(10)	ce_shipment.shipment_no	
CE_ORD_ITEM	File Type Descriptor	Char(5)	TORDI	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Order Number	Number(8)	ce_ord_item.order_no	
	Item	Char(25)	ce_ord_item.item	
	BL AWB ID	Char(15)	ce_ord_item.bl_awb_id	'MULTI' – means multiple airway bills (otherwise a single airway bill will be retrieved)
	Invoice ID	Char(30)	ce_ord_item.invoice_id	
	Invoice Date	Char(14)	ce_ord_item.invoice_date	YYYYMMDDHH24MISS format
	Invoice Amount	Number(20,4)	ce_ord_item.invoice_amt	
	Currency Code	Char(3)	ce_ord_item.currency_code	
	Exchange Rate	Number(20,10)	ce_ord_item.exchange_rate	
	Manifest Item Quantity	Number(12,4)	ce_ord_item.manifest_item_qty	
	Manifest Item Quantity UOM	Char(4)	ce_ord_item.manifest_item_qty_uom	
	Carton Quantity	Number(12,4)	ce_ord_item.carton_qty	
	Carton Quantity UOM	Char(4)	ce_ord_item.carton_qty_uom	
	Gross Weight	Number(12,4)	ce_ord_item.gross_wt	
	Gross Weight UOM	Char(4)	ce_ord_item.gross_wt_uom	
	Net Weight	Number(12,4)	ce_ord_item.net_wt	
	Net Weight UOM	Char(4)	ce_ord_item.net_wt_uom	
	Cubic	Number(12,4)	ce_ord_item.cubic	
	Cubic UOM	Char(4)	ce_ord_item.cubic_uom	
	Cleared Quantity	Number(12,4)	ce_ord_item.cleared_qty	
	Cleared Quantity UOM	Char(4)	ce_ord_item.cleared_qty_uom	
	In Transit Number	Char(15)	ce_ord_item.in_transit_no	
	In Transit Date	Char(14)	ce_ord_item.in_transit_date	YYYYMMDDHH24MISS format
	Rush Indicator	Char(1)	ce_ord_item.rush_ind	
	Related Indicator	Char(1)	ce_ord_item.related_ind	
	Tariff Treatment	Char(10)	ce_ord_item.tariff_treatment	
	Ruling Number	Char(10)	ce_ord_item.ruling_no	
	Do Number	Char(10)	ce_ord_item.do_no	
	Do Date	Char(14)	ce_ord_item.do_date	YYYYMMDDHH24MISS format
	Manufacture ID	Char(18)	ce_ord_item.mfg_id	
BL_AWB_ID	File Type Descriptor	Char(5)	TBLAW	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	BL AWB ID	Char(15)	Transportation.bl_awb_id	
CONTAINER	File Type Descriptor	Char(5)	TCONT	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Container ID	Char(20)	Transportation.container_id	
	Container SCAC Code	Char(6)	Transportation.container_scac_code	
CE_LIC_VISA	File Type Descriptor	Char(5)	TLICV	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	License/Visa Type	Char(6)	ce_lic_visa.license_visa	

12 Retek Merchandising System

			type	
	License/Visa ID	Char(30)	ce_lic_visa.license_visa_id	
	License/Visa Quantity	Number(12,4)	ce_lic_visa.license_visa_qty	
	License/Visa Quantity UOM	Char(4)	ce_lic_visa.license_visa_qty_uom	
	Quota Category	Char(6)	ce_lic_visa.quota_category	
	Net Weight	Number(12,4)	ce_lic_visa.net_weight	
	Net Weight UOM	Char(4)	ce_lic_visa.net_weight_uom	
	Holder ID	Char(18)	ce_lic_visa.holder_id	
CE_CHARGES	File Type Descriptor	Char(5)	TCHRG	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Sequence Number	Number(6)	ce_charges.seq_no	
	Pack Item	Number(25)	ce_charges.pack_item	
	HTS	Char(10)	ce_charges.hts	
	Effect From Date	Char(14)	ce_charges.effect_from	YYYYMMDDHH24MISS format
	Effect To Date	Char(14)	ce_charges.effect_to	YYYYMMDDHH24MISS format
	Component ID	Char(10)	ce_charges.comp_id	
	Component Rate	Number(20,4)	ce_charges.comp_rate	
	Per Count UOM	Char(3)	ce_charges.per_count_uom	
	Component Value	Number(20,4)	ce_charges.comp_value	
MISSING_DOC	File Type Descriptor	Char(5)	TMDOC	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Doc_id	Number(6)	Missing_doc.doc_id	
	Received_date	Date	Missing_doc.received_date	
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)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

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

Competitive pricing purge [cmpprg]

Design Overview

This Competitive Pricing Purge program performs the desired deletions from the competitive price history table and the competitive shopping list table based on system variables. This program also currently deletes completed records from the competitive shopping list table each day. On the system options table, the comp_price_months field will determine how many months competitive price history should be maintained before deletion. The comp_list_days field will determine how long a requested shopping list should remain on the shopping list table if it is not complete by the requested shop date.

This program will be modified to retain the last competitive price history record of an item regardless of how old it is. It will also be modified to base deletion of the comp_shop_list table entirely on the comp_list_days value from the unit_options table, rather than deleting completed records on a daily basis.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Ad Hoc (Daily)

Scheduling Diagram:

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: No multi-threading

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

N/A

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

Function Level Description

All database interactions required and error handling considerations

This program will mirror other purge programs such as hstprg.pc

main:

This is the main routine of the program. The standard Retek main function can be used here.

init:

Retrieve the vdate from the period table along with the comp_price_months and comp_list_days variables from the system options table.

process:

Modify this function to retain the last comp_price_hist record for a given item. In other words, if the last recorded date of an item at the competitor's store falls within the deletion period, exclude it from deletion.

Modify this function to delete from comp_shop_list based only on the comp_list_days from unit_options (i.e. remove the 'OR comp_retail is NOT NULL' clause).

final:

Standard final processing.

I/O Specification

All files layouts input and output

No files.

Technical Issues**Testing Scenarios**

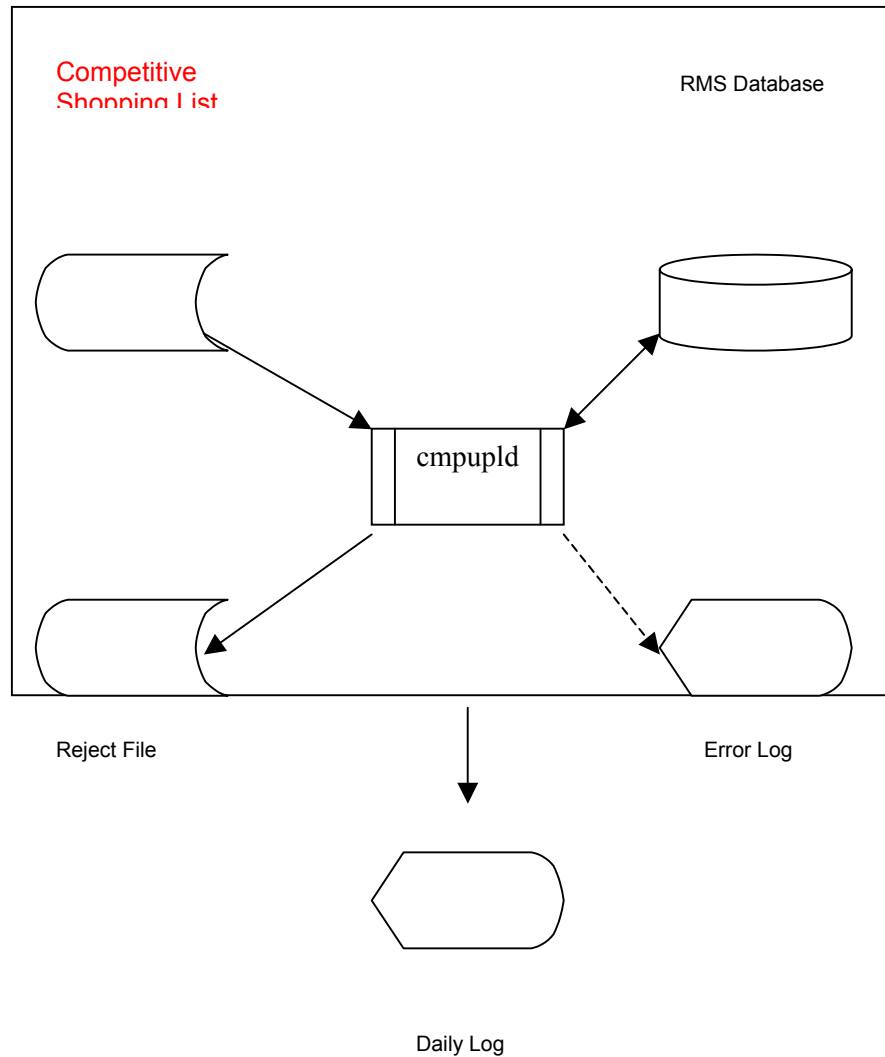
- Run program and make sure the correct fields are being deleted from the comp_price_hist table and the comp_shop_list table.

Competitive shopping list upload [cmpupld]

Design Overview

The purpose of this batch program is to upload the completed competitive shopping list information from a third party into the database tables related to the competitive pricing dialog of the Retek Merchandising System (RMS).

Below is a flow diagram for this process:



The following are the database tables related to the competitive pricing dialog of RMS and the types of access that will be done by the upload program:

Table Name	Select	Insert	Update	Delete
COMP_SHOP_LIST	Yes	Yes	No	No
COMP_PRICE_HIST	No	Yes	No	No
ITEM_MASTER	Yes	No	No	No
COMPETITOR	Yes	No	No	No
COMP_STORE	Yes	No	No	No
COMP_SHOPPER	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
IF_RPM_PRICE_EVENT	No	Yes	No	No

Stored Procedures / Shared Modules (Maintainability)

- Competitor_SQL.Comp_Shopper_Exist - determines if a shopper is in use in the system or not.
- Competitor_SQL.Get_Name - retrieves the competitor name for the passed in competitor id.
- Competitor_SQL.Get_Store_Name - retrieves the competitor's store name for the passed in competitor's store id.
- Item_Attrib_SQL.Get_Desc - returns the description for the entered item.
- Item_Attrib_SQL.Get_Levels - returns the item level and transactional level for the entered item.
- Item_Attrib_SQL.Get_Parent_Info - retrieves the entered item's parent and grandparent along with their descriptions.
- Item_Validate_SQL.Exist - checks for the existence of an item.

Input - Output Specifications

Input File

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	CHAR(5)	"FHEAD"	Value that identifies the record type.
	File Line Identifier	NUMBER(10)	To be specified by the external system	Numeric value that uniquely identifies the current line being processed by input file. This should be right-justified with leading spaces (if any) padded with zeroes.
	File Type Definition	CHAR(4)	"CMPU"	Value that identifies the file as "Competitive Pricing Upload".
	File Create Date	CHAR(14)		Date when the file was written by external system. It should be in the YYYYMMDDHH24MISS format.
File Detail	File Type Record Descriptor	CHAR(5)	"FDETL"	Value that identifies the record type.
	File Line Identifier	NUMBER(10)	To be specified by the external system	Numeric value that uniquely identifies the current line being processed by input file. This should be right-justified with leading spaces (if any) padded with zeroes.
	Shopper Id	NUMBER(4)		Numeric value that uniquely identifies the shopper to which the competitive shopping list is assigned. This should be right-justified with leading spaces (if any) padded with zeroes.

	Shop Date	CHAR(14)		Date when the competitive shopping is performed. It should be in the YYYYMMDDHH24MISS format.
	Item	CHAR(25)		Alphanumeric value that uniquely identifies the transaction level item that was competitively shopped. This should be left-justified with trailing spaces, if any.
	Competitor Id	NUMBER(10)		Numeric value that uniquely identifies a competitor. This should be right-justified with leading spaces (if any) padded with zeroes.
	Competitor Store Id	NUMBER(10)		Numeric value that uniquely identifies a competitor's store. This should be right-justified with leading spaces (if any) padded with zeroes.
	Recorded Date	CHAR(14)		Date when the item's retail price is recorded at the competitor's store. It should be in the YYYYMMDD24MISS format.
	Competitive Retail Price	NUMBER(20,4)		Numeric value that represents the retail price at the competitor's store. There should be four (4) implied decimal places. This should be right-justified with leading spaces (if any) padded with zeroes.
	Competitive Retail Type	CHAR(6)		Value that represents the retail type ("R" is for regular; "P", promotional; and "C", clearance.) that is recorded. This should be left-justified with trailing spaces, if any.
	Promotion Start Date	CHAR(14)	NULL	Effective start date of the competitor's price. It should be in the YYYYMMDDHH24MISS format.
	Promotion End Date	CHAR(14)	NULL	Effective end date of the competitor's price. It should be in the YYYYMMDDHH24MISS format.
	Offer Type Code	CHAR(6)	NULL	Alphanumeric value that corresponds to a valid offer type (e.g., Coupon, Bonus Card, Pre-priced). This should be left-justified with trailing spaces, if any.
	Multi-Units	NUMBER(12,4)		Numeric value that represents the number of units (e.g., 2 for, 3 for) selling for a given amount (Multi-unit retail) if a multiple pricing method was in place for the item when it was competitively shopped. There should be four (4) implied decimal places. This should be right-justified with leading spaces (if any) padded with zeroes.
	Multi-Units Retail	NUMBER(20,4)		Numeric value that represents the amount of all the units selling if a multiple pricing method was in place for the item when it was competitively shopped. There should be four (4) implied decimal places. This should be right-justified with leading spaces (if any) padded with zeroes.
File Trailer	File Type Record Descriptor	CHAR(5)	"FTAIL"	Value that identifies the record type.
	File Line Identifier	NUMBER(10)	To be specified by the external system	Numeric value that uniquely identifies the current line being processed by input file. This should be right-justified with leading spaces (if any) padded with zeroes.
	File Record Counter	NUMBER(10)	To be specified by the external system	Numeric value that represents the number of FDETL records in the file. This should be right-justified with leading spaces (if any) padded with zeroes.

Output Files

Reject File:

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

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

Function Level Description

Here is the main logic flow of the module:

```

Include standard libraries.
Define macros and constants.
Accept and evaluate the runtime parameters, specifically the number of arguments.
If number of parameters < 4
    Display "Usage: cmpupld <username>/<password> <input file> <reject file>"
    End program with a fatal code
End-If
Do LOGON()
If LOGON() < 0
    End program with a fatal code
End-If
Do init()
If init() < 0
    Do LOG_MESSAGE("Thread [%d] - Aborted in init")
    End program with a fatal code
Else-if init() == NO_THREAD_AVAILABLE
    Do LOG_MESSAGE("Terminated OK. No available threads for processing")
    End program
End-If
Do process()
If process() < 0
    Set 1 to gi_error_flag
End-If
Do final()
If final() < 0
    Do LOG_MESSAGE("Thread [%d] - Aborted in final")
    End program with a fatal code
Else
    If gi_error_flag == 0
        If g_l_rej_cnt == 0
            Do LOG_MESSAGE("Thread [%d] - Terminated OK")

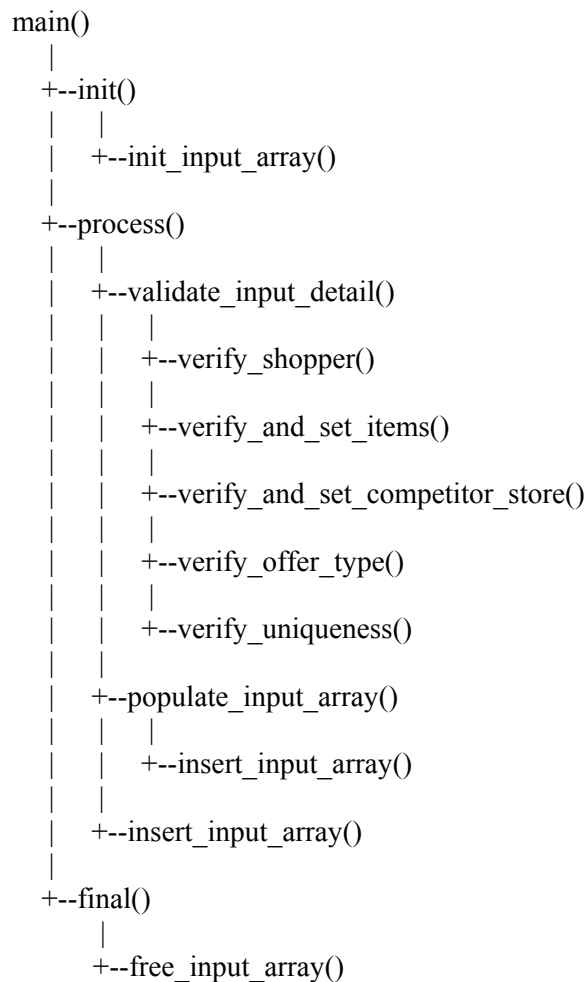
```

```

    Else
        Do LOG_MESSAGE("Thread [%d] - Terminated OK with non-fatal errors")
    End-If
End program
Else
    Do LOG_MESSAGE("Thread [%d] - Aborted while processing")
    End program with a fatal code
End-If
End program

```

The program uses shared modules and other functions. The following structure chart shows how the functions are related to each other:



init() Function

Purpose:

Sets the filenames of the reject and temporary files, initializes the restart-recovery control variables, and then checks if it is a new start or not. If it is a new start, it reads and validates the input file header record.

Called Functions and Shared Modules:

- `init_input_array()`
- `is_new_start()`
- `parse_name_for_thread_val()`
- `retek_get_record()`
- `retek_init()`
- `set_filename()`
- `WRITE_ERROR()`

`init_input_array()` Function

Purpose:

Allocates memory for the input record array, and initializes record index and array size.

Called Functions and Shared Modules:

- `WRITE_ERROR()`

`process()` Function

Purpose:

Reads each input file record, and checks if it is a detail record or a trailer record. If it is a detail record, it validates each record field. If a non-fatal error exists as a result of the validation, a reject file record is created. If there are no errors found, a `COMP_SHOP_LIST` row is inserted into the database (after this, a corresponding `COMP_PRICE_HIST` row will be inserted and `IF_RPM_PRICE_EVENT` row may also be inserted into the database via the `RMS_TABLE_CML_AIUR` trigger) and restart-recovery control variables are updated. The function returns to the main function when a trailer record is read.

Called Functions and Shared Modules:

- `validate_input_detail()`
- `populate_input_array()`
- `insert_input_array()`
- `commit_point_reached()`
- `retek_force_commit()`
- `retek_get_record()`
- `retek_write_rej()`

`validate_input_detail()` Function

Purpose:

Validates all the input data fields. Each data field will be validated as follows:

- *Shopper Id* must be populated, numeric, and must exist in the `COMP_SHOPPER` table.
- *Shop Date* must be populated, and a valid date in the `YYYYMMDDHH24MISS` format.
- *Item* must be populated, must exist in the `ITEM_MASTER` table, and must be at or below the transactional level. Valid items include transaction-level items and reference items.
- *Competitor Id* must be populated, numeric, and must exist in the `COMPETITOR` table.
- *Competitor Store Id* must be populated, numeric, and must exist in the `COMP_STORE` table.
- *Competitor* and *Competitor Store Id* must be a valid combination. The store must belong to the correct competitor.

- *Recorded Date* must be a valid date in the YYYYMMDDHH24MISS format or null.
- *Competitive Retail Price* must be numeric, or null.
- *Competitive Retail Type* must be equal to “R”, “P”, “C” or null.
- *Recorded Date*, *Competitive Retail Price*, and *Competitive Retail Type* must all be null, or all be populated.
- *Promotion Start Date* must be a valid date in the YYYYMMDDHH24MISS format, or null.
- *Promotion End Date* must be a valid date in the YYYYMMDDHH24MISS format, or null.
- *Offer Type Code* must exist in the CODE_DETAIL table with a code type of “OFTP”, or be null.
- *Multi-Units* must be numeric, or null. If it is populated, it must be a non-zero value.
- *Multi-Units Retail* must be numeric, or null.
- *Multi-Units* and *Multi-Units Retail* must both be null, or both be populated.
- *Competitive Retail Price* must be greater than or equal to (*Multi-Units Retail* / *Multi-Units*), if these fields are populated.
- The combination of the *Shopper Id*, *Shop Date*, *Item*, and *Competitor Store Id* must not exist in the COMP_SHOP_LIST table.

Called Functions and Shared Modules:

- verify_shopper()
- verify_and_set_items()
- verify_and_set_competitor_store()
- verify_offer_type()
- verify_uniqueness()
- all_blank()
- OraNumDiv()
- valid_all_numeric()
- valid_date()

verify_shopper() Function

Purpose:

Verifies whether the shopper exists in the system or not.

Called Functions and Shared Modules:

- Competitor_SQL.Comp_Shopper_Exist
- SQL_Lib.Batch_Msg
- WRITE_ERROR()

verify_and_set_items() Function

Purpose:

Verifies whether the item is below or at the transactional level, derives the reference item, and gets the item description. If the item is below the transaction level, then the parent must be retrieved. If the parent is a UPC then a check will need to be performed to see if it already exists on COMP_SHOP_LIST. If it does not exist, then the parent will be written to the item field and the child item will be written to the ref_item field. If the item parent does exist on COMP_SHOP_LIST, then all processing should stop for that record. If an item is at the transaction level and it is a UPC, then the item and ref_item value will be set to the passed in item. Otherwise,

only populate the item field on COMP_SHOP_LIST with the item from the input file.

Called Functions and Shared Modules:

- Item_Attrib_SQL.Get_Desc
- Item_Attrib_SQL.Get_Levels
- Item_Attrib_SQL.Get_Parent_Info
- Item_Validate_SQL.Exist
- SQL_Lib.Batch_Msg
- WRITE_ERROR()

verify_and_set_competitor_store() Function

Purpose:

Verifies the relationship of the store and the competitor in the system. It checks whether the store belongs to the correct competitor or not. Also, it retrieves the competitor name and the competitor's store name from the database tables.

Called Functions and Shared Modules:

- Competitor_SQL.Get_Store_Name
- SQL_Lib.Batch_Msg
- WRITE_ERROR()

verify_offer_type() Function

Purpose:

Verifies whether the offer type exists in the system or not.

Called Functions and Shared Modules:

- WRITE_ERROR()

verify_uniqueness() Function

Purpose:

Verifies whether the combination of the shopper id, shop date, the derived items, and competitor's store id already exists in the system or not.

Called Functions and Shared Modules:

- WRITE_ERROR()

populate_input_array() Function

Purpose:

Populates the input record array with a valid input detail record.

Called Functions and Shared Modules:

- insert_input_array()
- nullpad()

insert_input_array() Function

Purpose:

Inserts the input record array into the COMP_SHOP_LIST table.

Called Functions and Shared Modules:

- WRITE_ERROR()

final() Function

Purpose:

Frees the memory used in the input record array, cleans up the restart-recovery tables, and performs a rollback or final commit.

Called Functions and Shared Modules:

- free_input_array()
- retek_close()

free_input_array() Function

Purpose:

Frees the memory used in the input record array.

Called Functions and Shared Modules:

None.

Scheduling Considerations

Processing Cycle: N/A.

Scheduling Diagram: This upload program should be scheduled to run before any of the Retek Pricing Management (RPM) batch modules.

Pre-Processing: N/A.

Post-Processing: N/A.

Threading Scheme: The default is single threading. The program can be set-up to handle multithreading using one or a combination of the following likely data field candidates as the basis for the thread driver: shopper id, item, competitor id or competitor's store id. The number of threads will be based on the number of input files.

Several factors (e.g., the architecture of the runtime system, the volume of the data, the distribution of data across key discrete data fields) should be considered when choosing a thread driver value.

Locking Strategy

N/A.

Restart/Recovery

The logical unit of work will be a valid shopper shopping an item at a competitor's store on a particular date. The COMMIT_MAX_CTR field on the RESTART_CONTROL table will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups of numbers equal to the COMMIT_MAX_CTR. After a logical unit of work is processed, the restart-commit logic and restart-file-writing logic will be called, and then the next logical unit of work will be processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters), and commit all database transactions. The file-writing logic will append the temporary holding files to the final output files.

The COMMIT_MAX_CTR field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining records will be validated, but the invalid record will be rejected and written to the reject file. If a fatal error is returned, then file processing will end immediately. A restart will be

initiated from the file pointer position saved in the RESTART_BOOKMARK string at the time of the last commit point that was reached during file processing.

Performance Considerations

N/A.

Security Considerations

N/A.

Contract purge [cntrmain]

Design Overview

All contracts with statuses not equal to 'Reviewed' or 'Approved' and their associated records will be deleted after a specified number of months has elapsed since their last change in status if there are no existing orders against it.

In the status maintenance function, 'Approved' closed (types A and B) contracts will be set to 'Complete' automatically. Those that are of type C and D will be set to 'Review' status automatically.

Contracts can also be set to 'Complete' manually in the contract header form. Validation on-line will only check outstanding approved orders and no orders exist in the orders tables. In 'Review' status contracts are still active and it is valid to raise new orders against them (manually or automatically). Open contracts (types C and D) will not automatically be set to 'Complete' or 'Cancelled' as this will be a manual action that takes place as part of the review process. The user will only be able to re-approve a contract in review status using the on-line functionality.

This program accesses the following database tables:

TABLE	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No
CONTRACT_HEADER	Yes	No	Yes	Yes
CONTRACT_SKU	No	No	No	Yes
CONTRACT_PROD_PLAN	No	No	No	Yes
ORDHEAD	Yes	No	No	No

Scheduling Constraints

Processing Cycle: This module will run daily.

Scheduling Diagram: This module needs to be scheduled before the replenishment cycle.

Pre-Processing: None.

Post-Processing: None.

Threading Scheme: None.

Restart Recovery

Contracts pending deletion are read into an array, and all records in contract_cost, contract_detail, and contract_header relating to those contracts are deleted. Transactions are committed after an entire array has been read and deleted. Arrays in this program may need to be smaller than in most other array based restart/recovery programs to control the size of the rollback segment. This is because for each entry in the array records need to be deleted from three tables.

Program Flow

init

 retrieve program variables

process

 call delete_contracts function

 call reset_inactive function

delete_contracts function

 driving cursor LOOP through contracts valid for deletion on contract_header

 if contract status is C (complete) then

 validate no valid orders are attached to contract (do not delete contract if an order exists)

 delete from contract tables

 END LOOP

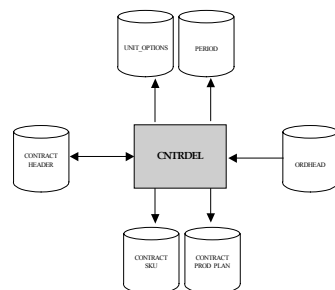
reset_inactive function

 update contract_header

 where contract type is A or B, approved, and passed the end-date then

 update contract_header

 where contract type is C or D, approved, and passed the end-date then

Data Flow**Shared Modules**

N/A

Function Level Description

include std_err header file and use error handling routines in oracle.pc library.

init()

 retrieve system date.

retrieve order_history_months, contract_inactive_months and contract_review_days from unit_options.

process():
 call delete_contracts function
 call reset_inactive function

delete_contracts():
 loop
 driving cursor on contract_header table with the following where clause: status in ('W', 'S', 'C', 'X', 'D') AND months between status_date and today >= order_history_months AND orders do not exist

exit loop if no rows found.

delete from the following tables in the specified order using the contract_no selected from the cursor:

- contract_cost
- contract_detail
- contract_header

commit transaction
 end loop

reset_inactive():

Update contracts where: contract_type is A or B AND approved AND today's date >= end-date then set contract_header.status = 'C', and contract_header.status_date and contract_header.completed_date to today's date.

Update contracts where: contract_type is C or D AND approved AND today's date >= end-date then set contract_header.status = 'R', contract_header.status_date and contract_header.review_date to today's date, and contract_header.review_id to 'AUTOMATIC'.

exception handling: when SQL_ERROR_FOUND, WRITE_ERROR using SQLCODE and return (-1).

end execution block.

I/O Specification

N/A

Contract replenishment – type B [cntrordb]

Design Overview

Batch module creates automatically replenished orders for type B contracts

Orders will be created for all type B contract items that are ready to have orders raised against them. Contract, item, and location information will be selected from the contracting tables where production dates are ready to be met.

An order will be written for each contract and all of the items & locations on the contract.

Scheduling Constraints

Processing Cycle: Phase 3. Must run after rpladjs & rpladjf

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Update of system_variables, set last_cont_order_date = vdate

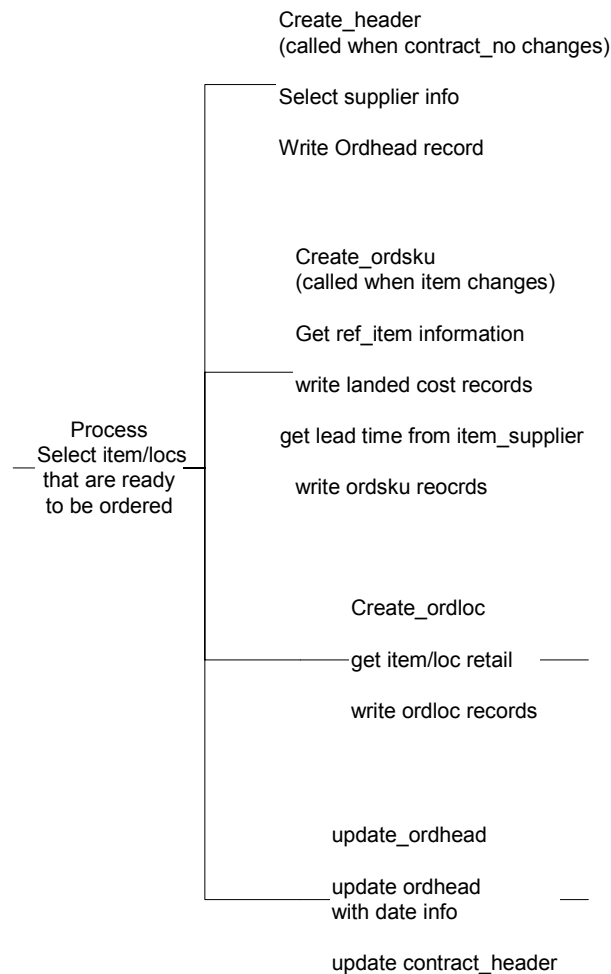
Threading Scheme: Contract_no (questionable whether this is necessary)

Restart Recovery

The logical unit of work is a unique contract. A commit will take place after the number of contracts processed is equal to the max counter from the restart_control table.

Multi-threading issue: If the contract_header table is sizable, the threading mechanism may have to be re-thought or omitted.

Program Flow



Shared Modules

N/A

Function Level Description

Init()

System level variables are selected here including the internal RMS date, the date of the last type B contract ordering run, the minimum number of days before a contracted ready date before an order can be raised, and landed cost information. The restart/recovery process should be initialized.

Process()

Contracted item/locations that are to have orders raised against them are selected from the contracting tables. These are only for contracts of type B, that are within the production plan time frame. An order will be created for each contract, and the **create_header** function will perform the inserts into the RMS ordhead table. After the order has completed, the order header and contract header information is updated by calling the **update_ordhead** function.

The add_ordsku function is called to insert item level information into the RMS ordsku table. If the ELC indicator is “Y”, then each item on the order should be sent to the add_cost_comp function.

The add_ordloc function is called for every record to insert location quantity and retail information into the RMS ordloc table.

create_ordhead()

Insert the order header information into the ordhead table. The NEXT_ORDER_NUMBER stored function is called to retrieve the next available order number for the insert. The package CURRENCY_SQL.GET_RATE is also called to get the exchange rate to use in the insert. The insert is in the form of an insert/select. It joins with the sup_import_attr and addr tables to get the information necessary to create the order.

Add_ordsku()

This function adds item level order records. If the contract does not include reference item information, the supplier’s primary reference item value is retrieved for inserting into the ordering table. The item information, including ref_item, is inserted into the ordsku table. The get_dates function is called to retrieve the lead time for the item and supplier. These dates will be used to determine the not before and not after dates on ordhead. This functions call the item_defaults function.

Item_defaults()

This function calls the stored procedure ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS. This package call defaults all the required documents that are needed when creating ordsku_records.

Update_ordsku()

This function is called whenever processing for a given item, on a given contract is completed. It updates the latest_ship_date with the correct value given the latest ready_date for the item on the contract. The item could have many different ready_dates and the latest ship date should reflect the latest one.

Add_ordloc()

This function first checks if an ordloc record already has been inserted. If a record, exists it updates the ordered qty. If a record does not exist this function adds location level order records. The item/location retail value is retrieved by the get_retail function. The item/location information will be inserted into ordloc. The contracting item-level tables will be updated to reflect that the ordered quantity was ordered against that contract, item, and production plan date. The stored procedure CONTRACT_SQL.GET_UNIT_COST is call to get the unit cost to use in the insert.

Get_retail()

The retail value should be selected from the appropriate item/location (item_loc) table.

Get_dates()

This function determines the most earliest and latest ready dates. These dates are then used when updating ordhead and ordsku.

Add_cost_comp

This function will update the landed cost tables with the new order information. The stored package ORDER_EXPENSE_SQL.INSERT_COST_COMP is called to update the system tables. The package ORDER_HTS_SQL.DEFAULT_CALC_HTS is called to default the hts information for item.

ELC_CALC_SQL.CALC_COMP is called to recalculate expenses.

Update_ordhead()

The information (on ordhead) is updated with the derived not before and not after dates (determined by the item lead times).

Update_ordsku)

The latest ship_date is updated based on the greatest ready_date determined by get_dates().

I/O Specification

N/A

Technical Issues

N/A

Contract replenishment – types A, B, C [cntrprss]

Design Overview

This module processes stock requirements on the ord_temp table by sourcing the stock from the best contract or supplier. It updates the repl_results table to hold information about the qty of the item that is satisfied by the contracts. This module only processes records on ord_temp which are due and do not have an order control of 'B'uyer worksheet. This module will only be run if contracting is turned on in the system. Only system controlled replenishment stock will be integrated with contracting.

Contract type B are not processed in this module. Automatic orders for this contract type are generated in the CNTRORDB module.

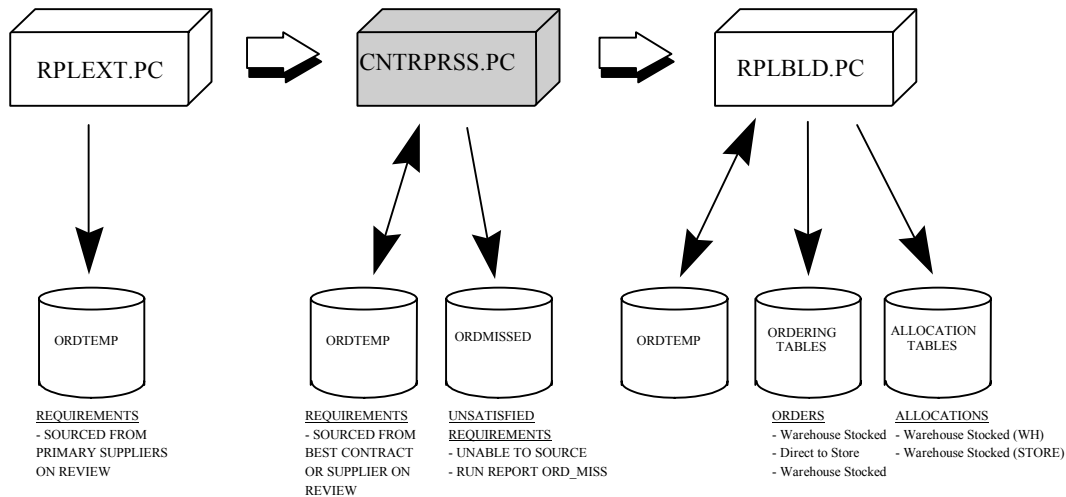
Current replenishment process overview:

The requirements extraction process (rplext) generates requirements for primary suppliers on review. Records are written to the ord_temp and repl_results. The ord_temp table holds requirements detailing the supplier, department, item, quantity required, location, and replenishment stock type (Direct to Store, Cross-dock, Warehouse stocked, Cross-Linked). The order build process (rplbld) takes these requirements and generates orders and allocations.

Contracting integration conceptual overview:

In order to integrate the replenishment process with contracting CNTRPRSS will act as an interpreter between the requirements extraction and order build modules. CNTRPRSS will take requirements (from ord_temp) and source them from the best contract/supplier in the same review cycle. If the contract cannot satisfy the whole requirement, then another contract is used to satisfy the remaining requirement. Any unsatisfied requirement is written to the ord_missed table.

TABLES/FILES ACCESSED	SELECT	INSERT	UPDATE	DELETE
ORD_TEMP	Yes	Yes	Yes	Yes
REPL_RESULTS	Yes	No	Yes	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
CONTRACT_DETAIL	Yes	No	Yes	No
CONTRACT_HEADER	Yes	No	Yes	No
CONTRACT_COST	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
SUPS	Yes	No	No	No
ORD_MISSED	No	Yes	No	No
SUP_AVAIL	Yes	No	Yes	No



Background

1. contract_replenish_ind: The system level `contract_replenish_ind` indicates whether a requirement can be satisfied from outside of a contract or not (i.e., can orders be raised with non-contracted suppliers in the replenishment cycle). This flag is used in CNTRPRSS to determine when requirements should be written to `ord_misseds` as unsatisfied if no contract is available for sourcing.
2. replenishment cycles: Review cycles will be allowed at the vendor level for items replenished from the vendor, and at the store level for items replenished from the warehouse (Warehouse Stocked (W)). The user replenishes items daily, multiple times per week or in a weekly increment (every 2 weeks, every 3 weeks, etc.). Also, the user may specify the week day for review (i.e. Monday, Tuesday, Wednesday...) so that he/she may better balance their workload across the week.
3. rplext: This program is run on a cyclic basis (in conjunction with the programs `rplbld`, `reqext`) in order to maintain optimum stock levels of staple stock items. For further detail see `rplext` design.
4. rplbld: To build orders during the automatic re-order cycle by consolidating the required items as generated in `rplext`. For further detail see `rplbld` design.
5. Replenishment stock categories: There are three stock categories which determine how an item is sourced; Direct to Store (D) where the item flows from the vendor to the store, Cross-docked (C) where the item flows from the vendor to the store, flowing through a distribution centre, or Warehouse Stocked (W) where the item flows from the vendor to a warehouse where it is stocked and sourced at a later date by the stores.
6. Replenishment order control determines whether or not the replenishment will be buyer controlled or system controlled. For system control, orders can be created in worksheet or approved status. For buyer control, requirements can be written to `repl_results` where they can be processed in the buyer worksheet or a report can be run and orders can be created manually.

Detailed Overview

The process will attempt to source the item requirement from the best contract available. The existing ord_temp supplier is then replaced/updated with the contract/supplier details.

The best contract is selected according to the following set of rules (in the rankcntr procedure);

1. earliest - today + lead time
2. cheapest - unit cost (include duty for foreign suppliers if applicable)
3. closed contracts are preferred over open contracts (i.e., A over C & D)
4. contract type C is preferred over contract type D

The selected contracts are validated for the following:

Type	Production plan location	Supplier availability	Total outstanding commitment	Item level commitment	Supplier active & on review
A		X		X	X
C			X		X
D		X	X		X

Any requirement that could not be satisfied by a contract is sourced from the replenishment supplier. If sourcing is not permitted outside of contracts then any unsatisfied requirement is written to ord_missed and not ordered.

Scheduling Constraints

Processing Cycle:

Phase 3

Scheduling Diagram:

This module needs to be scheduled after rplexts and before rplblds.

Pre-Processing:

Rplextpc

Post-Processing:

Prepost (rplbld pre), rplbld

Threading Scheme:

The threads will be split by department.

Restart Recovery

The logical unit of work will comprise the processing of an item requirement on ord_temp. As this can cross many locations, it is a non-unique logical unit of work.

Complex deals maintenance [costcalc]

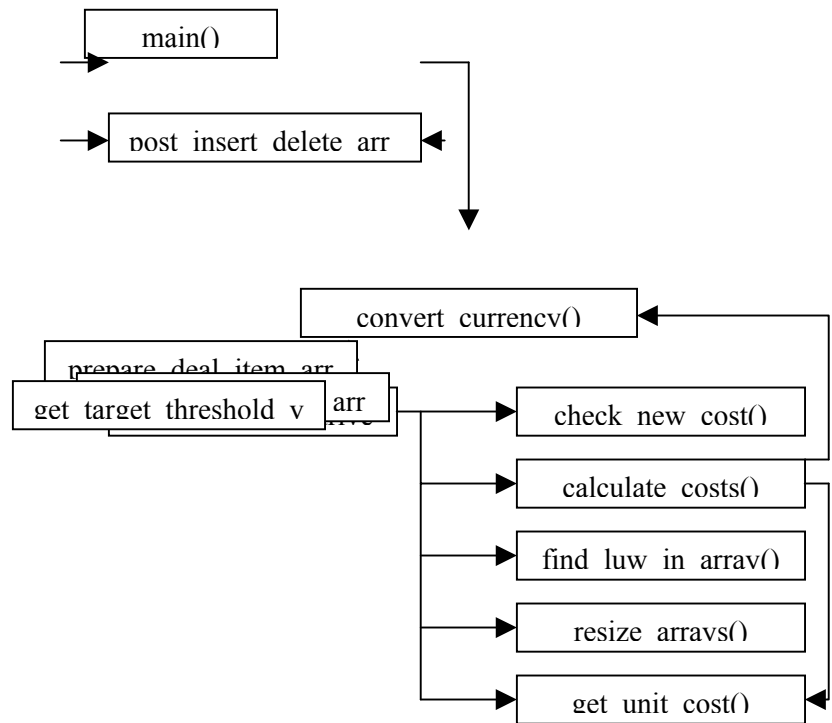
Design Overview

This program is responsible for maintaining information on the FUTURE_COST table based on records on the DEAL_SKU_TEMP table.

This 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). Also items with new item-location relationships, future cost changes or merchandise hierarchy reclassifications or the cancellation of such events will appear here along with appearing on the RECLASS_COST_CHG_QUEUE table. All active deals for each item will be used in the calculation along with any future reclassification or cost change information. Once calculated, the costs will be inserted into the FUTURE_COST table.

Logical unit of work: item/supplier/origin country/location/start date.

Stored Procedures / Shared Modules (Maintainability)

Program Flow**Function Level Description**

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

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 by calling `get_vdate_and_priorities()`.

- Allocate memory for the deal fetch and cost arrays by calling `size_arrays()` and initialize the linked list for deal target values by calling `init_list()`.
- Restart/recovery initialization by calling `retex_init()`.

Process():

- In a while loop:
- Call `get_items()` to array fetch records from the `DEAL_SKU_TEMP` table.
- Loop through the array fetched records and for each:
- Call `get_case_item()` if the simple pack indicator for the current item is Y.
- In a while loop:
- Call `get_deals()` to array fetch deals for the current item.
- Call `prepare_deal_item_array()` to copy the current item and the deals found for the current item into one single array.
- Loop through the records in the array holding the deal item combinations and for each record:
- Call `get_target_threshold()` if the current deal's threshold is not quantity (type Q).
 - Call `calculate_cost_driver()` to get the net, net net, and dead net net cost, 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 `FUTURE_COST` table for each LUW, and to delete processed records from the `DEAL_SKU_TEMP` table.
 - After each set of deals has been processed, call the restart commit logic.
- After loop, call `post_insert_delete_records()` to make sure the final set of records get inserted too.

Get_items():

This function will return data from two separate driving cursors when it is called. The function masks all other db operation requirements from calling entity. The function keeps track of which cursor is being fetched and whether the cursor is open or not. The function will open the active cursor when called if the cursor is not open. The function will array fetch the cursor for data. If no data is found, the program will determine if there was no data at all, no data in this array fetch, or there was data and there was data in this fetch but not to the full array size. Depending on the circumstance, the function will either return a no records found indicator (last cursor fetched and no data found) or will attempt to open/array fetch the next still available cursor. If this second fetch returned records, the function will not return a set indicator for no data found, and therefore 'pretends' that the source is still dumping data.

Get_case_item():

This function will look up if the case item is the primary costing pack for an other item. If there is such an other item, this function gets it.

Get_deals():

This function will open/array fetch one of the four cursors that look up deals for an item base on values fetched from the `system_options` table in the `init ()` function. Signals to calling entity if the 'no data found' indicator was set or not. This function will not re-open a cursor until a previous open has been array fetched to no records found.

Prepare_deal_item_array():

This function will merge the current item with deals that were fetched for the current item into one single array. If the current item is different from the last call to this function, a core element

is inserted into the target array (to maintain functionality requirements. We still need a record for the item in the merged array, even if no deals were found for the item.)

Check_new_cost():

Fetch from RECLASS_COST_CHG_QUEUE to check that the current item/supplier/origin country/location/start date combination has the right cost (which is initially the cost from ITEM_SUPP_COUNTRY_LOC.UNIT_COST, but this may be overwritten by this function if a cost change is found affecting this item/supplier/origin country/location before the item/supplier/origin country/location's start date).

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. Each fetched record is passed on to the calculate_costs function to do the actual calculation for each LUW + deal/deal detail, that is, item/supplier/origin country/location/start date + deal component that applied. For each set of deals for a unique item/supplier/origin country/location/start date, the desired end result is to have one record on FUTURE_COST that will hold the item's costs.

- For each new LUW, 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 merchandise/organization level for 'EX'clusive deal class discount. 'EX'clusive deal class discount should only be applied once for each LUW.
- Reset the net/net net/dead net net costs according to the following rules:
 - If new LUW, set to supplier's original unit cost + RECLASS_COST_CHG_QUEUE cost if future cost change that is earlier than start date of this record is found. This is done by calling check_new_cost()
 - If the same LUW:
 - 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 deal_id and deal_detail_id are not 0, call calculate_costs() to calculate net/net net/dead net net costs.
- For the same LUW, 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. Find this LUW's record in the cost array and update the costs based on whatever new costs were returned from calculate_costs().
- If new LUW, 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). Make sure if current item is a UPC pack, a second record is inserted into the cost array for the component item with cost of the case UPC divided by the pack qty.
- Write costs into the current indexed record of the cost array. Use the start_date from DEAL_SKU_TEMP as the active_date. VDATE should be used for the calc_date.

Calculate_costs(): This function performs the actual calculation of the new 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 is:

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'

pricing cost is the base cost, unless pricing_cost_appl_ind is 'Y' in the deal definition, in which case it will be the dead net cost.

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 needed to process 'CU' cumulative deal class discounts with 'P' percentage 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 needs to be followed when applying discounts:

- Deal classes:
 - 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")---take 2% 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:

$$\text{discount cost} = \text{unit cost} - (\text{unit cost} * \% / 100)$$
 - If 'CU' cumulative:

$$\text{discount cost} = \text{unit cost} - (\text{initial unit cost} * \% / 100)$$
 - for an amt discount (first convert amount to be in supplier currency if necessary)

$$\text{discount cost} = \text{unit cost} - \text{amt} \quad (\text{amount discounts are per unit cost already})$$
 - fixed amt: if have fixed amount discount must start with THAT amount rather than the unit cost (convert to supplier currency if necessary)

$$\text{discount cost} = \text{fixed amt} \quad (\text{converted to supplier's currency if necessary})$$
 - 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:

- Get the get item's initial unit cost. If the buy and get items are the same, then this item's unit cost is the get item's unit cost. If the buy and get items are not the same, then if this item is the get item, use this item's unit cost. Otherwise call `get_unit_cost()` to get the get item's unit cost.
- Next get the buy and get item unit cost. The get item unit cost here is different from the above initial get item unit cost since this unit cost may be set from the `deal_detail.free_item_unit_cost` field if the get item is free (not percent off, or amount off or fixed amount.) So, if this item is the buy item, set the buy item unit cost to this item's unit cost. The get item unit cost is the `deal_detail.free_item_unit_cost` field if the get item is free (not percent off, or amount off or fixed amount), otherwise it is the buy item unit cost if the buy and get items are the same, if they are not the same, call `get_unit_cost()` to get it. If the get item unit cost was populated from `deal_detail.free_item_unit_cost`, call `convert_currency()` for the get item unit cost if the `deal_detail.free_item_unit_cost` was stored in a currency other than the get item's supplier's default currency, which is what this program will use for cost calculations. If this item is the get item, set its cost to the `deal_detail.qty_thresh_free_item_unit_cost` if the get type is free and call `convert_currency()` as necessary, otherwise set its cost to this item's unit cost. If the buy and get items are the same, set the buy item's unit cost to this item's unit cost. Otherwise call `get_unit_cost()` to get it.
- 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; no discounts are applied
 - Otherwise, figure out how many get items we actually get.
 - If the `qty_thresh_recur_ind` is 'N':
 - If buy item is same as get item:
 - `get qty = deal_detail.qty_thresh_free_qty` if buy target $>$ buy qty.
 - Cut off get qty to buy target minus buy qty if necessary.
 - Else:
 - `get qty = deal_detail.qty_thresh_free_qty` if buy target $>$ buy 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))`
 - Add any additional free quantity to get qty where additional free qty = `(qty_thresh_buy_target - (qty_thresh_buy_qty + qty_thresh_free_qty) * FLOOR(qty_thresh_buy_target / (qty_thresh_buy_qty + qty_thresh_free_qty))) - qty_thresh_buy_qty`
 - If buy item different from free item:
 - `free qty = FLOOR(qty_thresh_buy_target / qty_thresh_buy_qty) * qty_thresh_free_qty`
 - 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
- Location specific get qty = get qty * location proportion (which is average location buy qty / buy target)
- If buy item = free item:
 - Get discount share = get qty * get item unit cost / (get qty * get item unit cost + (buy target - get qty) * buy item unit cost)
 - Buy discount share = 1 - get discount share.
 - New get item unit cost = get item initial unit cost - (total discount * get discount share / get qty)
 - New buy item unit cost = current cost of this item - (total discount * buy discount share / (buy target - get qty))
 - If buy item unit cost turns negative, cut its cost off to 0 and push the cut off cost to the get item. If the get item's cost turns negative too, simply cut it off to 0.
 - New item unit cost = ((new get item unit cost * get qty) + new buy item unit cost * (buy target - get qty)) / buy target.
 - Discount = this item's current cost - new item unit cost.
- If the free item and the buy item are different:
 - Get discount rate = (get item cost * get qty) / (buy item unit cost * buy target + get item unit cost * get qty)
 - Buy get discount rate = 1 - get discount rate
 - New get item unit cost = get item initial unit cost - (total discount * get discount rate / get qty)
 - If current item is buy item
New buy item unit cost = this item's current cost - (total discount * buy discount rate / buy target)

If buy item unit cost turns negative, cut its cost off to 0 and push the cut off cost to the get item. If the get item's cost turns negative too, simply cut it off to 0.

Discount = this item's current cost - new buy item unit cost.

- If current item is get item
Discount = this item's unit cost - ((this item's unit cost * (get target - get qty)) + (new get item unit cost * get qty)) / get target
- Final cost = this item's current cost - discount
- If final cost < 0, cut it off to 0.
- Set all cost levels as necessary based on calculated discount and cut them off to 0 as necessary.
- If discount is 0, return indicating that the deal was not applied, otherwise return 0.

Find_luw_in_array(): This function looks up argument item/supplier/origin country/location/start date in the cost array and returns its index if found, -1 if not found.

Get_vdate_and_priorities(): This function retrieves the VDATE from the PERIOD table (use as calculation date for inserts into deal_cost table), gets 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 the SYSTEM_OPTIONS table.

Size_arrays(): Allocate memory for arrays.

Resize_array(): Allocate additional memory for the cost array.

Free_arrays(): Free the memory used by the arrays.

Init_list(): Initialize the linked list for target threshold values.

Free_list(): Free the memory used by the linked list for target threshold values.

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

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_level_ind is 'Y'). Since this function is often called multiple times for the same input (multiple ITEMS 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 found then 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 item/supplier/origin country/location/start date, get the unit cost from ITEM_SUPP_COUNTRY_LOC or RECLASS_COSTCHG_QUEUE depending on whether the latter has a record for the item/supplier/origin country/location/start date or not. If not, look the item/supplier/origin country/location up on the ITEM_SUPP_COUNTRY_LOC table and get the unit costs from there. Since usually the unit cost is fetched by the driving cursor, the function is only called for buy/get type discounts when an item's unit cost is needed.

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

Post_insert_delete_records():

- Array delete all records from the FUTURE_COST table where the delete array's item/supplier/origin country/location matches the record in FUTURE_COST and FUTURE_COST's active date is higher or equal to the start date from the cost array. This step is necessary to prevent primary key violations during insert into the FUTURE_COST table.
- Array insert all records of the cost array into the FUTURE_COST table and array delete processed records, which are also all records of the cost array, from the DEAL_SKU_TEMP table.
- Delete record from DEAL_SKU_TEMP that is associated with the records in the cost array.

Final():

- Call free_arrays() and free_list().
- Call restart/recovery close logic by calling retek_close().

'Table-To-Table'

DEAL_SKU_TEMP			
DEAL_SKU_TEMP			
DEAL_SKU_TEMP			
DEAL_SKU_TEMP			
DEAL_SKU_TEMP			
DEAL_SKU_TEMP			
DEAL_SKU_TEMP			
DEAL_HEAD			
DEAL_HEAD			

Output Specifications**'Table-To-Table'**

The following table will be inserted:

FUTURE_COST			
FUTURE_COST			
FUTURE_COST			
FUTURE_COST			
FUTURE_COST			
FUTURE_COST			
FUTURE_COST			
FUTURE_COST		NUMBER(20,4)	
FUTURE_COST		NUMBER(20,4)	
FUTURE_COST		NUMBER(20,4)	
FUTURE_COST		NUMBER(20,4)	
FUTURE_COST		NUMBER(20,4)	
FUTURE_COST			

Scheduling Considerations

Processing Cycle: Phase II (daily)

Scheduling Diagram: Must be run after ditinsrt.pc, and precostcalc.pc which populate the DEAL_SKU_TEMP table

Threading Scheme: SUPPLIER

Deal close [dealcls]

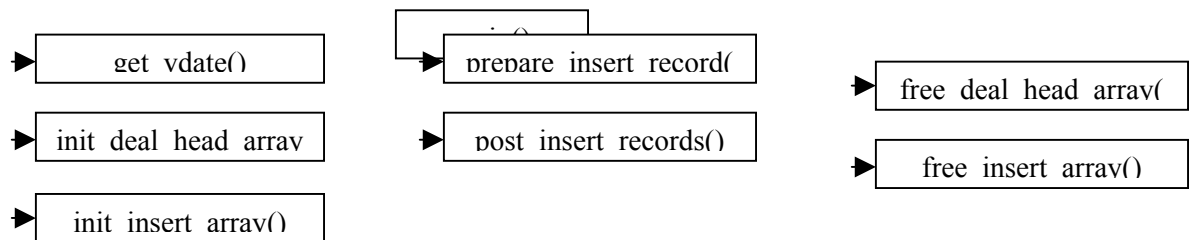
Design Overview

This new batch program will close deals that have reached their end dates. This program will not cover PO-specific deals, which will be maintained with the PO.

The LUW of this module is a single record from the DEAL_HEAD table.

Stored Procedures / Shared Modules (Maintainability)

Program Flow



Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init(): Standard Retek init function. Calls retek_init(), get_vdate(), init_deal_head_array() and init_insert_array().

Process(): This is the main function that does all the work.

- In a while loop, array fetch the DEAL_ID from DEAL_HEAD table for records whose CLOSE_DATE value is less than or equal to the current VDATE and STATUS is 'A'ctive and the deal's TYPE is not 'O' (PO specific).
 - For each DEAL_ID from the DEAL_HEAD table, update the DEAL_HEAD table by setting the STATUS to 'C'losed, and CLOSE_ID to USER.
 - For each DEAL_ID fetched, call prepare_insert_record().
 - Call post_insert_records().
 - Call retek_force_commit().
- Return.

Final(): Standard Retek final function. Calls free_deal_head_array(), free_insert_array(), and retek_close().

Init_deal_head_array(): Allocate memory for the driving cursor fetch array pa_deal_fetch.

Init_insert_array(): Allocate memory for the array pa_deal_insert which will temporarily hold the DEAL_ID for the DEAL_QUEUE table.

Free_deal_head_array(): Release memory for the array pa_deal_fetch.

Free_insert_array(): Release memory for the array pa_deal_insert.

Prepare_insert_record(): Fetch DEAL_ID of current pa_deal_fetch array from the DEAL_QUEUE table. If the DEAL_ID does not exist on the DEAL_QUEUE table, insert the DEAL_ID into the pa_deal_insert array, otherwise do nothing, just return.

Post_insert_records(): Array insert DEAL_ID from the array pa_deal_insert into the table DEAL_QUEUE.

Scheduling Considerations

This program should run after all other deals batch programs, but before dealprg.pc.

Processing Cycle: Phase 3 (daily)

Scheduling Diagram:

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: NONE

Deals purge [dealprg]

Functional Area

Complex Deals Management

Design Overview

This new batch program will purge deals after they have been held in the system for the specified number of history months after they are closed. This program will not cover PO-specific deals, which will be purged with the PO.

The LUW of this module is a single record from the DEAL_HEAD table.

Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init(): Standard Retek init function. Calls retek_init(). Also gets VDATE, DEAL_HISTORY_MONTHS, and FUTURE_COST_HISTORY_DAYS from the SYSTEM_OPTIONS table.

Process(): This is the main function that does all the work.

- Call size_deals_array().
- In a while loop, array fetch the DEAL_ID from DEAL_HEAD table for records whose months between the CLOSE_DATE value and the current VDATE is more than the DEAL_HISTORY_MONTHS from the SYSTEM_OPTIONS table, STATUS is 'C'losed, the deal's TYPE is not 'O' (PO specific), and no records for the DEAL_ID exist on the ORDLOC_DISCOUNT or ORDHEAD_DISCOUNT tables.
 - For each DEAL_ID from the DEAL_HEAD table, delete matching records from the DEAL_QUEUE, DEAL_ITEMLOC, DEAL_THRESHOLD, POP_TERMS_FULFILLMENT, POP_TERMS_DEF, DEAL_DETAIL, and DEAL_HEAD tables.
 - Call retek_force_commit().
- Delete from FUTURE_COST where active_date is less than VDATE - FUTURE_COST_HISTORY_DAYS and at least one record remains on the table for each item/supplier/origin country/location combination being deleted where the active_date is less than VDATE, and is the highest of all the records being deleted. (Delete from the table where a record in FUTURE_COST exists for the item/supplier/origin country/location combination with a higher active_date than the one being deleted, but with an active_date less than the VDATE. This can be achieved with an exists clause and a second instance of FUTURE_COST being fetched in this clause with the above described join.)
- Delete from RECLASS_COST_CHG_QUEUE where start_date is less than the VDATE.
- Call free_deals_array().
- Return.

Final(): Standard Retek final function. Free deals array. Call retek_close().

Size_deals_array(): Allocate memory for the driving cursor fetch array.

Free_deals_array(): Free memory allocated for the driving cursor fetch array.

Scheduling Considerations

This program should run after all other deals batch programs.

Processing Cycle: Ad Hoc

Scheduling Diagram:

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Deal upload [dealupld]

Design Overview

This program will read a Retek flat file, which was created by an external program and will upload information from this file to the following Deals Management tables in RMS: DEAL_HEAD, DEAL_DETAIL, DEAL_THRESHOLD, DEAL_ITEMLOC, POP_TERMS_DEF.

The external program will translate the standard EDI upload file into the RMS format specified below and will also translate the content of the EDI input file into RMS values (i.e.: a location's DUNS number to its RMS number) using the RMS database.

Dealupld.pc will take the above output file and use it as an input file to upload data into the Deals Management RMS database. The LUW is a single Deal Head Detail record and its associated component records in the input file. Therefore in each loop of the program one deal is uploaded at a time.

The program will verify that check constraints are not violated before the inserts actually take place. If a validation fails, a warning is written to the batch error file and the program resumes processing. Once the deal is processed, a check is made whether any warnings occurred. If yes, no insert occurs to the database but the entire deal is written to the reject file. This allows the user to see not only the first validation failure, but all of them before the deal was written out to the reject file.

See Section XIII Design Assumptions for more information on how records in the input file should be formatted and looped.

Stored Procedures / Shared Modules (Maintainability)

DEAL_SEQUENCE – This sequence is used to get the next deal_id for the deal being uploaded.

Input Specifications

'File-To-Table'

All fields that are of type Char in the input file format description above should be left justified and padded with the space character. If the field is not required and no value is being uploaded, the entire field should be padded with the space character.

Date fields are always formatted 'YYYYMMDDHH24MISS' (Char(14)).

All fields that are of type Number in the input file format description above should be right justified and padded with 0s. If the field is not required and no value is being uploaded, the entire field should be padded with 0s. All Number fields which have an implied decimal (e.g.: Number(10,4), but not Number(8)) are in effect the length of the number PLUS its decimal places. (E.g.: 14 byte wide field to hold a Number(10,4).) The program will upload the value formatted to include the 4 decimal digits (e.g.: Number(10,4): 01234567891234 -> 123456789.1234, Number(6,2): 12345600 -> 123456.00). Therefore if a number of size Number(20,4) is uploaded, this number will take up 24 bytes in the input file.

The input file must have the following structure:

```

FHEAD
{
    THEAD      of DHDTL      REQUIRED    for deal head record
    TDETL      REQUIRED      1 deal head record
    TTAIL      REQUIRED      end of deal head record
    THEAD      of DCDTL      REQUIRED    for deal component records
    [
        TDETL      OPTIONAL    for deal component records
    ]
    TTAIL      REQUIRED      end of deal component records
    THEAD      of DIDTL      REQUIRED    for item-loc records
    [
        TDETL      OPTIONAL    for item-loc records
    ]
    TTAIL      REQUIRED      end of item-loc records
    THEAD of PPDTL      REQUIRED    for proof of performance records
    [
        TDETL      OPTIONAL    for proof of performance records
    ]
    TTAIL      REQUIRED      end of proof of performance records
    THEAD      of DTDTL      REQUIRED    for threshold records
    [
        TDETL      OPTIONAL    for threshold records
    ]
    TTAIL      REQUIRED      end of threshold records
}
FTAIL

```

The set between the curly brackets may be looped to upload multiple deals from the same file. Within each set, the TDETL records in angle brackets may be sub-looped as a sub-set of the main set.

See Section XIII Design Assumptions for more information on how records in the input file should be formatted and looped.

Input File

Record Name	Field Name	Field Type	Default Value	Description/Constraints
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type (the beginning of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Type Definition	Char(5)	EDIDU	Identifies file as 'EDI Deals Upload'
	File Create Date	Char(14)	Create date	current date, formatted to 'YYYYMMDDHH24MISS'.
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal header.
	File Line Identifier	Numeric ID(10)	Sequential number	ID of current line being read from input file.

			Created by program.	
	Transaction Detail Record Type	Char(5)	DHDTL	Identifies file record type Deal Header. This record MUST BE FOLLOWED BY ONE AND ONLY ONE REQUIRED TDETL RECORD that holds the deal head information.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload a new deal.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Partner Type	Char(6)	REQUIRED	Type of the partner the deal applies to. Valid values are 'S' for a supplier, 'S1' for supplier hierarchy level 1 (e.g. manufacturer), 'S2' for supplier hierarchy level 2 (e.g. distributor) and 'S3' for supplier hierarchy level 3 (e.g. wholesaler). Descriptions of these codes will be held on the codes table under a code_type of 'SUHL'. Information pertaining to a single deal has to belong to the same supplier, since a deal may have only one supplier hierarchy associated with it. Only items with the same supplier hierarchy can be on the same deal. Supplier hierarchy is stored at an item / supplier / country / location level.
	Partner Id	Char(10)	Blank (space character string)	Level of supplier hierarchy (e.g. manufacturer, distributor or wholesaler), set up as a partner in the PARTNER table, used for assigning rebates by a level other than supplier. Rebates at this level will include all eligible supplier/item/country records assigned to this supplier hierarchy level. This field is required if the Partner Type field was set to 'S1', 'S2' or 'S3'. This field must be blank if the Partner Type field was set to 'S'.
	Supplier	Number(10)	Blank (space character string)	Deal supplier's number. This supplier can be at any level of supplier hierarchy. This field is required if the Partner Type field was set to 'S'. This field must be blank if the Partner Type field was set to 'S1', 'S2' or 'S3'.
	Type	Char(6)	REQUIRED	Type of the deal. Valid values are A for annual deal, P for promotional deal, O for PO-specific deal or M for vendor-funded markdown. Deal types will be held on the codes table under a code type of 'DLHT'.
	Currency Code	Char(3)	Blank (space character string)	Currency code of the deal's currency. All costs on the deal will be held in this currency. If Type is 'O', 'P' or 'A', then Currency Code may not be blank. Currency Code has to be blank if Type is 'M'.
	Active Date	Char(14)	REQUIRED	Date the deal will become active. This date will determine when deal components begin to be factored into item costs. For a PO-specific deal, the active_date will be the order's written date.

	Close Date	Char(14)	Blank (space character string)	Date the deal will/did end. This date determines when deal components are no longer factored into item costs. It is optional for annual deals, required for promotional deals. It will be left NULL for PO-specific deals. Close Date must not be blank if Type is 'P' or 'M'. Close Date has to be blank if Type is 'O'.
	External Reference Number	Char(30)	Blank (space character string)	Any given external reference number that is associated with the deal.
	Order Number	Number(8)	Blank (space character string)	Order the deal applies to, if the deal is PO-specific.
	Recalculate Approved Orders	Char(1)	REQUIRED	Indicates if approved orders should be recalculated based on this deal once the deal is approved. Valid values are Y for yes or N for no. Valid values are 'Y' and 'N'.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail). For DHDTL TDETL records this will always be 1!
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DCDTL	Identifies file record type of sub loop as Deal Component Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal components.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Component Type	Char(6)	REQUIRED	Type of the deal component, user-defined and stored on the DEAL_COMP_TYPE table.
	Application Order	Number(10)	Blank (space character string)	Number indicating the order in which the deal component should be applied with respect to any other deal components applicable to the item within the deal. This number will be unique across all deal components within the deal. It must be NULL for an M-type deal (vendor funded markdown).
	Billing Type	Char(6)	REQUIRED	Billing type of the deal component. Valid values are 'OI' for off-invoice, 'BD' for bill-back with debit memo or 'BC' for bill-back with credit note request. Billing types will be held on the codes table under a code type of 'DLBT'.

	Bill Back Period	Char(6)	Blank (space character string)	Code that identifies the bill-back period for the deal component. This field will only be populated for billing types of 'BD' or 'BC'. Bill back period codes will be user-defined and stored on the BILL_BACK_PERIOD table. If Billing Type is 'BD' or 'BC' then Bill Back Period must not be blank, otherwise it has to be blank.
	Collect Start Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should begin. If Billing Type is 'BD' or 'BC' then Collect Start Date must not be blank, otherwise it has to be blank.
	Collect End Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should end. If Billing Type is 'BD' or 'BC' then Collect End Date must not be blank, otherwise it has to be blank.
	Deal Application Timing	Char(6)	Blank (space character string)	Indicates when the deal component should be applied - at PO approval or time of receiving. Valid values are 'O' for PO approval, 'R' for receiving. These values will be held on the codes tables under a code type of 'AALC'. It must be NULL for an M-type deal (vendor funded markdown).
	Cost Application Level Indicator	Char(6)	Blank (space character string)	Indicates what cost bucket the deal component should affect. Valid values are 'N' for net cost, 'NN' for net net cost and 'DNN' for dead net net cost. These values will be held on the codes tables under a code type of 'DLCA'. It must be NULL for an M-type deal (vendor funded markdown).
	Pricing Cost Indicator	Char(1)	REQUIRED	Identifies deal components that should be included when calculating a pricing cost. Valid values are 'Y'es and 'N'o.
	Deal Class	Char(6)	Blank (space character string)	Identifies the calculation class of the deal component. Valid values are 'CU' for cumulative (discounts are added together and taken off as one lump sum), 'CS' for cascade (discounts are taken one at a time with subsequent discounts taken off the result of the previous discount) and 'EX' for exclusive (overrides all other discounts). 'EX' type deal components are only valid for promotional deals. Deal classes will be held on the codes table under a code type of 'DLCL'. It must be NULL for an M-type deal (vendor funded markdown).
	Threshold Limit Type	Char(6)	Blank (space character string)	Identifies whether thresholds will be set up as qty values, currency amount values or percentages (growth rebates only). Valid values are 'Q' for qty, 'A' for currency amount or 'P' for percentage. Threshold limit types will be held on the codes table under a code type of 'DLLT'. It must be NULL for an M-type deal (vendor funded markdown) or if the threshold value type is 'Q' (buy/get deals). If Growth Rebate Indicator is 'Y', then the Threshold Limit Type has to be 'P', otherwise 'Q', 'A' or NULL.

	Threshold Limit Unit of Measure	Char(4)	Blank (space character string)	Unit of measure of the threshold limits, if the limit type is quantity. Only Unit of Measures with a UOM class of 'VOL' (volume), 'MASS' or 'QTY' (quantity) can be used in this field. Valid Unit of Measures can be found on the UOM_CLASS table. If the Threshold Limit Type is 'A' or 'P', then Threshold Limit Unit of Measure has to be blank. If the Threshold Limit Type is 'Q', Threshold Limit Unit of Measure must not be blank. If Threshold Limit Type is blank, Threshold Limit Unit of Measure must be blank.
	Threshold Value Type	Char(6)	Blank (space character string)	Identifies whether the discount values associated with the thresholds will be set up as qty values, currency amount values, percentages or fixed amounts. Valid values are 'Q' for qty, 'A' for currency amount, 'P' for percentage or 'F' for fixed amount. Qty threshold value (buy/get) deals are only allowed on off-invoice discounts. Deal threshold value types will be held on the codes table under a code type of 'DLL2'. It must be NULL for an M-type deal (vendor funded markdown). If Billing Type is 'BD' or 'BC', then the Threshold Value Type must not be 'Q'.
	Buy Item	Char(25)	Blank (space character string)	Identifies the item that must be purchased for a quantity threshold-type discount. This value is required for quantity threshold value type discounts. Otherwise it has to be blank.
	Get Type	Char(6)	Blank (space character string)	Identifies the type of the 'get' discount for a quantity threshold-type (buy/get) discount. Valid values include 'X' (free), 'P' (percent), 'A' (amount) and 'F' (fixed amount). They are held on the codes table under a code type of 'DQGT'. This value is required for quantity threshold value deals. Otherwise it has to be blank.
	Get Value	Number(20,4)	All 0s.	Identifies the value of the 'get' discount for a quantity threshold-type (buy/get) discount that is not a 'free goods' deal. The Get Type above identifies the type of this value. This value is required for quantity threshold value type deals that are not a Get Type of free. Otherwise it has to be 0. If Get Type is 'P', 'A' or 'F', then Get Value must not be blank. If the Get Type is 'X' or blank, then Get Value has to be blank.
	Buy Item Quantity	Number(12,4)	All 0s.	Identifies the quantity of the threshold 'buy' item that must be ordered to qualify for the 'free' item. This value is required for quantity threshold value type discounts. Otherwise it has to be 0.
	Recursive Indicator	Char(1)	REQUIRED	For 'buy/get free' discounts, indicates if the quantity threshold discount is only for the first 'buy amt.' purchased (e.g. for the first 10 purchased, get 1 free), or if a free item will be given for every multiple of the 'buy amt' purchased on the order (e.g. for each 10 purchased, get 1 free). Valid values are 'Y' for yes or 'N' for no. If the Get Type is blank, then Recursive

				Indicator has to be 'N'.
	Buy Item Order Target Quantity	Number(12,4)	All 0s.	Indicates the targeted purchase level for all locations on a purchase order. This is the target level that will be used for future calculation of net cost. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Average Buy Item Order Target Quantity Per Location	Number(12,4)	All 0s.	Indicates the average targeted purchase level per location on the deal. This value will be used in future cost calculations. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Get Item	Char(25)	Blank (space character string)	Identifies the 'get' item for a quantity threshold-type (buy/get) discount. This value is required for quantity threshold value deals. Otherwise it has to be blank. If Get Type is 'P', 'A', 'F' or 'X', then Get Item must not be blank. If the Get Type is blank, then Get Item has to be blank.
	Get Quantity	Number(12,4)	All 0s.	Identifies the quantity of the identified 'get' item that will be given at the specified 'get' discount if the 'buy amt' of the buy item is purchased. This value is required for quantity threshold value type discounts. Otherwise it has to be 0. If Get Type is 'P', 'A', 'F' or 'X', then Get Quantity must not be 0. If the Get Type is blank, then Get Quantity has to be 0.
	Free Item Unit Cost	Number(20,4)	All 0s.	For 'buy/get free' discounts, identifies the unit cost of the threshold 'free' item that will be used in calculating the prorated qty. discount. It will default to the item/supplier cost, but can be modified based on the agreement with the supplier. It must be greater than zero as this is the cost that would normally be charged for the goods if no deal applied. If Get Type is 'P', 'A', 'F' or blank, then Free Item Unit Cost must be 0. If the Get Type is 'X', then Free Item Unit Cost must not be 0.
	Transaction Level Discount Indicator	Char(1)	REQUIRED	Indicates if the discount is a transaction-level discount (e.g. 10% across an entire PO). Valid Values are 'Y' or 'N'. If set to 'Y', Deal Class has to be 'CU' and Billing Type has to be 'OI'. No DIDL or PPDTL records may be present for a Transaction Level Discount DCDTL record.
	Rebate Indicator	Char(1)	REQUIRED	Indicates if the deal component is a rebate. Deal components can only be rebates for bill-back billing types. Valid values are 'Y' for yes or 'N' for no. If Billing Type is 'OI', then Rebate Indicator must be 'N'.
	Rebate Active Date	Char(14)	Blank (space character string)	If the rebate becomes active on a different date than the deal active date, this field will hold that date. If this field is NULL for a rebate line, it will be assumed that the rebate becomes active on the deal active date.
	Rebate Calculation Type	Char(6)	Blank (space character string)	Indicates if the rebate should be calculated using linear or scalar calculation methods. Valid values are 'L' for linear or 'S' for scalar. This field will be required if the rebate

				indicator is 'Y'. Rebate calculation types will be held on the codes table under a code type of 'DLCT'. If Rebate Indicator is 'Y', then Rebate Calculation Type must not be blank. Otherwise it has to be blank.
	Growth Rebate Indicator	Char(1)	REQUIRED	Indicates if the rebate is a growth rebate, meaning it is calculated and applied based on an increase in purchases or sales over a specified period of time. Valid values are 'Y' for yes or 'N' for no. If Rebate Indicator is 'N', then Growth Rebate Indicator must be 'N'.
	Historical Comparison Start Date	Char(14)	Blank (space character string)	The first date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison Start Date must not be blank. Otherwise it must be blank.
	Historical Comparison End Date	Char(14)	Blank (space character string)	The last date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison End Date must not be blank. Otherwise it must be blank.
	Current Comparison Start Date	Char(14)	Blank (space character string)	The first date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison Start Date must not be blank. Otherwise it must be blank.
	Current Comparison End Date	Char(14)	Blank (space character string)	The last date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison End Date must not be blank. Otherwise it must be blank.
	Rebate Purchases or Sales Application Indicator	Char(6)	Blank (space character string)	Indicates if the rebate should be applied to purchases or sales. Valid values are 'P' for purchases or 'S' for sales. It will be required if the rebate indicator is 'Y'. Rebate purchase/sales indicators will be held on the codes table under a code type of 'DLRP'. If the Rebate Indicator is 'Y', then the Rebate Purchases or Sales Application Indicator must not be blank. Otherwise it has to be blank.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal component.
Transaction	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the

Trailer				transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DIDTL	Identifies file record type of sub loop as Deal Component Item-location Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal item-location details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Merchandise Level	Char(6)	REQUIRED	Indicates what level of the merchandise hierarchy the record is at. Valid values include '1' for company-wide (all items), '2' for division, '3' for group, '4' for dept, '5' for class, '6' for subclass, '7' for line, '8' for line/differentiator 1, '9' for line/differentiator 2 and '10' for item. These level types will be held on the codes table under a code type of 'DIML'.
	Company Indicator	Char(1)	REQUIRED	Indicates if the deal component is applied company-wide (e.g. all items in the system will be included in the discount or rebate). Valid values are 'Y' for yes and 'N' for no.
	Division	Number(4)	Blank (space character string).	ID of the division included in or excluded from the deal component. Valid values are on the DIVISION table. If Group is not blank, then Division must not be blank. If Merchandise Level is 2, then Division must not be blank and Group, Department, Class and Subclass must be blank.
	Group	Number(4)	Blank (space character string).	ID of the group included in or excluded from the deal component. Valid values are on the GROUPS table. If Department is not blank, then Group must not be blank. If Merchandise Level is 3, then Group must not be blank and Department, Class and Subclass must be blank.
	Department	Number(4)	Blank (space character string).	ID of the department included in or excluded from the deal component. Valid values are on the DEPS table. If Class is not blank, then Department must not be blank. If Merchandise Level is 4, then Department must not be blank and Class and Subclass must be blank.
	Class	Number(4)	Blank (space	ID of the class included in or excluded from

			character string).	the deal component. Valid values are on the CLASS table. If Subclass is not blank, then Class must not be blank. If Merchandise Level is 5, then Class must not be blank and Subclass must be blank.
	Subclass	Number(4)	Blank (space character string).	ID of the subclass included in or excluded from the deal component. Valid values are on the SUBCLASS table. If Merchandise Level is 6 or more than 6, then Subclass must not be blank.
	Item Parent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group at the level above the item. This value must exist as an item in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).
	Item Grandparent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group two levels above the item. This value must exist as both an item and an item parent in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).
	Differentiator 1	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent, Item Parent and Differentiator 2 are blank, then Differentiator 1 must be blank. If Merchandise Level is 8, then Differentiator 1 must not be blank.
	Differentiator 2	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent, Item Parent and Differentiator 1 are blank, then Differentiator 2 must be blank. If Merchandise Level is 9, then Differentiator 2 must not be blank.
	Organizational Level	Char(6)	Blank (space character string)	Indicates what level of the organizational hierarchy the record is at. Valid values include '1' for chain, '2' for area, '3' for region, '4' for district and '5' for location. These level types will be held on the codes table under a code type of 'DIOL'. If company indicator is N, this must not be blank. If location type is warehouse or location list, this must be 5.
	Chain	Number(4)	Blank (space character string).	ID of the chain included in or excluded from the deal component. Valid values are on the CHAIN table. If org. level is 1, this field must not be blank.
	Area	Number(4)	Blank (space character string).	ID of the area included in or excluded from the deal component. Valid values are on the AREA table. If org. level is 2, this field and chain must not be blank.

	Region	Number(4)	Blank (space character string).	ID of the region included in or excluded from the deal component. Valid values are on the REGION table. If org. level is 3, this field, area, and chain must not be blank.
	District	Number(4)	Blank (space character string).	ID of the district included in or excluded from the deal component. Valid values are on the DISTRICT table. If org. level is 4, then this field, region, area, and chain must not be blank.
	Location	Number(10)	Blank (space character string).	ID of the location included in or excluded from the deal component. Valid values are on the STORE, WH, or LOC_LIST_HEAD table. If org. level is 5, this field must not be blank. Chain, area, region, and district should be blank if the loc_type is L or W. If the loc_type is S, then they all must not be blank. If Location Type is not blank, then Location must not be blank. Otherwise it has to be blank.
	Origin Country Identifier	Char(3)	Blank (space character string)	Origin country of the item that the deal component should apply to.
	Location Type	Char(1)	Blank (space character string)	Type of the location referenced in the location field. Valid values are 'S' and 'W'. Location types will be held on the codes table under the code type 'LOC3'. If location is blank then this field has to be blank also.
	Item	Char(25)	Blank (space character string)	Unique alphanumeric value that identifies the item. If Merchandise Level is 10, then Item must not be blank.
	Exclusion Indicator	Char(1)	REQUIRED	Indicates if the deal component item/location line is included in the deal component or excluded from it. Valid values are 'Y' for yes or 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this item-loc record belongs to. See the section XII Design Assumptions for more explanation on how this field should be populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by	ID of current line being read from input file.

			program.	
	Transaction Detail Record Type	Char(5)	PPDTL	Identifies file record type of sub loop as Proof of Performance Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal proof of performance details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Sub Item	Char(25)		Specific transaction level (or below) item that's proof of performance is being measured. This can be populated when the deal itself is on a case UPC but the proof of performance is on an individual selling unit.
	Proof of Performance Type	Char(6)	REQUIRED	Code that identifies the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_type is code 'E' for end cap display). Valid values for this field are stored in the code_type = 'PPT'. This field is required by the database.
	Proof of Performance Value	Number(20,4)	All 0s.	Value that describes the term of the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value is 28). This field is required by the database if the record has a pop_value_type. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Proof of Performance Value Type	Char(6)	Blank (space character string)	Value that describes the type of the pop_value (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value_type is the code 'D' for days). Valid values for this field are stored in the code_type = 'PPVT'. This field is required by the database if the record has a pop_value. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Vendor Recommended Start Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP begin.
	Vendor Recommended End Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP end.
	Planned Start Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to begin the POP.
	Planned End Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to end the POP.
	Comment	Char(255)	Blank (space character string)	Free-form comments.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Proof of Performance record belongs to. See the Assumptions section for more explanation on how this field should be

				populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DTDTL	Identifies file record type of sub loop as Deal Component Threshold Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal threshold details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Lower Limit	Number(20,4)	REQUIRED	Lower limit of the deal component. This is the minimum value that must be met in order to get the specified discount. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Upper Limit	Number(20,4)	REQUIRED	Upper limit of the deal component. This is the maximum value for which the specified discount will apply. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Value	Number(20,4)	REQUIRED	Value of the discount that will be given for meeting the specified thresholds for this deal component. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_value_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Target Level Indicator	Char(1)	REQUIRED	Indicates if a threshold level is the targeted purchase or sales level for a deal component. This indicator will be used for cost calculations. Valid values are 'Y' for yes and 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Threshold record belongs to. See the

				Assumptions section for more explanation on how this field should be populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
File Trailer	File Line Identifier	Char(5)	FTAIL	Identifies file record type (the end of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Record Counter	Numeric ID(10)	Sequential number Created by program.	Number of records/transactions in current file (only records between head & tail)

Output Specifications

'Table-To-Table'

Reject File

Record Name	Field Name	Field Type	Default Value	Description/Constraints
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type (the beginning of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Type Definition	Char(5)	EDIDU	Identifies file as 'EDI Deals Upload'
	File Create Date	Char(14)	Create date	current date, formatted to 'YYYYMMDDHH24MISS'.
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal header.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DHDTL	Identifies file record type Deal Header. This record MUST BE FOLLOWED BY ONE AND ONLY ONE REQUIRED TDETL RECORD that holds the deal head information.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload a new deal.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Partner Type	Char(6)	REQUIRED	Type of the partner the deal applies to. Valid

				values are 'S' for a supplier, 'S1' for supplier hierarchy level 1 (e.g. manufacturer), 'S2' for supplier hierarchy level 2 (e.g. distributor) and 'S3' for supplier hierarchy level 3 (e.g. wholesaler). Descriptions of these codes will be held on the codes table under a code_type of 'SUHL'. Information pertaining to a single deal has to belong to the same supplier, since a deal may have only one supplier hierarchy associated with it. Only items with the same supplier hierarchy can be on the same deal. Supplier hierarchy is stored at an item / supplier / country / location level.
	Partner Id	Char(10)	Blank (space character string)	Level of supplier hierarchy (e.g. manufacturer, distributor or wholesaler), set up as a partner in the PARTNER table, used for assigning rebates by a level other than supplier. Rebates at this level will include all eligible supplier/item/country records assigned to this supplier hierarchy level. This field is required if the Partner Type field was set to 'S1', 'S2' or 'S3'. This field must be blank if the Partner Type field was set to 'S'.
	Supplier	Number(10)	Blank (space character string)	Deal supplier's number. This supplier can be at any level of supplier hierarchy. This field is required if the Partner Type field was set to 'S'. This field must be blank if the Partner Type field was set to 'S1', 'S2' or 'S3'.
	Type	Char(6)	REQUIRED	Type of the deal. Valid values are A for annual deal, P for promotional deal, O for PO-specific deal or M for vendor-funded markdown. Deal types will be held on the codes table under a code type of 'DLHT'.
	Currency Code	Char(3)	Blank (space character string)	Currency code of the deal's currency. All costs on the deal will be held in this currency. If Type is 'O', 'P' or 'A', then Currency Code may not be blank. Currency Code has to be blank if Type is 'M'.
	Active Date	Char(14)	REQUIRED	Date the deal will become active. This date will determine when deal components begin to be factored into item costs. For a PO-specific deal, the active_date will be the order's written date.
	Close Date	Char(14)	Blank (space character string)	Date the deal will/did end. This date determines when deal components are no longer factored into item costs. It is optional for annual deals, required for promotional deals. It will be left NULL for PO-specific deals. Close Date must not be blank if Type is 'P' or 'M'. Close Date has to be blank if Type is 'O'.
	External Reference Number	Char(30)	Blank (space character string)	Any given external reference number that is associated with the deal.
	Order Number	Number(8)	Blank (space character string)	Order the deal applies to, if the deal is PO-specific.
	Recalculate Approved Orders	Char(1)	REQUIRED	Indicates if approved orders should be recalculated based on this deal once the deal is approved. Valid values are Y for yes or N for no.

				no. Valid values are 'Y' and 'N'.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail). For DHDTL TDETL records this will always be 1!
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DCDTL	Identifies file record type of sub loop as Deal Component Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal components.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Component Type	Char(6)	REQUIRED	Type of the deal component, user-defined and stored on the DEAL_COMP_TYPE table.
	Application Order	Number(10)	Blank (space character string)	Number indicating the order in which the deal component should be applied with respect to any other deal components applicable to the item within the deal. This number will be unique across all deal components within the deal. It must be NULL for an M-type deal (vendor funded markdown).
	Billing Type	Char(6)	REQUIRED	Billing type of the deal component. Valid values are 'OI' for off-invoice, 'BD' for bill-back with debit memo or 'BC' for bill-back with credit note request. Billing types will be held on the codes table under a code type of 'DLBT'.
	Bill Back Period	Char(6)	Blank (space character string)	Code that identifies the bill-back period for the deal component. This field will only be populated for billing types of 'BD' or 'BC'. Bill back period codes will be user-defined and stored on the BILL_BACK_PERIOD table. If Billing Type is 'BD' or 'BC' then Bill Back Period must not be blank, otherwise it has to be blank.
	Collect Start Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should begin. If Billing Type is 'BD' or 'BC' then Collect Start Date must not be blank, otherwise it has to be blank.

	Collect End Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should end. If Billing Type is 'BD' or 'BC' then Collect End Date must not be blank, otherwise it has to be blank.
	Deal Application Timing	Char(6)	Blank (space character string)	Indicates when the deal component should be applied - at PO approval or time of receiving. Valid values are 'O' for PO approval, 'R' for receiving. These values will be held on the codes tables under a code type of 'AALC'. It must be NULL for an M-type deal (vendor funded markdown).
	Cost Application Level Indicator	Char(6)	Blank (space character string)	Indicates what cost bucket the deal component should affect. Valid values are 'N' for net cost, 'NN' for net net cost and 'DNN' for dead net net cost. These values will be held on the codes tables under a code type of 'DLCA'. It must be NULL for an M-type deal (vendor funded markdown).
	Pricing Cost Indicator	Char(1)	REQUIRED	Identifies deal components that should be included when calculating a pricing cost. Valid values are 'Y'es and 'N'o.
	Deal Class	Char(6)	Blank (space character string)	Identifies the calculation class of the deal component. Valid values are 'CU' for cumulative (discounts are added together and taken off as one lump sum), 'CS' for cascade (discounts are taken one at a time with subsequent discounts taken off the result of the previous discount) and 'EX' for exclusive (overrides all other discounts). 'EX' type deal components are only valid for promotional deals. Deal classes will be held on the codes table under a code type of 'DLCL'. It must be NULL for an M-type deal (vendor funded markdown).
	Threshold Limit Type	Char(6)	Blank (space character string)	Identifies whether thresholds will be set up as qty values, currency amount values or percentages (growth rebates only). Valid values are 'Q' for qty, 'A' for currency amount or 'P' for percentage. Threshold limit types will be held on the codes table under a code type of 'DLLT'. It must be NULL for an M-type deal (vendor funded markdown) or if the threshold value type is 'Q' (buy/get deals). If Growth Rebate Indicator is 'Y', then the Threshold Limit Type has to be 'P', otherwise 'Q', 'A' or NULL.
	Threshold Limit Unit of Measure	Char(4)	Blank (space character string)	Unit of measure of the threshold limits, if the limit type is quantity. Only Unit of Measures with a UOM class of 'VOL' (volume), 'MASS' or 'QTY' (quantity) can be used in this field. Valid Unit of Measures can be found on the UOM_CLASS table. If the Threshold Limit Type is 'A' or 'P', then Threshold Limit Unit of Measure has to be blank. If the Threshold Limit Type is 'Q', Threshold Limit Unit of Measure must not be blank. If Threshold Limit Type is blank, Threshold Limit Unit of Measure must be blank.
	Threshold Value Type	Char(6)	Blank (space character string)	Identifies whether the discount values associated with the thresholds will be set up as qty values, currency amount values,

				percentages or fixed amounts. Valid values are 'Q' for qty, 'A' for currency amount, 'P' for percentage or 'F' for fixed amount. Qty threshold value (buy/get) deals are only allowed on off-invoice discounts. Deal threshold value types will be held on the codes table under a code type of 'DLL2'. It must be NULL for an M-type deal (vendor funded markdown). If Billing Type is 'BD' or 'BC', then the Threshold Value Type must not be 'Q'.
	Buy Item	Char(25)	Blank (space character string)	Identifies the item that must be purchased for a quantity threshold-type discount. This value is required for quantity threshold value type discounts. Otherwise it has to be blank.
	Get Type	Char(6)	Blank (space character string)	Identifies the type of the 'get' discount for a quantity threshold-type (buy/get) discount. Valid values include 'X' (free), 'P' (percent), 'A' (amount) and 'F' (fixed amount). They are held on the codes table under a code type of 'DQGT'. This value is required for quantity threshold value deals. Otherwise it has to be blank.
	Get Value	Number(20,4)	All 0s.	Identifies the value of the 'get' discount for a quantity threshold-type (buy/get) discount that is not a 'free goods' deal. The Get Type above identifies the type of this value. This value is required for quantity threshold value type deals that are not a Get Type of free. Otherwise it has to be 0. If Get Type is 'P', 'A' or 'F', then Get Value must not be blank. If the Get Type is 'X' or blank, then Get Value has to be blank.
	Buy Item Quantity	Number(12,4)	All 0s.	Identifies the quantity of the threshold 'buy' item that must be ordered to qualify for the 'free' item. This value is required for quantity threshold value type discounts. Otherwise it has to be 0.
	Recursive Indicator	Char(1)	REQUIRED	For 'buy/get free' discounts, indicates if the quantity threshold discount is only for the first 'buy amt.' purchased (e.g. for the first 10 purchased, get 1 free), or if a free item will be given for every multiple of the 'buy amt' purchased on the order (e.g. for each 10 purchased, get 1 free). Valid values are 'Y' for yes or 'N' for no. If the Get Type is blank, then Recursive Indicator has to be 'N'.
	Buy Item Order Target Quantity	Number(12,4)	All 0s.	Indicates the targeted purchase level for all locations on a purchase order. This is the target level that will be used for future calculation of net cost. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Average Buy Item Order Target Quantity Per Location	Number(12,4)	All 0s.	Indicates the average targeted purchase level per location on the deal. This value will be used in future cost calculations. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Get Item	Char(25)	Blank (space character string)	Identifies the 'get' item for a quantity threshold-type (buy/get) discount. This value is required for quantity threshold value deals. Otherwise it has to be blank.

				If Get Type is 'P', 'A', 'F' or 'X', then Get Item must not be blank. If the Get Type is blank, then Get Item has to be blank.
	Get Quantity	Number(12,4)	All 0s.	Identifies the quantity of the identified 'get' item that will be given at the specified 'get' discount if the 'buy amt' of the buy item is purchased. This value is required for quantity threshold value type discounts. Otherwise it has to be 0. If Get Type is 'P', 'A', 'F' or 'X', then Get Quantity must not be 0. If the Get Type is blank, then Get Quantity has to be 0.
	Free Item Unit Cost	Number(20,4)	All 0s.	For 'buy/get free' discounts, identifies the unit cost of the threshold 'free' item that will be used in calculating the prorated qty. discount. It will default to the item/supplier cost, but can be modified based on the agreement with the supplier. It must be greater than zero as this is the cost that would normally be charged for the goods if no deal applied. If Get Type is 'P', 'A', 'F' or blank, then Free Item Unit Cost must be 0. If the Get Type is 'X', then Free Item Unit Cost must not be 0.
	Transaction Level Discount Indicator	Char(1)	REQUIRED	Indicates if the discount is a transaction-level discount (e.g. 10% across an entire PO). Valid Values are 'Y' or 'N'. If set to 'Y', Deal Class has to be 'CU' and Billing Type has to be 'OI'. No DIDTL or PPDTL records may be present for a Transaction Level Discount DCDTL record.
	Rebate Indicator	Char(1)	REQUIRED	Indicates if the deal component is a rebate. Deal components can only be rebates for bill-back billing types. Valid values are 'Y' for yes or 'N' for no. If Billing Type is 'OI', then Rebate Indicator must be 'N'.
	Rebate Active Date	Char(14)	Blank (space character string)	If the rebate becomes active on a different date than the deal active date, this field will hold that date. If this field is NULL for a rebate line, it will be assumed that the rebate becomes active on the deal active date.
	Rebate Calculation Type	Char(6)	Blank (space character string)	Indicates if the rebate should be calculated using linear or scalar calculation methods. Valid values are 'L' for linear or 'S' for scalar. This field will be required if the rebate indicator is 'Y'. Rebate calculation types will be held on the codes table under a code type of 'DLCT'. If Rebate Indicator is 'Y', then Rebate Calculation Type must not be blank. Otherwise it has to be blank.
	Growth Rebate Indicator	Char(1)	REQUIRED	Indicates if the rebate is a growth rebate, meaning it is calculated and applied based on an increase in purchases or sales over a specified period of time. Valid values are 'Y' for yes or 'N' for no. If Rebate Indicator is 'N', then Growth Rebate Indicator must be 'N'.
	Historical Comparison Start	Char(14)	Blank (space character string)	The first date of the historical period against which growth will be measured in this growth

	Date			rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison Start Date must not be blank. Otherwise it must be blank.
	Historical Comparison End Date	Char(14)	Blank (space character string)	The last date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison End Date must not be blank. Otherwise it must be blank.
	Current Comparison Start Date	Char(14)	Blank (space character string)	The first date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison Start Date must not be blank. Otherwise it must be blank.
	Current Comparison End Date	Char(14)	Blank (space character string)	The last date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison End Date must not be blank. Otherwise it must be blank.
	Rebate Purchases or Sales Application Indicator	Char(6)	Blank (space character string)	Indicates if the rebate should be applied to purchases or sales. Valid values are 'P' for purchases or 'S' for sales. It will be required if the rebate indicator is 'Y'. Rebate purchase/sales indicators will be held on the codes table under a code type of 'DLRP'. If the Rebate Indicator is 'Y', then the Rebate Purchases or Sales Application Indicator must not be blank. Otherwise it has to be blank.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal component.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.

	Transaction Detail Record Type	Char(5)	DIDTL	Identifies file record type of sub loop as Deal Component Item-location Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal item-location details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Merchandise Level	Char(6)	REQUIRED	Indicates what level of the merchandise hierarchy the record is at. Valid values include '1' for company-wide (all items), '2' for division, '3' for group, '4' for dept, '5' for class, '6' for subclass, '7' for line, '8' for line/differentiator 1, '9' for line/differentiator 2 and '10' for item. These level types will be held on the codes table under a code type of 'DIML'.
	Company Indicator	Char(1)	REQUIRED	Indicates if the deal component is applied company-wide (e.g. all items in the system will be included in the discount or rebate). Valid values are 'Y' for yes and 'N' for no.
	Division	Number(4)	Blank (space character string).	ID of the division included in or excluded from the deal component. Valid values are on the DIVISION table. If Group is not blank, then Division must not be blank. If Merchandise Level is 2, then Division must not be blank and Group, Department, Class and Subclass must be blank.
	Group	Number(4)	Blank (space character string).	ID of the group included in or excluded from the deal component. Valid values are on the GROUPS table. If Department is not blank, then Group must not be blank. If Merchandise Level is 3, then Group must not be blank and Department, Class and Subclass must be blank.
	Department	Number(4)	Blank (space character string).	ID of the department included in or excluded from the deal component. Valid values are on the DEPS table. If Class is not blank, then Department must not be blank. If Merchandise Level is 4, then Department must not be blank and Class and Subclass must be blank.
	Class	Number(4)	Blank (space character string).	ID of the class included in or excluded from the deal component. Valid values are on the CLASS table. If Subclass is not blank, then Class must not be blank. If Merchandise Level is 5, then Class must not be blank and Subclass must be blank.
	Subclass	Number(4)	Blank (space character string).	ID of the subclass included in or excluded from the deal component. Valid values are on the SUBCLASS table. If Merchandise Level is 6 or more than 6, then Subclass must not be blank.
	Item Parent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group at the level above the item. This value must exist as an item in another row on the item_master table.

				If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).
	Item Grandparent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group two levels above the item. This value must exist as both an item and an item parent in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).
	Differentiator 1	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent, Item Parent and Differentiator 2 are blank, then Differentiator 1 must be blank. If Merchandise Level is 8, then Differentiator 1 must not be blank.
	Differentiator 2	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent, Item Parent and Differentiator 1 are blank, then Differentiator 2 must be blank. If Merchandise Level is 9, then Differentiator 2 must not be blank.
	Organizational Level	Char(6)	Blank (space character string)	Indicates what level of the organizational hierarchy the record is at. Valid values include '1' for chain, '2' for area, '3' for region, '4' for district and '5' for location. These level types will be held on the codes table under a code type of 'DIOL'. If company indicator is N, this must not be blank. If location type is warehouse or location list, this must be 5.
	Chain	Number(4)	Blank (space character string).	ID of the chain included in or excluded from the deal component. Valid values are on the CHAIN table. If org. level is 1, this field must not be blank.
	Area	Number(4)	Blank (space character string).	ID of the area included in or excluded from the deal component. Valid values are on the AREA table. If org. level is 2, this field and chain must not be blank.
	Region	Number(4)	Blank (space character string).	ID of the region included in or excluded from the deal component. Valid values are on the REGION table. If org. level is 3, this field, area, and chain must not be blank.
	District	Number(4)	Blank (space character string).	ID of the district included in or excluded from the deal component. Valid values are on the DISTRICT table. If org. level is 4, then this field, region, area, and chain must not be blank.
	Location	Number(10)	Blank (space character string).	ID of the location included in or excluded from the deal component. Valid values are on the STORE, WH, or LOC_LIST_HEAD table. If org. level is 5, this field must not be blank.

				Chain, area, region, and district should be blank if the loc_type is L or W. If the loc_type is S, then they all must not be blank. If Location Type is not blank, then Location must not be blank. Otherwise it has to be blank.
	Origin Country Identifier	Char(3)	Blank (space character string)	Origin country of the item that the deal component should apply to.
	Location Type	Char(1)	Blank (space character string)	Type of the location referenced in the location field. Valid values are 'S' and 'W'. Location types will be held on the codes table under the code type 'LOC3'. If location is blank then this field has to be blank also.
	Item	Char(25)	Blank (space character string)	Unique alphanumeric value that identifies the item. If Merchandise Level is 10, then Item must not be blank.
	Exclusion Indicator	Char(1)	REQUIRED	Indicates if the deal component item/location line is included in the deal component or excluded from it. Valid values are 'Y' for yes or 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this item-loc record belongs to. See the section XII Design Assumptions for more explanation on how this field should be populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	PPDTL	Identifies file record type of sub loop as Proof of Performance Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal proof of performance details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Sub Item	Char(25)		Specific transaction level (or below) item that's proof of performance is being measured. This can be populated when the deal itself is on a case UPC but the proof of performance is on an individual selling unit.
	Proof of Performance	Char(6)	REQUIRED	Code that identifies the proof of performance

	Type			type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_type is code 'E' for end cap display). Valid values for this field are stored in the code_type = 'PPT'. This field is required by the database.
	Proof of Performance Value	Number(20,4)	All 0s.	Value that describes the term of the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value is 28). This field is required by the database if the record has a pop_value_type. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Proof of Performance Value Type	Char(6)	Blank (space character string)	Value that describes the type of the pop_value (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value_type is the code 'D' for days). Valid values for this field are stored in the code_type = 'PPVT'. This field is required by the database if the record has a pop_value. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Vendor Recommended Start Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP begin.
	Vendor Recommended End Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP end.
	Planned Start Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to begin the POP.
	Planned End Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to end the POP.
	Comment	Char(255)	Blank (space character string)	Free-form comments.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Proof of Performance record belongs to. See the Assumptions section for more explanation on how this field should be populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential	ID of current line being read from input file.

			number Created by program.	
	Transaction Detail Record Type	Char(5)	DTDTL	Identifies file record type of sub loop as Deal Component Threshold Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal threshold details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Lower Limit	Number(20,4)	REQUIRED	Lower limit of the deal component. This is the minimum value that must be met in order to get the specified discount. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Upper Limit	Number(20,4)	REQUIRED	Upper limit of the deal component. This is the maximum value for which the specified discount will apply. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Value	Number(20,4)	REQUIRED	Value of the discount that will be given for meeting the specified thresholds for this deal component. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_value_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Target Level Indicator	Char(1)	REQUIRED	Indicates if a threshold level is the targeted purchase or sales level for a deal component. This indicator will be used for cost calculations. Valid values are 'Y' for yes and 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Threshold record belongs to. See the Assumptions section for more explanation on how this field should be populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
File Trailer	File Line Identifier	Char(5)	FTAIL	Identifies file record type (the end of the input file).

	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Record Counter	Numeric ID(10)	Sequential number Created by program.	Number of records/transactions in current file (only records between head & tail)

Information from the input file is uploaded into the following Deals Management tables:

DEAL_HEAD

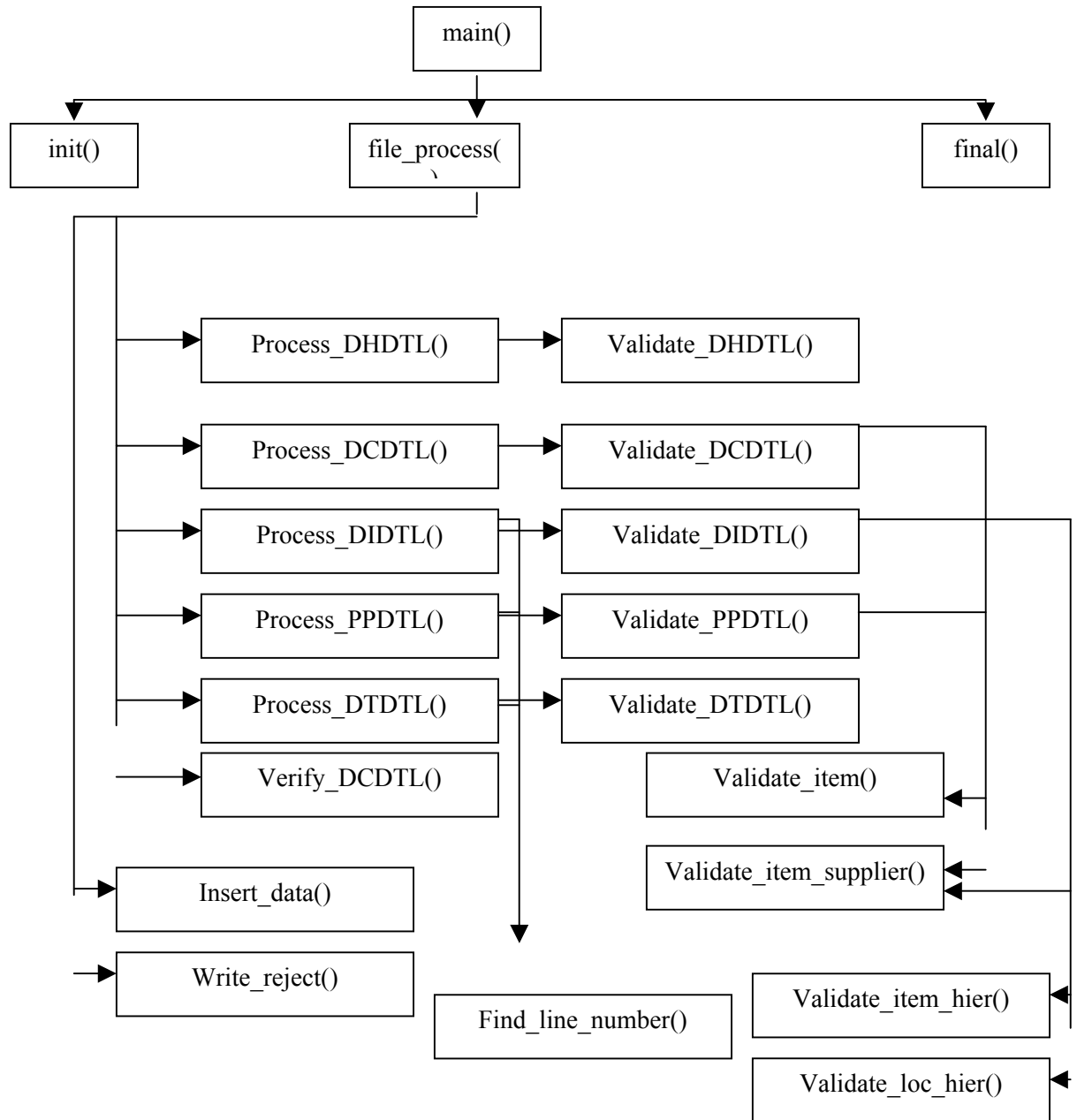
DEAL_DETAIL

DEAL_THRESHOLD

DEAL_ITEMLOC

POP_TERMS_DEF

Function Level Description



Declare reject file line number counter(s) and structs to hold data fetched from input file until one deal is complete and to process validation. Declare global variables as needed. The following single record structs are needed:

- A struct to hold the FHEAD record.

- A struct to hold the THEAD record.
- A struct to hold the DHDTL record + system fields for the insert into RMS (eg: deal_id, line-number).

Declare sizeable structs to hold the potentially multi-line sub loops of a DHDTL record from the input file:

- A struct to hold TDETL of DCDTL records + system fields for the insert into RMS (eg: deal_id, line-number).
- A struct to hold TDETL of DIDTL records + system fields for the insert into RMS (eg: deal_id, line-number).
- A struct to hold TDETL of DTDTL records + system fields for the insert into RMS (eg: deal_id).
- A struct to hold TDETL of PPDTL records + system fields for the insert into RMS (eg: deal_id).

Main(): Standard RETEK main function.

- Log on to DATABASE.
- Calls init(), process() and final().
- Log appropriate messages for batch run based on return from above calls.

Init(): Handles restart/recovery initialization, populates global system variables for batch run. Opens input file, and reject output file.

Size_structs(): This function will size the four sizeable structs:

- A struct to hold TDETL of DCDTL records.
- A struct to hold TDETL of DIDTL records.
- A struct to hold TDETL of DTDTL records.
- A struct to hold TDETL of PPDTL records.

Resize_DCDTL_array(): grows the struct to hold TDETL of DCDTL records.

Resize_DIDTL_array(): grows the struct to hold TDETL of DDTL records.

Resize_DTDTL_array(): grows the struct to hold TDETL of DTDTL records.

Resize_PPDTL_array(): grows the struct to hold TDETL of PPDTL records.

File_process(): This function will call the rest of the functions necessary to process the input file while it loops through the input file.

- Call size_structs().
- Loop-get records from input file:
 - Save current file position of this record into a local variable in case something in the deal gets rejected and we need to write the entire deal and its sub-records to the reject file.
 - When FTAIL record is reached, break out of loop.
 - Get THEAD record, make sure it signals a DHDTL record to follow.
 - Call process_DHDTL().
 - Get THEAD record, make sure it signals a DCDTL record to follow.
 - Call process_DCDTL().
 - Get THEAD record, make sure it signals a DIDTL record to follow.

- Call process_DIDTL().
- Get THEAD record, make sure it signals a PPDTL record to follow.
- Call process_PPDTL().
- Get THEAD record, make sure it signals a DTDTL record to follow.
- Call process_DTDTL().
- If sub function returned code to signal a failed validation, set a local variable to reflect a non-fatal error.
- Call insert_data() or write_reject() depending on whether any non-fatal errors have been recorded.
- Set restart variables and force a commit.
- Return.

Process_DHDTL(): This function manages the processing of a DHDTL record.

- Fetch new deal_id.
- Get next line from input file, verify that it holds a TDETL.
- Insert DHDTL record from generic file-read buffer to the DHDTL struct, perform any nullpad or zeropad as necessary. Also insert line number of record into struct.
- Validate the DHDTL record by calling validate_DHDTL().
- If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
- If validation of the DHDTL record passed, return code representing this status.

Validate_DHDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)
- If an order number is given, verify it exists in RMS.
- If a supplier is given, verify it exists in RMS.

Process_DCDTL(): This function manages the processing of DCDTL records.

- Create deal_detail_id.
- Loop process:
 - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
 - Insert DCDTL record from generic file-read buffer to the DCDTL struct, perform any nullpad or zeropad as necessary. Also insert line number of record, deal_id and deal_detail_id into struct.
 - Validate DCDTL record by calling validate_DCDTL().
 - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
 - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_DCDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.

- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)

Process_DIDTL(): This function manages the processing of DIDTL records.

- Create seq_no.
- Loop process:
 - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
 - Insert DIDTL record from generic file-read buffer to the DIDTL struct, perform any nullpad or zeropad as necessary. Also insert line number of record and seq_no into struct.
 - Validate DIDTL record by calling validate_DIDTL().
 - Call find_line_number() to get the index of the record and which struct the record is in that this DIDTL record belongs to. At this level, verify that the line number was found in the DCDTL struct and copy over the deal_id and deal_detail_id into this struct. If the referenced DCDTL record that this DIDTL record belongs to has a type 'M' (vendor-funded markdown), there must be no DIDTL records. If the DCDTL record this DIDTL record is associated with is a Transaction Level Discount, write an error message.
 - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
 - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_DIDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)

Process_PPDTL(): This function manages the processing of PPDTL records.

- Create pop_def_seq_no.
- Loop process:
 - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
 - Insert PPDTL record from generic file-read buffer to the PPDTL struct, perform any nullpad or zeropad as necessary. Also insert pop_def_seq_no into struct.
 - Validate PPDTL record by calling validate_PPDTL().
 - Call find_line_number() to get the index of the record and which struct the record is in that this PPDTL record belongs to. Copy over the deal_id, and deal_detail_id, and seq_no if available into this struct. If the referenced DCDTL record that this PPDTL record belongs to is a Transaction Level Deal, there must be no PPDTL records.
 - Validate deal_sub_item, which needs to be blank unless referenced record for this PPDTL record is a DIDTL record, in which case

deal_sub_item must be a child or component item of the item in the referenced DIDTL record.

- If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
- If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_PPDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)
- Make sure that if both Vendor Recommended Start Date and Vendor Recommended End Date are given, then the start date is before the end date.
- Make sure that if both Planned Start Date and Planned End Date are given, then the start date is before the end date.
- Make sure that the current deal is not of type M (Vendor Funded Markdown). Such deals should have no PPDTL records.

Process_DTDTL(): This function manages the processing of DTDTL records.

- Loop process:
 - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
 - Insert DTDTL record from generic file-read buffer to the DTDTL struct, perform any nullpad or zeropad as necessary.
 - Validate DTDTL record by calling validate_DTDTL().
 - Call find_line_number() to get the index of the record and which struct the record is in that this DTDTL record belongs to. At this level, verify that the line number was found in the DCDTL struct and copy over the deal_id, and deal_detail_id into this struct. If the referenced DCDTL record that this DTDTL record belongs to has a type 'M' (vendor-funded markdown), there must be no DTDTL records.
 - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
 - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_DTDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)

Validate_item(): Check if item is a valid item in RMS. Item has to be in approved status, may not be a buyer pack and has to be at or above transaction level.

- Fetch from item_master in RMS to verify the item identifier to be validated exists in RMS.

Validate_item_hier(): Check if item and its merchandise hierarchy is valid in RMS. Item has to be in approved status, may not be a buyer pack and has to be at or above transaction level.

- Fetch from item_master, deps, and groups in RMS to verify the item identifier and its merchandise hierarchy to be validated exists in RMS.

Validate_item_supplier(): Check if item and supplier relationship is valid in RMS.

- Fetch from item_supplier in RMS to verify the item and supplier relationship is valid in RMS.

Validate_loc_hier(): Check if location and its organizational hierarchy is valid in RMS. Location may be a store or a physical warehouse only.

- Fetch a union from wh, store, district, region, and area in RMS to verify the location identifier and its organizational hierarchy to be validated exists in RMS.

Verify_DCDTL(): Check every DCDTL record and verify that:

- If DCDTL record has a billing type of OI (off-invoice) then there are at least one or more DIDTL records for this DCDTL record.
- If DCDTL record is a rebate then there are at least one or more DIDTL records for this DCDTL record.

Find_line_number(): Look in DHDTL then loop through the DCDTL and DIDTL arrays and look for the sequence number given in the argument of this function. Return the index and which array the line number was found in, or return an error message if the argument line number was not found in any of the arrays.

Insert_data(): Does a SQL looped insert from all the data structs with the loop count equaling the struct's record count. Inserts are performed in the following order: DHDTL, DCDTL, DIDTL, PPDTL, DTDTL.

Write_reject(): Writes to the reject file one deal record set with all its sub loop records from the input file.

Scheduling Considerations

This program should run as the first batch of the Deals batch cycle.

Locking Strategy

N/A.

Restart/Recovery

File based.

Performance Considerations

N/A.

Security Considerations

N/A.

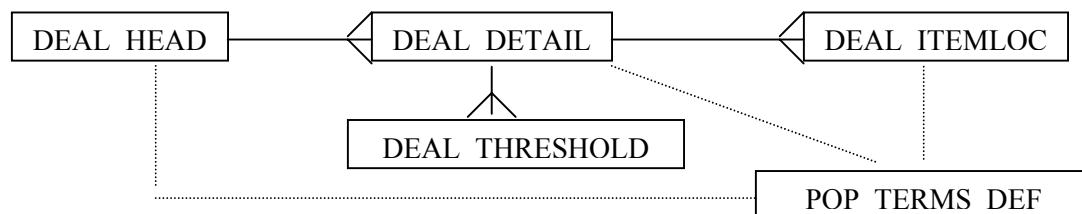
Design Assumptions

All fields that are of type Char in the input file format description above should be left justified and padded with the space character. If the field is not required and no value is being uploaded, the entire field should be padded with the space character.

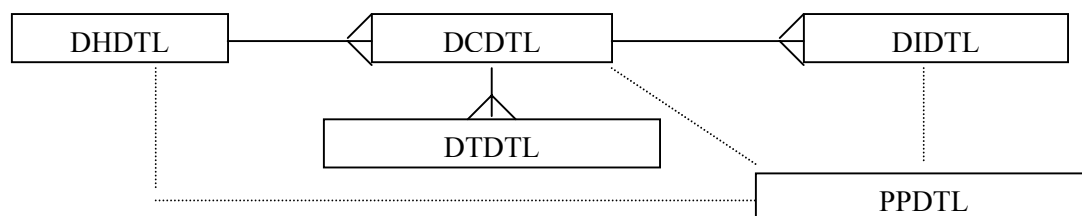
Date fields are always formatted 'YYYYMMDDHH24MISS' (Char(14)).

All fields that are of type Number in the input file format description above should be right justified and padded with 0s. If the field is not required and no value is being uploaded, the entire field should be padded with 0s.

Regarding the ordering of records in the input file, we require records to be looped in the same hierarchy as their corresponding RMS tables are in relation to each other:



Or in terms of the input file:



Note: Pop terms def may be associated with a single deal (FDETL), a deal component (TDETL of DCDTL) or a deal-item-location (TDETL of DIDTL) record. This 'or' type relationship is symbolized by a dashed 'one-to-one' line above.

ALSO NOTE: Buy/Get deals must have only the Buy and Get item(s) on the DEAL_ITEMLOC table, not more, nor less records.

The input file must have the following structure:

```

FHEAD
{
    THEAD      of DHDTL      REQUIRED    for deal head record
    TDETL      REQUIRED      1 deal head record
    TTAIL
    THEAD      of DCDTL      REQUIRED    for deal component records
    [
        TDETL      OPTIONAL    for deal component records
    ]
    TTAIL
    THEAD      of DIDTL      REQUIRED    for item-loc records
    [
        TDETL      OPTIONAL    for item-loc records
    ]
    TTAIL
    THEAD of PPDTL      REQUIRED    for proof of performance records
    [
        TDETL      OPTIONAL    for proof of performance records
    ]
    TTAIL
    THEAD      of DTDTL      REQUIRED    for threshold records
    [
        TDETL      OPTIONAL    for threshold records
    ]
    TTAIL
}
FTAIL

```

The set between the curly brackets may be looped to upload multiple deals from the same file. Within each set, the TDETL records in angle brackets may be sub-looped as a sub-set of the main set.

One set equals one deal. For each set, the first TDETL record is used to upload the DEAL_HEAD (RMS table) table record into RMS. This record is required and maximum one may exist in each set, and it has to be the first THEAD-TDETL-TTAIL record group in the set.

The next THEAD-TDETL-TTAIL group holds the DEAL_DETAIL (RMS table for storing deal component records) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple DEAL_DETAIL records for the same deal.

The next THEAD-TDETL-TTAIL group holds the DEAL_ITEMLOC (RMS table) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple DEAL_ITEMLOC records for the same deal. Note that the line number field in these TDETL records refers to the input file line number of the DEAL_DETAIL record (DCDTL type, as signaled by preceding THEAD record,) to which this DEAL_ITEMLOC record belongs. This

referenced line must be before current line and after current deal's DHDTL record (current deal is most recent DHDTL record in input file).

The next THEAD-TDETL-TTAIL group holds the POP_TERMS_DEF (RMS table for storing proof of performance records) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple POP_TERMS_DEF records for the same deal. Note that the line number field in these TDETL records refers to the input file line number of the DEAL_HEAD, DEAL_DETAIL, or DEAL_ITEMLOC record (DHDTL, DCDTL, or DIDTL type, as signaled by preceding THEAD record,) to which this POP_TERMS_DEF record belongs. Note that a proof of performance record may belong to a DEAL_HEAD, DEAL_DETAIL, or DEAL_ITEMLOC record, not just the DEAL_DETAIL record. This referenced line must be before current line and at or after current deal's DHDTL record (current deal is most recent DHDTL record in input file).

The next THEAD-TDETL-TTAIL group holds the DEAL_THRESHOLD (RMS table) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple DEAL_THRESHOLD records for the same deal. Note that the line number field in these TDETL records refers to the input file line number of the DEAL_DETAIL record (DCDTL type, as signaled by preceding THEAD record,) to which this DEAL_THRESHOLD record belongs. This referenced line must be before current line and after current deal's DHDTL record (current deal is most recent DHDTL record in input file).

Note that if a THEAD-TDETL-TTAIL group has no TDETL records, the THEAD-TTAIL records are still required in the input file, simply no TDETL records will appear between them. This is for explicitly signaling the fact that no such sub-records exist for the set.

Note: for an M type deal, no TDETL records of DIDTL, DTDTL or PPDTL type are allowed. (Vendor-funded markdowns have no records on the DEAL_ITEMLOC, DEAL_THRESHOLD or POP_TERMS_DEF RMS tables.)

Distro price change publish [distropcpub]

Functional Area

Price Changes \ Transfers \ Allocations

Module Affected

Distropcpub.pc (Distro Price Change Publish) will check both Allocations and Transfers for item/location which have undergone a price change. It will then publish this Allocation and Transfer information to their respective queue tables. This corresponds to the integration functionality implemented for Retek 10.

Design Overview

The distropcpub.pc program will get price change information for any allocations and transfers, and write the information to the corresponding queue table. This program will ensure that RDM will have access to any ITEM/location unit retail information that is changed after an allocation or transfer has been downloaded.

This program will loop through the price_hist table, selecting records whose unit retail will change tomorrow, and transaction type is in 4 (Single unit retail was changed) or 11 (Single unit retail and Multi-unit retail were changed). It will then search for allocations and transfers with matching item/locations. When a match is found the program will call existing packages which will insert into the queue table which corresponds to the type of distro it is. The detail records which are written to the queue table 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.

Stored Procedures / Shared Modules (Maintainability)

ALLOC_XML.BUILD_MESSAGE(O_status VARCHAR2, O_text VARCHAR2, O_message CLOB, IO_alloc_msg, I_action_type VARCHAR2)

RMSMFM_ALLOC.ADDTOQ(O_status_code VARCHAR2, O_error_message VARCHAR2, I_message_type VARCHAR2, I_alloc_no NUMBER, I_virtual_from_location NUMBER, I_physical_from_location NUMBER, I_to_loc_type VARCHAR2, I_to_loc NUMBER, I_alloc_status VARCHAR2, I_orig_ind NUMBER, I_message CLOB)

ALLOCDTL_XML.BUILD_MESSAGE(O_status VARCHAR2, O_text VARCHAR2, O_message CLOB, IO_alloc_rec, I_action_type VARCHAR2)

WH_ATTRIB_SQL.GET_PWH_FOR_VWH(O_error_message VARCHAR2, O_pwh NUMBER, I_vwh NUMBER)

TSFDETAIL_XML.BUILD_MESSAGE(O_status VARCHAR2, O_text VARCHAR2, O_message CLOB, I_record, I_action_type VARCHAR2)

RMSMFM_TRANSFERS.ADDTOQ(I_message_type VARCHAR2, I_tsf_no NUMBER, I_item VARCHAR2, I_appr_ind VARCHAR2, I_physical_from_loc NUMBER, I_virtual_from_loc NUMBER, I_from_loc_type VARCHAR2, I_to_loc_type VARCHAR2, L_tsf_message CLOB, L_status VARCHAR2, L_text VARCHAR2)

Input/Output Specifications

N/A

Program Flow

Main()

Init (void)

Process(void)

Loop through price_hist records

Loop through allocation records that match the item\location on price_hist

Loop through transfer records that match the item\location on price_hist

Final()

Function Level Description

Main() – call init() and write to error files.

Init (void) – get vdate, tomorrow's date and period between pick-not-before date and pick-not-after date; initialize restart variables

Size_arrays(void) – allocate memory for array.

Process(void)

Loop through price_hist records (the driving cursor should select records that have a tran_type in 4 (returns) or 11 (markups)), the action_date is tomorrow, and for packs the sellable_ind is 'N'.

Loop through allocation records that match the item\location on price_hist (this cursor should select only allocations which have a qty_allocated greater than the qty_transferred.)

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 write to alloc_mfqueue, because the order has not been downloaded yet.

If order revision exists,

If indicator = 'S' (ie. Processing a single item)

Set the values of the alloc header message rectype,

Call alloc_xml.build_message passing in the record type and 'M' for Modify in the action_type. This function will return a clob that contains an xml message that needs to be inserted on the alloc_mfqueue table which can be accomplished by calling rmsmf_m_alloc.addtoq passing in the clob.

Follow the same steps for the detail record. Call allocdtl_xml instead of alloc_xml.

If indicator = 'P' (ie. processing a pack item component)

Follow the same steps only for the detail record.

Loop through transfer records that match the item\location on price_hist (this cursor should ignore book transfers and non-salable book transfers)

Get the tsf_type, virtual_from_loc, from_loc_type and to_loc_type from tsfhead.

Set the transfer detail record values.

Get the physical wh for the virtual wh on the header record.

Call tsfdetail_xml.build_message passing in the tsf detail record and 'M' for Modify in the action_type.

Pass the clob into rmsmf_m_transfers.addtoq to have the xml message inserted to the tsf_mfqueue table.

Final()

Scheduling Considerations

This program will take the place of allocupd.pc in the schedule. It will be run after pcext.pc.

Locking Strategy

Restart/Recovery

The logical unit of work is a row from the price_hist table, selecting information for all item/location 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 retrieves all items that have price changes in effect tomorrow. It also gets all of the component items of the non sellable packs that have price changes.

```

SELECT distinct 'S',
    '0' pack_no,
    r.item,
    r.loc,
    '||'TO_CHAR(r.loc)
FROM price_hist r,
    v_restart_store v
WHERE r.tran_type in (4,11)
    AND r.loc_type = 'S'
    AND r.action_date = TO_DATE(:ps_tomorrow, 'YYYYMMDDHH24MISS')
    AND v.driver_value = r.loc
    AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
    AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
    AND r.loc > NVL(:ps_restart_to_loc, -9999)
UNION ALL
SELECT distinct 'P',
    vpq.pack_no,
    vpq.item,
    r.loc,
    '||'TO_CHAR(r.loc)
FROM price_hist r,
    v_packsku_qty vpq,
    item_master im,
    v_restart_store v
WHERE r.item = vpq.item
    AND im.item = vpq.pack_no
    AND im.sellable_ind = 'N'
    AND r.tran_type in (4, 11)
    AND r.action_date = TO_DATE(:ps_tomorrow, 'YYYYMMDDHH24MISS')
    AND v.driver_value = r.loc
    AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
    AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
    AND r.loc > NVL(:ps_restart_to_loc, -9999)
ORDER BY 4,
    3;

```

Deal item insert [ditinsrt]

Functional Area

Complex Deals Management

Design Overview

This new batch program will populate the DEAL_SKU_TEMP table with all items that are on non vendor-funded, non PO-specific deals listed on the DEAL_QUEUE table and all items that fall within a hierarchy from these deals. It will get values for the entire merchandise and organizational hierarchies to populate DEAL_SKU_TEMP. The DEAL_SKU_TEMP table will then be used by precostcalc.pc and costcalc.pc to (re)calculate future costs for all listed items.

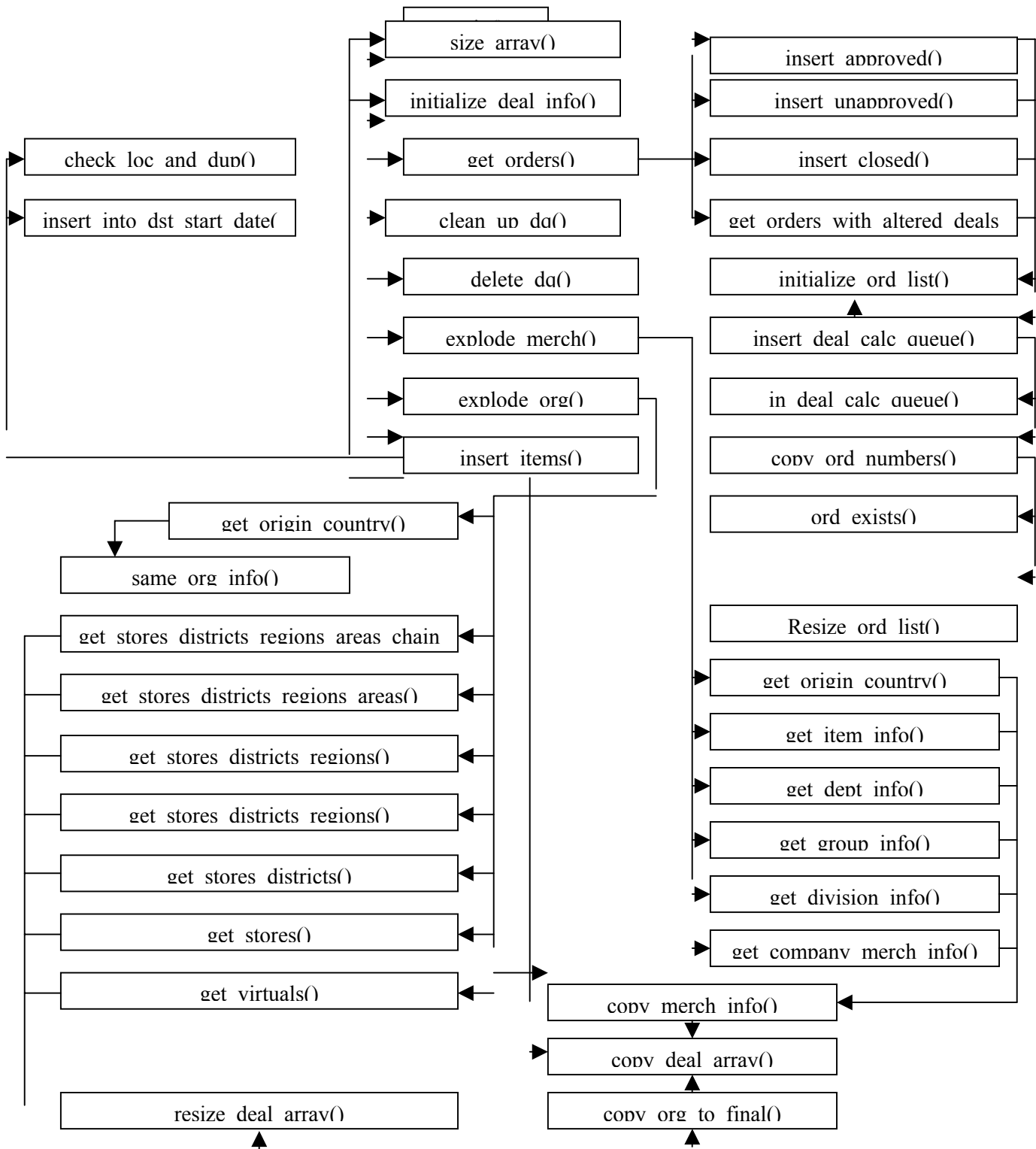
In addition, this program will populate the DEAL_CALC_QUEUE table with orders that may be affected by non vendor-funded, non PO-specific deals that are on the DEAL_QUEUE table (for future processing by orddscnt.pc). Orders that had been applied to deals that no longer apply will also be inserted into the DEAL_CALC_QUEUE table.

The LUW of this module is a single record from the DEAL_QUEUE table and deal definition information that belongs to the DEAL_ID from the DEAL_QUEUE table.

Stored Procedures / Shared Modules (Maintainability)

None

Program Flow



Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init(): Standard Retek init function. Calls retek_init().

Process(): Drives the rest of the program:

- Call size_array() and initialize_deal_info().
- Array-fetch the driving cursor. For each fetched record:
 - If new deal, call delete_dq() and retek_force_commit().
 - If new deal, call get_orders() to populate DEAL_CALC_QUEUE.
 - Call explode_merch().
 - Call explode_org().
 - Call insert_items() to populate exploded merchandise and organizational records into DEAL_SKU_TEMP.
 - Save this deal id so next record's deal id can be compared to previous one.
- Call clean_up_dq().

Explode_merch():

For each record from the deal_itemloc arrays, we need to obtain all missing merchandise hierarchy and origin country and supplier information, as follows:

(Only vendor pack items should be used—skip buyer packs and non-approved, non-transaction level items.)

- If merch_level = 10 get all origin countries for this item/supplier from the ITEM_SUPP_COUNTRY table if an origin country was not given. (We should already have all necessary merchandise hierarchy information). If no supplier was given, we must get all suppliers from the ITEM_SUPP_COUNTRY table that have the same partner as the deal (if the partner_id on deal_head is 'S1', get all suppliers from ITEM_SUPP_COUNTRY where supp_hier_lvl_1 matches the partner id; if it's 'S2', where supp_hier_lvl_2 matches, if 'S3', where supp_hier_lvl_3 matches. If the partner_id is 'S', a supplier is given on DEAL_HEAD; just match to that supplier. This supplier match must be done for all merch levels.
- If merch_level = 9 (have info down to line/diff 2), need to get all items for the supplier/item parent/item grandparent/diff 2 from ITEM_MASTER and ITEM_SUPP_COUNTRY, and all origin countries from ITEM_SUPP_COUNTRY if the origin country was not given. (Remember that item parent, item grandparent, diff 1, or diff 2 could be NULL).
- If merch_level = 8 (info down to line/diff 1), need to get all items (and diff 2 information) for this supplier/style/diff 2 from ITEM_MASTER and ITEM_SUPP_COUNTRY, and all origin countries (if the origin country was not given) from ITEM_SUPP_COUNTRY.
- If merch_level = 7 (info down to line), get all items for this supplier and line, along with diff 1 and diff 2 information and all origin countries (if the origin country was not given) from ITEM_MASTER and ITEM_SUPP_COUNTRY.
- If merch_level = 6 (info down to subclass level), get all item and line/diff info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from ITEM_SUPP_COUNTRY.
- If merch_level = 5 (info down to class level), get all subclass/items/line/diff information info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.

- If merch_level = 4 (info to dept level), get all class/subclass/item/line/diff information info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 3 (info to group level), get all departments for the group from DEPS, get classes/subclasses/items/lines/diffs for each department info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 2 (info to division level) get all groups under that division from GROUPS, and all departments under each group from DEPS. Then get classes/subclasses/items/lines/diffs for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 1 (Company level--ALL items for a supplier), get all divisions from the DIVISION table, get all groups for each division from the GROUPS table, and all departments under each group from the DEPS table. Then get all items and their line and diff info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from ITEM_SUPP_COUNTRY.

Set up as a case statement; each merchandise level value calls a different function:

Case 10: Call get_origin_country().

Case 9, 8, or 7: Call get_item_info().

Case 6, 5, or 4: Call get_dept_info().

Case 3: Call get_group_info().

Case 2: Call get_division_info().

Case 1: Call get_company_merch_info().

Explode_org():

For each record from the merchandise arrays (containing merchandise/country information and all organization information that was present on DEAL_ITEMLOC), get any additional organizational hierarchy information as follows:

- If the org_level = 5 (location) and the given location is a warehouse, the chain, area, region, and district fields should be left as NULL when inserted into DEAL_SKU_TEMP--warehouses aren't part of a district (no additional information needs to be fetched). If multichannel is on, remember to blow out warehouse from DEAL_ITEMLOC (which always holds physical warehouses) to member virtual warehouses.
- If org_level = 5 (location) and the given location is a store, all the necessary information should already be present; no additional information needs to be fetched.
- If org_level = 4 (district), get all stores in this district from the store table. Again, the organizational hierarchy above the district should already have been copied from DEAL_ITEMLOC, so that does not need to be looked up.
- If org_level = 3 (region), get all districts and stores in this region from the district and store tables. Again, the organizational hierarchy above the region should already have been copied from DEAL_ITEMLOC, so that does not need to be looked up.
- If org_level = 2 (area), get all regions, districts, stores for this area from region, district, and store tables. Again, the organizational hierarchy above the area should already have been copied from DEAL_ITEMLOC, so that does not need to be looked up.

- If `org_level = 1` (chain), get all areas, all regions, all districts, all stores for this chain from the area, region, district, and store tables.
- If nothing is given in the organizational hierarchy (`org_level` is null or 0), call `get_stores_districts_regions_areas_chains()` which will blow out the entire organizational hierarchy as it exists in RMS.

Keep track of how many organizational hierarchy combinations were fetched and for what level so that if the next record has the same organization level for the same organization hierarchy element (say, it's org level 4 and the current district is the same as the last district) the organization values can just be copied from the ones previously obtained instead of having to get them from the database again. This check is done by calling `same_org_info()`.

As the information is obtained, it should be inserted into the final array (use `copy_deal_array()` for insert to `DEAL_SKU_TEMP` (be sure to resize this array as necessary).

Set up as a case statement; each organizational level value calls a different function:

Case 5: If location is a store, or the location is a warehouse and the multichannel is turned off in the system, simply copy record, no lookup or blowout is needed. (Call `copy_deal_array()`.)

If the location is a warehouse and the multichannel option is turned on in the system, call `get_virtuals()` to blow out the physical warehouse from `DEAL_ITEMLOC` (which always only holds physical warehouses) to its virtual members.

Case 4: Call `get_stores()`.

Case 3: Call `get_stores_districts()`.

Case 2: Call `get_stores_districts_regions()`.

Case 1: Call `get_stores_districts_regions_areas()`.

Case 0: Call `get_stores_districts_regions_areas_chains()`.

Same_org_info():

This function will check if the previous record whose organizational hierarchy was blown out contains the same organizational level and parameters.

Get_origin_country():

Given an array containing the records retrieved by the driving cursor, find all countries and suppliers for the given item, plus any line and diff information given, and copy the info into the passed output array. This function should only be called if all merchandise info is already known (merchandise level 10), except for the country.

Get_item_info():

Given an array containing the records retrieved by the driving cursor, find all missing merchandise information, and copy the info into the passed output array. This function should only be called if some line and optionally diff information is given, but no item exists (merchandise levels 7-9).

Get_dept_info():

Given an array containing the records retrieved by the driving cursor, find all missing merchandise information and copy the info into the passed output array. This function should only be called if dept is given, but line, diff and item is not given (merchandise levels 4-6).

Get_group_info():

Given an array containing the records retrieved by the driving cursor, find all missing merchandise information and copy the info into the passed output array. This function should only be called if group number is given but dept and everything below is not given (merchandise level 3).

Get_division_info():

Given an array containing the records retrieved by the driving cursor, find all missing merchandise information and copy the info into the passed output array. This function should only be called if division is given, but group number and everything below is not (merchandise level 2).

Get_company_merch_info():

Given an array containing the records retrieved by the driving cursor, find all merchandise information and copy the info into the passed output array. This function should only be called if no merchandise information is given (merchandise level 1).

Get_stores_districts_regions_areas_chains():

Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing in the company and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use organizational information contained in this third array rather than running queries.

Get_stores_districts_regions_areas():

Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given chain and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores_districts_regions():

Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given area and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores_districts():

Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given region and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores():

Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given district and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_virtuals():

Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations existing under the given district and copy them to the output array. (Note that warehouses have no organizational hierarchy.) Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Copy_merch_info():

This function takes three deal record arrays and an index. The first array contains the information gathered from the driving cursor. The second contains the information gathered from the preceding get_<level>_info functions. For each record in the second array, copy all merchandise related information from the second array to the first array. Then insert the combined record into the third array using copy_deal_array().

Initialize_deal_info():

Initializes all indicator variables in every record and sets deal record count to 0 for a passed deal struct.

Insert_items():

Array insert records into the DEAL_SKU_TEMP table, first checking for duplicates (check to make sure that each record does not yet exist in DEAL_SKU_TEMP and that the item-location combinations exist in ITEM_LOC; if check was OK, put it into an array for insert; insert once all have been checked).

To perform the above task:

- Call necessary sizing and initialization functions to allocate final insert array. (Call size_array(), initialize_array().)
- For each record call check_loc_and_dup().
- If previous function call indicated that the record is good to go for the insert, call copy_deal_array() to insert that record into the final insert array.
- Check to see if the current record has a reset date. If it does, copy that reset date into the active date and call check_loc_and_dup() again. If current record is ok to insert, call copy_deal_array() to copy this record into the final insert array. This way we insert the deal's close date for its items into DEAL_SKU_TEMP.
- When records are checked and the final insert array's size is greater than 0, call insert_into_dst_start_date() to array insert the final records.

Check_loc_and_dup():

Check to make sure insert record does not exist in DEAL_SKU_TEMP and check to make sure insert record's item-location relationship does exist in ITEM_LOC. When checking the item-location relationship from the ITEM_LOC table, also verify that the item is not the component of a primary costing pack. Such records are not inserted, only the component's case UPC.

Insert_into_dst_start_date():

Insert passed record into DEAL_SKU_TEMP.

Ord_exists():

Sets return_flag to true if passed ord is already in passed ord_list.

Copy_ord_to_final():

This function takes an output array, an array containing all merch information, and an array containing org information. For each record in the org array, copy information to given record in the merch array, and place this combined record into the output array.

Copy_ord_numbers():

Given to ord_lists and an index, copy value at index from start array to final array if the value does not already exist in the final array.

In_deal_calc_queue():

Set return flag to true if passed ord_num is already in DEAL_CALC_QUEUE.

Insert_deal_calc_queue():

Insert values contained in given ord_list into DEAL_CALC_QUEUE.

Size_ord_list():

Allocate memory for ord_list, and set ord_list count to 0.

Resize_ord_list():

Allocate additional memory for deal array. Allocation is incremented by commit_max_ctr.

Get_orders():

Given a deal id, a deal status, the deal's recalculate approved orders indicator, and the deal start (active) and close dates, do the following:

- If the deal's recalculate approved orders indicator is 'Y' and the status is 'A'pproved, call insert_approved().
- Call insert_unapproved() if the status is 'W'orksheet (it was unapproved and just got set back to worksheet status).
- Call insert_closed() if the status is 'C'losed.
- Call get_orders_with_altered_deals() if the status is 'A'pproved. (This is done for approved deals that had no close date and just got closed, which could disqualify some orders that have already been calculated with this deal from using this deal.)

This will populate the DEAL_CALC_QUEUE table with orders that may be affected by the deal.

Get_orders_with_altered_deals():

Select all approved orders from the ORDHEAD and ORDLOC_DISCOUNT tables whose not before date is higher than the deal close date. Also make sure the contract_no field on ORDHEAD is null. Insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind, and override_manual_ind should be inserted as 'N').

Insert_approved():

Select all approved orders from the ORDHEAD table whose not before date is between the deal start date and the deal close date (or just after the deal start date, if the deal close date is NULL). Also make sure the contract_no field on ORDHEAD is null. Insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind, and override_manual_ind should be inserted as 'N').

Insert_unapproved():

Select all orders from the ORDLOC_DISCOUNT table for the given deal that do not have a status (on ORDHEAD) of 'C'losed, and insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N'). Also make sure the contract_no field on ORDHEAD is null.

Insert_closed():

Select all orders from the ORDLOC_DISCOUNT table for the given deal whose not before date is after the deal's close date (all closed deals must have a close date) and whose status (from ORDHEAD) is not 'C'losed. Also make sure the contract_no field on ORDHEAD is null. Insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N').

Size_array():

Allocate memory for passed array.

Resize_deal_array():

Allocate additional memory for deal array. Allocation is incremented by commit_max_ctr.

Copy_deal_array():

Given two deal arrays and an index, copy deal at index from I_deal_info to o_unique_records. This function will resize o_unique_records if necessary, but it will not check if deal already exists in the final array.

Delete_dq():

Delete processed records from the DEAL_QUEUE table.

Clean_up_dq():

Delete all records left on the DEAL_QUEUE table.

Final():

Performs restart/recovery close logic. Calls retek_close().

Output Specifications

'Table-To-Table'

The following table will be inserted:

DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
DEAL_SKU_TEMP			NONE
			NONE
			Will always be 'N'.
			Will always be 'N'.
			Will always be 'N'.

Scheduling Considerations

This program should run as the first batch program in the deals batch cycle.

Daily record deletion [dlyprg]

Design Overview

The purpose of this program is to delete all of the records in the system marked for delete (by having a record on the DAILY_PURGE table) during the day. Before deleting the records, all relations will be checked to ensure that the record can be deleted. For example, if an item has been marked for delete, this program will check that the item was not put on order later in the day. If relations are found to exist, a record is written to the DAILY_PURGE_ERROR_LOG table. Records on this table will be used to generate a report itemizing any problems found when running this program. If a record is written to the DAILY_PURGE_ERROR_LOG table, meaning that relations exist, the record will not be deleted that night.

After removing records marked for deletion, the program will delete from tables (i.e. MASK_TEMP) that must be cleared out nightly for maintenance reasons.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_PURGE	No	Yes	No	Yes	No
DAILY_PURGE_ERROR_LOG	No	No	No	Yes	Yes
MASK_TEMP	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: PHASE 0 (daily)

Scheduling Diagram: This program must run first to avoid processing deleted entities.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```
EXEC SQL DECLARE c_daily_purge CURSOR FOR
  SELECT  key_value,
          table_name,
          delete_type
  FROM    daily_purge
  ORDER BY delete_order;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Document close [docclose]

Functional Area

Appointments

Module Affected

docclose.pc

Design Overview

This program will be used to close un-Appointed Receipts (POs, Transfers or Allocations), i.e. Receipts that do not have an associated Appointment record within RMS (and therefore cannot be closed via Appointment processing).

Un-Appointed Receipts are recorded on the DOC_CLOSE_QUEUE table. The batch program will retrieve unique Documents from the table and use existing functions to attempt closure for each. ALL entries for any closed Documents will then be deleted from the table.

The LUW will be a Document/Document Type combination.

Stored Procedures / Shared Modules (Maintainability)

APP_DOC_CLOSE_SQL – the functions within this package will close Documents of each type.

Input/Output Specifications

N/A

Program Flow

The program should fetch all *unique* Document-Document Type combinations from the Document Close Queue table. Duplicates should be ignored. The program should attempt to close each Document, using the appropriate functions in the Appointments Document Close package. If a Document is successfully closed, the program should delete *all* instances of that Document from the DOC_CLOSE_QUEUE table.

Function Level Description

Init()

Call retek_init() to get restart-recover variables.

Process()

This function drives the processing of the program. It calls size_arrays() to size the arrays used in this program. The driving cursor is declared and opened here, and then fetched into arrays within a while loop. The cursor retrieves records from DOC_CLOSE_QUEUE, ignoring duplicates. Inside a for loop, the doc_type determines which function is called next. If the doc_type for the doc being processed is 'P'(PO), then close_po() is called. If the doc_type is 'T'(Transfer), then close_tsf() is called. And if the doc_type is 'A'(Allocation), close_alloc() gets called. The current doc is passed into the correct function for processing. Delete_doc() is called when a record has been closed by one of the three close functions above. When the for loop is finished, retek_force_commit() is called, and when the while loop is done, the driving cursor is closed.

Close_po()

This function is called when the doc_type from the driving cursor = 'P'. The PO being processed is passed into the package APPT_DOC_CLOSE_SQL.CLOSE_PO, which makes the appropriate checks to determine if this PO should be closed. If so, the status on ORDHEAD is changed to 'C' and the closed variable is passed back as a value of 1 to tell the program that it should be deleted from DOC_CLOSE_QUEUE. If the PO should remain open, the closed variable will hold a value of 0, and no deletion will occur.

Close_tsf()

This function is called when the doc_type from the driving cursor = 'T'. The current transfer being processed is passed into the package APPT_DOC_CLOSE_SQL.CLOSE_TSF, which makes the appropriate checks to determine if this transfer should be closed. If so, the status on ORDHEAD is changed to 'C' and the closed variable is passed back as a value of 1 to tell the program that it should be deleted from DOC_CLOSE_QUEUE. If the transfer should remain open, the closed variable will hold a value of 0, and no deletion will occur.

Close_alloc()

This function is called when the doc_type from the driving cursor = 'A'. The current doc being processed is passed into the package APPT_DOC_CLOSE_SQL.CLOSE_ALLOC, which makes the appropriate checks to determine if this doc should be closed. If so, the status on ORDHEAD is changed to 'C' and the closed variable is passed back as a value of 1 to tell the program that it should be deleted from DOC_CLOSE_QUEUE. If the doc should remain open, the closed variable will hold a value of 0, and no deletion will occur.

Size_arrays()

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

Delete_doc()

This function is called when any of the records from the driving cursor have been closed. All instances of each record passed in to this function are deleted from DOC_CLOSE_QUEUE.

final():

This function will finish restart/recovery logic.

Scheduling Considerations

This program will ad-hoc.

Restart/Recovery

The logical unit of work is a doc, doc_type combination. The logical unit of work is unique.

Direct store delivery upload [dsdupld]

Functional Area

Purchase Orders

Module Affected

Design Overview

This program is a batch version of the quick order entry form. Some products are delivered to stores directly by vendors without orders issued (soda, bread, chips, etc.). This program will take the information from an input file, validate the information, create an order, apply deals, ship it, receive it, and create an invoice so that all items can be accounted for in the system. An output file will be created with all rejected transactions.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
Period	No	Yes	No	No	No
system_options	No	Yes	No	No	No
unit_options	No	Yes	No	No	No
Sups	No	Yes	No	No	No
Country	No	Yes	No	No	No
Store	No	Yes	No	No	No
Deps	No	Yes	No	No	No
Shipment	No	No	Yes	No	No
Shipsku	No	No	Yes	No	No
Currencies	No	Yes	No	No	No
non_merch_code_head	No	Yes	No	No	No
vat_codes	No	Yes	No	No	No
invc_non_merch_temp	No	Yes	Yes	No	Yes
item_master	No	Yes	No	No	No
item_loc	No	Yes	No	No	No
item_supplier	No	Yes	No	No	No
v_packsku_qty	No	Yes	No	No	No
ordauto_temp	No	No	Yes	No	Yes

Stored Procedures / Shared Modules (Maintainability)

Program Flow

```
main()
|
|--init()
|
|--process()
|  |
|  |--LOOP for each transaction
|  |  |
|  |  |--reset_variables()
|  |  |
|  |  |
```



```

| | | +-free_linked_list_memory()
| | |
| | | +-get_line()
| | | |
| | | | +-add_to_list()
| | | |
| | | +-if FTAIL then
| | | | |
| | | | | +-break out of loop
| | | |
| | | +-process_thead()
| | | |
| | | | +-format_thead()
| | | | |
| | | | +-ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER()
| | | |
| | | +LOOP for each detail record
| | | |
| | | | +-get_line()
| | | | |
| | | | | +-add_to_list()
| | | | |
| | | | +-if TTAIL then
| | | | | |
| | | | | | +-process_ttail()
| | | | | | |
| | | | | | | +-create_order()
| | | | | | | |
| | | | | | | +-ORDER_CREATE_SQL.CREATE_ORDER()
| | | | | | |
| | | | | | | +-apply_deals()
| | | | | | | |
| | | | | | | +-APPLY_DEALS_TO_ORDER()
| | | | | | |
| | | | | | | +-shipment_insert()
| | | | | | | |
| | | | | | | +-Insert into SHIPMENT
| | | | | | |
| | | | | | | +-shipsku_insert()
| | | | | | | |
| | | | | | | +-Insert into SHIPSKU
| | | | | | |
| | | | | | | +-receive_order()
| | | | | | | |
| | | | | | | +-PO_RCV_SQL.RECEIVE_AUTO_PO()
| | | | | | |
| | | | | | | +-if invc_ind = Y and system_options.inv_match_ind = Y and
| | | | | | | | (paid_ind = N or (paid_ind = Y and system_options.sales_audit_ind =
N))
| | | | | | | |
| | | | | | | | +-create_invoice()
| | | | | | | | |
| | | | | | | | +-DIRECT_STORE_INVOICE_SQL.CREATE_INVOICE()
| | | | | | | |
| | | | | | | +-break out of loop
| | | | |
| | | | +-if NMEXP and invc_ind = Y and system_options.inv_match_ind = Y and
| | | | | (paid_ind = N or (paid_ind = Y and system_options.sales_audit_ind =
N))
| | | | |
| | | | | +-process_nmexp()
| | | | | |
| | | | | | +-format_nmexp()
| | | | | | |
| | | | | | +-INVC_SQL.NEXT_INVC_ID()
| | | | |
| | | | +-if TDETL then
| | | | |
| | | | | +-process_tdetl()
| | | | | |
| | | | | | +-format_tdetl()

```

```

| | | | +-process_item()
| | | | | +-if buyer pack sold as eaches then
| | | | | | +-explode_pack()
| | | | | | +-LOOP for each item in pack
| | | | | | | +-process_item()
| | | | | | |--< end loop
| | | | +-else
| | | | | +-SUPP_ITEM_SQL.GET_COST()
| | | | | +-if supp currency not equal to order currency then
| | | | | | +-CURRENCY_SQL.CONVERT()
| | | | | +-if non-sellable pack then
| | | | | | | +-PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL()
| | | | | | +-else
| | | | | | | +-PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL()
| | | | | +-insert into ordauto_temp
| | | | |--< end loop
| | | +-if non-fatal error then
| | | | +-write_reject_to_file()
| | | | | +-free_linked_list_memory()
| | | |--< end loop
| | +-process_ftail()
| +-final()

```

Function Level Description

main()

Standard Retek main function: calls init, process, and final.

init()

Get vdate, invc_match_qty_ind, and dept_level_orders indicators. Do standard restart stuff and process THEAD record.

process()

Loop through each transaction in the input file. For each transaction it will loop through all detail records and call the appropriate function depending on the record type. At the end of each transaction, if any non-fatal errors are encountered the transaction is written to the reject file. Otherwise processing will continue with the next transaction.

process_thead()

Call function to format transaction header and validate all data. Get the next available order number.

format_thead()

Copy line buffer into structure. Do basic formatting validation and set indicators where appropriate.

process_nmexp()

Call function to format non-merch expense record and validate all data. If this is the first non-merch record for this transaction then get the next available invoice number. Check for duplicate non-merch record on `invc_non_merch_temp` and then insert record.

format_nmexp

Copy line buffer into structure. Do basic formatting validation and set indicators where appropriate.

process_tdetl()

Call function to format detail record and validate all data. If item is a reference item then use `item_parent` as item on order. Call function to process the item.

format_tdetl()

Copy line buffer into structure. Do basic formatting validation and set indicators where appropriate.

process_ttail()

Validate line count in transaction. Call functions to create order, apply deals, insert records into shipment and `shipsku`, receive the order, and create an invoice.

process_ftail()

Validate line count in file and write tail to reject file.

get_line()

Fetch a line from input file into a buffer. Validate the record type. Increment counters. If the record is not FTAIL then add this line to the reject file linked list.

reset_variables()

Reset counters and indicators. Call function to free the linked list memory. Delete records from temp tables.

write_reject_to_file()

Write linked list to reject file, increment reject counter, and call function to free linked list memory.

free_linked_list_memory()

Set pointers, release memory, and reset pointers.

add_to_list()

Allocate memory for line/ptr structure and add line to linked list.

create_order()

Call function `ORDER_CREATE_SQL.CREATE_ORDER()`.

apply_deals()

Call function APPLY_DEALS_TO_ORDER().

shipment_insert()

Insert into SHIPMENT.

shipsku_insert()

Insert into SHIPSKU.

receive_order()

Call function PO_RCV_SQL.RECEIVE_AUTO_PO().

create_invoice()

Call function DIRECT_STORE_INVOICE_SQL.CREATE_INVOICE().

process_item()

Get item info. Validate: approved status, not consignment department, item_loc exists, active status, item_supplier exists. If dept_level_orders indicator is Y then ensure all items are in same department. If this is a buyer pack sold as eaches, then call function to explode the pack. Otherwise, get the item cost, convert if necessary, get the item retail, and insert a record into ordauto_temp. This record will be picked up by the ORDER_CREATE_SQL.CREATE_ORDER function.

explode_pack()

Retrieve component items and quantities then call function to process the item/quantity.

Input Specifications

Command Line Parameters:

dsdupld userid/password input_file reject_file

Input File Format

Record Name	Field Name	Field Type	Default Value	Req'd	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Yes	Describes file line type
	Line number	Num(10)	0000000001	Yes	Sequential file line number
	Retek file ID	Char(5)	DSDUP	Yes	Describes file type
	File Date	Char(14)		Yes	Date file was written
THEAD	Record descriptor	Char(5)	THEAD	Yes	Describes file line type
	Line number	Num(10)		Yes	Sequential file line number
	Transaction ID	Num(10)		Yes	Unique transaction id
	Supplier	Num(10)		Yes	Order Supplier
	Origin Country ID	Char(3)		Yes	Supplier's origin country
	Store	Num(10)		Yes	Store
	Department	Num(4)		No	Department – if dept_level_orders = Y then department is

					required
	Currency Code ID	Char(3)		Yes	Order Currency
	Paid Ind	Char(1)		Yes	Indicates if invoice has been paid
	External Reference No	Char(30)		No	\ One of these 3 numbers
	Proof of Delivery No	Char(30)		No	must be entered if
	Payment Reference No	Char(16)		No	/ Invoice Ind = Y
	Payment Date	Char(14)		No	
	Invoice Ind	Char(1)		Yes	Y indicates invoice will be created - if 'Y' then one of the 3 numbers above needs to be entered
	Deals Ind	Char(1)		Yes	Y indicates deals will be applied after creating purchase order
NMEXP	Record descriptor	Char(5)	NMEXP	Yes	Describes file line type
	Line number	Num(10)		Yes	Sequential file line number
	Transaction ID	Num(10)		Yes	Unique transaction id
	Non Merch Code	Char(6)		Yes	Non merchandise code
	Non Merch Amount	Num(20)		Yes	Non merchandise amount
	Vat Code	Char(6)		No	Vat code
	Service Performed Ind	Char(1)		Yes	Service performed ind
TDETL	Record descriptor	Char(5)	TDETL	Yes	Describes file line type
	Line number	Num(10)		Yes	Sequential file line number
	Transaction ID	Num(10)		Yes	Unique transaction id
	Item	Char(25)		Yes	Item – may be item, reference item, or pack
	Qty	Num(12,4)		Yes	Quantity ordered
TTAIL	Record descriptor	Char(5)	TTAIL	Yes	Describes file line type
	Line number	Num(10)		Yes	Sequential file line number
	Transaction ID	Num(10)		Yes	Unique transaction id
	Detail lines	Num(6)		Yes	Number of lines between THEAD and TTAIL

FTAIL	Record descriptor	Char(5)	FTAIL	Yes	Describes file line type
	Line number	Num(10)		Yes	Sequential file line number
	Transaction lines	Num(10)		Yes	Number of lines between FHEAD and FTAIL

Output Specifications

Records will be written to the following tables:

Table Name	Select	Insert	Update	Delete
ordauto_temp	N	Y	N	Y
invc_non_merch_temp	Y	Y	N	Y

Other tables will be written to indirectly by the packages/functions called by dsdupld.

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 will be created by the interface library routine, the detail records will be created using the write_reject_to_file function, and the file trailer will be created by the process_ftail function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and a trailer record and the file line count will be equal to 0.

Scheduling Considerations

Processing Cycle: N/A.

Scheduling Diagram: This upload program is designed to run on an ad-hoc basis.

Pre-Processing: N/A.

Post-Processing: N/A.

Threading Scheme: The program uses single threading.

Locking Strategy

N/A

Restart/Recovery

The logical unit of work will be a transaction consisting of a supplier, origin country, store, and department. The COMMIT_MAX_CTR field on the RESTART_CONTROL table will determine the number of transactions that equal a logical unit of work.

If a non-fatal error is produced, the remaining records in the transaction will be read but not validated, and the invalid record will be rejected and written to the reject file. If a fatal error is returned, then file processing will end immediately. A restart will be initiated from the file

pointer position saved in the RESTART_BOOKMARK string at the time of the last commit point that was reached during file processing.

Increment set system date [dtesys]

Design Overview

Gets a date from the script in run_date() and builds a new row in the PERIOD table after deleting any existing row.

The flag, which indicates that the system is available to users, is maintained here.

Daily:

If user provides a second command line argument, it is used as the new vdate. Otherwise, vdate will increment by 1.

Weekly:

When vdate = next_eow_date_unit, update the SYSTEM_VARIABLES columns last_eow_date_unit and next_eow_date_unit, setting last_eow_date_unit to equal next_eow_date_unit and next_eow_date_unit to equal the next end-of-week date (calculated).

Monthly:

When vdate = next_eom_date_unit, update the SYSTEM_VARIABLES columns last_eom_date_unit and next_eom_date_unit, setting last_eom_date_unit to equal next_eom_date_unit and next_eom_date_unit to equal the next end-of-month date (calculated).

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	Yes	No	Yes
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

EDI location address to vendor download [edidladd]

Design Overview

edidladd will download store and warehouse addresses to vendors. The output file format will be a standard Retek file format that will be translated into EDI format by the Gentran translator. Addresses will be downloaded in two different scenarios. The program will download changes made to store or warehouse addresses into a flat file. Further, if a system_options table flag (addr_catalog) is set to true (Y), then the addresses of all stores and warehouses will be downloaded into a different file to be sent to suppliers. This program will run nightly.

When a store or warehouse address is changed a flag will be set to true to indicate that that location should have its address changes submitted to suppliers. When the changes occur on the location forms (store.fmb or wh.fmb form), the addr_change field is set to true (Y) on the location tables (store or wh). After each changed location address is processed by the EDIDL838.pc program its addr_change flag is reset to N.

The addr_catalog flag for sending address catalogs is set on the sys_ctrl.fmb form. When this is set a catalog listing of all store and warehouse addresses are sent to suppliers, then the addr_catalog field on system_options will be reset to N.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
STORE	Yes	Yes	No	Yes	No
WH	Yes	Yes	No	Yes	No
SYSTEM_OPTIONS	No	Yes	No	Yes	No
PERIOD	No	Yes	No	No	No

Indexes: store (store)
 wh (wh)

Scheduling Constraints

Processing Cycle: PHASE 4 (DAILY)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Because of the lack of volume and the flexibility requirements of EDI, the program will use Retek's standard restart/recovery only minimally. The driving query volume is limited to the volume of the store and warehouse tables. Further, the output files that are created are created if they don't exist and are overwritten if they already exist. In the event of a fatal error it is,

therefore, reasonable to expect clients to simply restart the job from the beginning without recovery.

Driving cursor:

```

SELECT :ps_str_type,
       store.store,
       store.store_name,
       store.store_add1,
       store.store_add2,
       store.store_city,
       store.state,
       store.store_pcode,
       store.country_id,
       store.addr_chg,
       ROWIDTOCHAR(rowid)
FROM store
WHERE store.addr_chg = DECODE(:pi_do_all,1,store.addr_chg,'Y')
UNION ALL
SELECT :ps_wh_type,
       wh.wh,
       wh.wh_name,
       wh.wh_add1,
       wh.wh_add2,
       wh.wh_city,
       wh.state,
       wh.wh_pcode,
       wh.country_id,
       wh.addr_chg,
       ROWIDTOCHAR(rowid)
FROM wh
WHERE wh.addr_chg = DECODE(:pi_do_all,1,wh.addr_chg,'Y');
```

This selects all location information if the catalog is to be processed but only information from locations where the addr_chg indicator is set to Y if the catalog shouldn't be processed. Pi_do_all will be an integer variable that should be initialized as zero and set to 1 just before calling process_catalog.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init()

Get system variables: current date and catalog indicator flag. Open output files: for location address changes, address change rejects, canonical catalog listing, and catalog listing rejects. (only open catalog and catalog reject files if a catalog is to be written). Call restart_init. Set up format strings for output records.

Process()

For each record fetched from the driving cursor, call a function to process location address changes, and if the catalog indicator flag is set to true to write catalog listing of all locations.

Write FTAIL record to file.

Write_records_loc()

This function should write out the address change record to the output file and update the flag on the location table.

Write_records_cat()

This function writes address change records to the catalog file.

Final()

Call restart/recovery close.

I/O Specification

Output file format (all characters should be right-padded with blanks and left justified; all numbers should be left-padded with zeroes and right justified)

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)	Start at 1 and increment	Line number of file
	File type definition	Char(5)	DLADD	Identifies file source
	Purpose code	Char(2)	04 (change) or 05 (replace)	Add/change location or replace whole list
Transaction Detail	File type record descriptor	Char(5)	TDETL	Identifies file record type
	File line sequence	Number(10)	Increment	Line number of file
	Transaction number	Number(10)	Start at 1, increment	Identifies transaction
	Date	Char(8)		Period.vdate YYYYMMDD
	Store or warehouse	Char(2)	ST (store) or WH (warehouse)	Location type
	Location	Number(10)		Store.store or wh.wh
	Location name	Char(20)		Store..store_name or wh.wh_name

	Address line 1	Char(30)		Store.store_add1/wh.wh_add1
	Address line 2	Char(30)		Store_add2 or wh_add2
	City	Char(20)		Store.store_city/wh.wh_city
	State	Char(3)		Store.state/wh.state
	Postal code	Char(10)		Store_pcode/wh_pcode
	Country	Char(3)		Store/wh.country_id
File trailer	File type record descriptor	Char(5)	FTAIL	Identifies file record type
	Total number lines	Number(10)		Total lines in file
	Total no. TDETL lines	Number(10)		(total lines – 2)

Technical Issues

Need to know exact format and parameters required by Gentran.

EDI contract download [edidlcon]

Design Overview

This program downloads a file of EDI contract information. Contracts are only processed if they are in approved status and have an edi_contract_ind of 'Y'.

Changes to make: add restart recovery and make output ONE file instead of one per supplier.
Minor changes to file format (add Gentran ID in FHEAD line and move supplier from FHEAD to THEAD; include transaction number on transaction lines). Let the user enter an output file name.

Scheduling Constraints

Processing Cycle: Daily, Phase 4

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A –file based processing

Restart Recovery

```
SELECT ch.contract_no,
       ch.contract_type,
       ch.dept,
       TO_CHAR(ch.supplier),
       TO_CHAR(NVL(ch.total_cost*1000,0)),
       ch.edi_contract_ind,
       ch.currency_code,
       ROWIDTOCHAR(ch.rowid)
FROM contract_header ch
WHERE ch.status = 'A'
      AND ch.edi_contract_ind = 'Y'
      AND ch.edi_sent_ind = 'N'
ORDER BY ch.supplier
```

Program Flow

N/A

Shared Modules

CONTRACT_SQL.GET_UNIT_COST – get the cost of a contract-item.

Function Level Description

Include the std_rest.h library.

Init:

Get period.vdate.

Call restart_file_init
 Make the format strings for output file lines
 Open output file and write fhead line with the write_head function

Process:

Fetch contract header cursor
 Update contract_header.edi_sent_ind to 'Y'
 write information out to file (call write_detail and write_summary)
 use restart_commit for commits

write_head
 write FHEAD line to file

write_detail

For contract types C and D (no production plans): Get item contract info from the contract_detail table. Get ref_item and ref_item type info from the item_master table. Write TDETL lines to file. For contract types A and B (production plans): Get item contract info from the contract_detail table. Get ref_item info from the item_master table. Write TDETL lines to file. (Note: ready_date, ready_quantity, location_type, and location_number will only have values for contract types A and B. Furthermore, in a multi-channel environment, all TDETL records need to be written at the physical location level.)

write_summary:
 write TTAIL string to file

write_trailer
 write FTAIL string to file

Final:
 Write output to final file and close files.
 Call restart_close.

I/O Specification

Output File layout:
 FHEAD File identification and date
 THEAD Supplier and contract header info
 TDETL Item information
 TTAIL Transaction trailer
 FTAIL File trailer; total number of transactions written

All character variables should be right-padded with blanks and left_justified; all numerical variables should be left-padded with zeroes and right-justified. All dates should be in YYYYMMDDHH24MISS format.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	00000000 01	Sequential file line number

	Gentran ID	Char(5)	DNCN	Identifies transaction type for Gentran
	Current date	Char(14)		Current date period.vdate
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Supplier	Char(10)		Contract_header.supplier
	Contract	No number (6)		Contract_header.contract_no
	Contract type	Char(1)		Contract_header.contract_type
	Department	Number(4)		Contract_header.dept
	Currency code	Char(3)		Contract_header.currency_code
	Total contract cost	Char(20)		Contract_header.total-cost*10000
TDETL	Record descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Item Number Type	Char(6)		Item type for item from item_master table.
	Item Number	Char(25)		item
	Ref Item Number Type	Char(6)		Reference item number type retrieved from item_master.item_number_type.
	Ref Item number	Char(25)		Primary reference item retrieved from Item Master table.
	Diff1 Desc	Char(30)		Diff 1 Description
	Diff2 Desc	Char(30)		Diff 2 Description
	VPN	Char(30)		Vendor Product Number for an item
	Unit cost	Char(2)		Contract_sku.unit_cost*10000 (4 implied decimal places)
The following variables will only have values for contract types of 'A' or 'B';				
	Ready date	Char(14)		Contract_prod_plan.ready_date
	Ready quantity	Char(20)		Contract_prod_plan.qty_ready*10000 (4 implied decimal places)

	Location type	Char(2)		'ST' or 'WH'
	Location number	Char(10)		Contract_prod_plan.store or .wh
FTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number (total # lines in file)
	Contract count	Number(10)		Total number of transactions in file

Technical Issues

N/A

EDI invoice matching debit and credit download [edidldeb]

Design Overview

This program will download debit memo, credit note request, and credit memo data from the invoice matching tables into an ASCII file that will be converted into EDI format by the client and sent via EDI.

Scheduling Constraints

Pre/Post Logic Description

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS		Yes	No	No	No
PERIOD		Yes	No	No	No
SUPS		Yes	No	No	No
INVC_HEAD		Yes	No	Yes	No
INVC_DETAIL		Yes	No	No	No
INVC_MERCH_VAT		Yes	No	No	No
INVC_XREF		Yes	No	No	No

Processing Cycle: Phase 4 (daily)

Scheduling Diagram:

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: no threading

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

The logical unit of work is the invoice.

Driving cursor:

```
SELECT invc_id,
       invc_type,
       partner_type,
       partner_id,
       supplier,
       ext_ref_no,
       ref_invc_id,
       ref_rtv_order_no,
       ref_rsn_code,
       terms,
```

```

        TO_CHAR(due_date, 'YYYYMMDD')
        payment_method,
        TO_CHAR(inv_c_date, 'YYYYMMDD')
        currency_code,
        (exchange_rate * 10000000000),
        (total_merch_cost * 10000),
        (total_qty * 10000,
        ROWIDTOCHAR(ih.rowid)
FROM inv_c_head
WHERE status = 'A'
      AND edi_sent_ind = 'N'
      AND inv_c_type in ('D','R','M')
ORDER BY inv_c_id;

```

(implicitly restartable because the edi_sent_ind gets updated in the program)

Commits should occur every 10,000 records (subject to experimentation)

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage
N/A

Function Level Description

All database interactions required and error handling considerations

init()

- restart/recovery initialization
- Fetch vdate from period and vat_ind from system_options.
- Open temporary output file and write FHEAD line (if program is not being restarted)
- Set up format strings for output file writes
- Call size_arrays to allocate memory for the arrays to be fetched by the driving cursor.

process()

- Fetch the header information (user array fetches of the driving cursor)
- For each credit request/memo or debit memo, write transaction header line to file (call write_THEAD)
- For each credit request/memo or debit memo, call get_total_vat to retrieve totals at each different vat rate on the invoice, if any, and write TVATS lines.. Only do this if VAT is turned on in the system and a cross-reference invoice id is given.
- If a cross-reference invoice id is given, call get_order to get the order numbers related to the invoice, if any.
- For each credit request/memo or debit memo, fetch all related detail records and write those to the file by calling get_items.
- For each credit request/memo or debit memo, write transaction trailer to file (call write_TTAIL)
- Update the edi_sent_ind on the inv_c_head table to 'Y' (use the rowid for the update)

get_items()

For the given invoice id, fetch all records from the invc_detail table, format, and write to output file by calling write_TDETL. Each invc_detail record corresponds to one TDETL line.

write_THEAD()

Write invoice header information line to output file.

write_TVATS()

Write total amount at a given VAT rate information line to output file.

write_TORDS()

Write an order line to output file.

write_TDETL()

Write an item detail line to output file.

write_TTAIL()

Write transaction tail line to output file.

size_arrays()

Use calloc to reserve memory for the arrays to be fetched.

get_total_vat()

See if there are summary-level totals with associated vat rate that apply to the credit memo request/credit memo or debit memo. To do this, check to see if there are records on invc_merch_vat with an invc_id of the invoice that the debit memo or credit request refers to.

get_order()

See if there are order numbers associated with the invoice that the debit memo/credit request refers to by checking the invc_xref table.

final()

Write FTAIL line to file

Restart/recovery close

Write temporary file to final file; remove temporary file; close final file.

I/O Specification

All files layouts input and output

Output file format:

(There will be no TVATS lines if the system does not use VAT or if there are no records on invc_merch_vat; there can be several TVATS lines for each invoice. It is possible to have no TDETL lines or many TDETL lines for each invoice.)

FHEAD (start of file)

THEAD (Credit memo request/debit memo header level info)

TVATS (info from invc_merch_vat table—not present if VAT is turned off)

TORDS (orders associated with a credit request/debit memo)

TDETL (items within a credit request/debit memo)

TTAIL (marks end of credit request/debit memo)

FTAIL (marks end of file)

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

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	DLINV	Identifies which translation Gentran uses
	Current date	Char(14)		YYYYMMDDHH24 MISS format (period.vdate)
THEAD	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction type flag	Char(1)		D=debit memo R=Credit note request M=Credit memo (invc_head.invc_type)
	Invoice id	Number(10)		ID for the credit request or debit memo (invc_head.invc_id)
	Supplier invoice id	Char(30)		Supplier's id for credit/debit invc_head.ext_ref_no (blank if none)
	Cross-reference invoice number	Number(10)		Invoice that a credit note request/debit memo refers to invc_head.ref_invc_id (blank if none)
	Cross-reference RTV order number	Number(6)		Related RTV order number invc_head.ref_rtv_order_no (blank if none)

	Reason code	Char(6)		invc_head.ref_rsn_code (blank if none)
	Partner Type	Char(6)		Debig/credit partner's type invc_head.partner_type (blank if none)
	Partner	Char(10)		Debig/credit partner invc_head.partner_id (blank if none)
	Supplier	Number(10)		Invoice supplier invc_head.supplier
	invoice date	Char(14)		Invc_head.invc_date YYYYMMDDHH24 MISS Date credit/debit memo created
	Due date	Char(14)		invc_head.due_date YYYYMMDDHH24 MISS
	Terms	Char(15)		Invc_head.terms
	Payment method	Char(6)		Invc_head.payment_method LC=letter of credit OA= Open Account WT= wire transfer
	Currency code	Char(3)		Currency code of invoice currency from invc_head.currency_code
	Exchange rate	Number(20)		From the invoice currency to the primary currency. 10 implied decimal places. Blank if none invc_head.exchange_rate
	Total credit or debit merchandise cost	Number(20)		Total cost of all items that are part of the credit or debit (4 implied decimal places) invc_head.total_merch_cost
	Total quantity	Number(12)		Total quantity of items on credit or debit (4 implied

				decimal places) invc_head.total_qty Blank if none

Record Name	Field Name	Field Type	Default Value	Description
TVATS	File record descriptor	Char(5)	TVATS	Marks total cost at given VAT rate line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	VAT code	char(6)		invc_merch.vat_code
	cost excluding VAT	Number(20)		invc_merch_vat.total_cost_excl_vat

Record Name	Field Name	Field Type	Default Value	Description
TORDS	File record descriptor	Char(5)	TORDS	Marks order info line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Order number	Number(8)		invc_xref.order_no

Record Name	Field Name	Field Type	Default Value	Description
TDETL	File record descriptor	Char(5)	TDETL	Marks item info line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Item	Char(25)		Item invc_detail.item (blank if none)
	Ref Item	Char(25)		Reference Item (invc_detail.ref_item) blank if none
	Invoice quantity	Number(12)		How many of this item on credit request/debit memo (4 implied decimal places) invc_detail.invc_qty
	Unit cost	Number(20)		Unit cost of item 4 implied decimal places invc_detail.invc_unit_c

				ost
MAY CHANGE to VAT CODE	VAT rate	Number(20)		VAT rate for this item (10 implied decimal places) (if exists, else blank) invc_detail.vat_rate
	Cost discrepancy indicator	char(1)		invc_detail.cost_dscrpn cy_ind
	Quantity discrepancy indicator	char(1)		invc_detail.qty_dscrpn y_ind
	VAT discrepancy indicator	char(1)		invc_detail.vat_dscrpn y_ind
TTAIL	File record descriptor	Char(5)	TTAIL	Marks end of transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction lines	Number(6)		Total number of TDETL and TVATS lines in transaction
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

EDI purchase order download [edidlord]

Functional Area

Purchase Orders

Module Affected

Design Overview

Orders generated within the Retek system are written to a flat file if they are approved and specified as EDI orders. If shipments are to be pre-marked for cross-dock allocation by the supplier, allocation location and quantities will be sent along with the order information. If the order contains pack items, hierarchical pack information will be sent (this may include outer packs, inner packs, and fashion styles with associated pack templates, as well as component item information). File output is to a Retek standard format file, with the translation to EDI format taking place via an outside translator such as Gentran.

In the past, edidlnew downloaded new orders to an output file, while edidlchg downloaded changed orders. These programs were combined and modified to work with changes that have been made to the ordering tables. The order revision tables and allocation revision table will also be used, to ensure that the latest changes are being sent and to allow both original and modified values to be sent. These revision tables are populated during the online ordering process and the batch replenishment process whenever an order has been approved, and constitute a history of all revisions to the order.

If multi-channel is turned on in the system, the program will sum all quantities to the physical warehouse level for an order before writing the output file.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDHEAD_REV	Yes	Yes	No	No	No
ORDHEAD	Yes	No	No	Yes	No
ORDSKU	Yes	Yes	No	No	No
ORDLOC	Yes	Yes	No	No	No
ORDSKU_REV	Yes	Yes	No	No	No
ORDLOC_REV	Yes	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No
ITEM_MASTER	Yes	Yes	No	No	No
WH	Yes	Yes	No	No	No
ALLOC HEADER	Yes	No	No	No	No
ALLOC DETAIL		Yes	No	No	No
ALLOC DETAIL_REV		Yes	No	No	No
DESC_LOOK		Yes	No	No	No
PACKITEM_BREAKOUT		Yes	No	No	No
SUPS_PACK_TEMPL_DESC		Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)**Program Flow**

Orders that are in approved status and that are new or have had changes made are fetched from the system tables. Additional information about the items on the order and their destination is gathered. Order, item, pack, and shipment information is written to an output file. The system tables are then updated to show that the orders have been sent.

```

main()
|
|--init()
| |
| |--init_terms_array()
| |
|--process()
| |
| |--LOOP for each p.o.
| | |
| | |--if backhaul_type = C then
| | | |
| | | |--calc_backhaul()
| | | |
| | | |--ELC_CALC_SQL.CALC_BACKHAUL_TOTAL()
| | | |
| | |--get_terms_des()
| | |
| | |--write_TORDR
| | |
| | | |--print TORDR to output file
| | |
| | |--write_items
| | |
| | | |--LOOP for each item in p.o.
| | | |
| | | | |--get_item_type()
| | | | |
| | | | |--get_supp_item()
| | | | |
| | | | |--get_ref_item()
| | | | |
| | | | |--write_TITEM()
| | | | |
| | | | |--print TITEM to output file
| | | |
| | | | |--get_pack()
| | | | |
| | | | |--LOOP for each item in pack
| | | | |
| | | | | |--get_supp_item()
| | | | | |
| | | | | |--get_ref_item()
| | | | | |
| | | | | |--write_TPACK()
| | | | | |
| | | | | |--print TPACK to output file
| | | | |
| | | | |--< end LOOP
| | |
| | |--write_shipto()
| | |
| | | |--LOOP for each item/location on p.o.
| | | |
| | | | |--get_item_dims()
| | | | |
| | | | |--write_TSHIP()
| | | | |
| | | | |--print TSHIP to output file
| | |
|

```

```

| | | | | |--< end LOOP
| | | | |
| | | | | +-write_alloc()
| | | | |
| | | | | +LOOP for each allocation record on p.o.
| | | | |
| | | | | +get_item_dims()
| | | | |
| | | | | +write_TSHIP()
| | | | |
| | | | | +print TSHIP to output file
| | | | |
| | | | | |--< end LOOP
| | | | |
| | | | |--< end loop
| | | |
| | | +-write_TTAIL
| | |
| | | +print TTAIL to output file
| | |
| | | +-update ordhead
| | |
| | | +-restart_commit()
| | |
| | | +-restart_file_write()
| | |
| | | |--< end loop
| | |
+-final()
|
+print FTAIL to output file

```

Function Level Description

init()

Get current date. Set up format strings for output file, open output file, and write file header. Set up restart/recovery. Call `init_terms_array` to fetch terms and descriptions from terms table.

process()

Select the new or changed orders in approved status for EDI download by fetching the driving cursor. Eligible orders are approved and have an EDI indicator set. The cursor selects “new” values from `ordhead` and “old” values from `ordhead_rev` for the next to last version (the last version is the current one, with the same information that is now on `ordhead`). For a new order, no earlier version will exist on the `ordhead_rev` table, so no “old” values will be fetched. If old values are null, this must mean that we have a **new** order. Information from different suppliers can be sent in the same file, but the file will be sorted by supplier. If the backhaul type = C (calculated) then call `calc_backhaul` function to calculate the backhaul totals. Call `get_terms_des` to fetch the description of the terms code. Call `write_TORDR` to write order header level information to file. Call `write_items` to fetch additional item-level information and write it to output file. Update the `ordhead` table to show that an EDI transaction has been sent and an acknowledgment has not yet been received.

write_TORDR()

Write the TORDR line (order header level information) to the output file.

write_items()

Get item information (from the `ordsku` and `ordsku_rev` tables--quantity ordered, outstanding quantity, description). Get item cost information and supplier information (by calling `get_supp_item`). If reference item information does not exist on `ordsku`, call `get_ref_item` to fetch

the ref item information (if any). Call `write_TITEM` to write item information line to file. If the item is a pack identifier (`pack_ind = 'Y'` on `item_master`), call `get_pack` to get information on component items within the pack. If the order is to be pre-marked, call `write_alloc` to write allocation information to file; otherwise write shipment information to file by calling `write_shipto`.

write_TITEM()

Write item information line to file.

write_shipto()

Fetch shipment location and quantity information for a particular item on the order from the `ordloc` and `ordloc_rev` tables. Call `get_item_dims` to get the case dimensions then call `write_TSHIP` to write it out to output file.

write_alloc()

This function is called only for cross-docked allocations that will be pre-marked by the supplier. Fetch allocation information from the `alloc_header`, `alloc_detail`, and `alloc_rev` tables, call `get_item_dims`, then call `write_TSHIP` to write it to output file.

get_pack()

Get information on items contained within a pack (from the `packitem_breakout` table). If the item is part of a pack template, fetch the template description from the `supps_pack_tmpl_desc` table. Call `get_supp_item()` to get the `item_supplier` for each of the items. Use `get_ref_item` to fetch ref item information for these items. Call `write_TPACK` to write pack item information lines to file.

write_TPACK()

Write pack component information lines to file.

get_supp_item()

Get supplier VPN, supplier's color code and supplier's size code from the `item_supplier` table.

get_ref_item()

Get ref item info (either primary or preferred for supplier)

write_TSHIP()

Write TSHIP line to file—shipment location and quantity info. Also used to write allocation information

write_TTAIL()

Write order trailer line to file.

calc_backhaul()

Call `ELC_CALC_SQL.CALC_BACKHAUL_TOTAL` to get the backhaul allowance for this order.

init_terms-array()

Fetches all terms and descriptions from terms table so the terms table doesn't have to be joined for each TORDR record.

get_terms_des()

Searches the terms array for a description.

get_item_dims()

Gets case dimensions from item_supp_country_dim.

final()

Write file trailer, copy temporary file to final file (restart/recovery close), close files.

Input Specifications

Command Line Parameters:

edidlord userid/password input_file

For a new order, the “old” fields should be blank. For a changed order, both old and new fields should hold values, if value has changed. “Old” values come from the revision tables for the latest revision before the current one (the last one sent) , while new orders come from the ordering tables.

FHEAD – REQUIRED. File identification, one line per file.

TORDR – REQUIRED. Order level info, one line per order.

TITEM – REQUIRED. Item description, multiple lines per order possible.

TPACK – OPTIONAL. Pack contents, multiple lines per order possible. This line will be written only for pack items.

TSHIP – REQUIRED. Ship to location and quantity, allocation location, multiple lines per item possible. Allocation information is optional on this line—will exist if premark_ind is ‘Y’.

TTAIL – REQUIRED. Order end, one line per order.

FTAIL – REQUIRED. End of file marker, one line per file.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	File head marker
	Line id	Char(10)	0000000001	Unique line id
	Translator id	Char(5)	DLORD	Identifies transaction type
	File create date	Char(14)	Current date	YYYYMMDDHH24MISS format
TORDR	Record descriptor	Char(5)	TORDR	Order header info
	Line id	Char(10)		Unique file line id
	Transaction id	Char(10)		Unique transaction id
	Order change type	Char(2)		‘CH’ (changed) or ‘NW’ (new)
	Order number	Number(8)		Internal Retek order no
	Supplier	Number(10)		Internal Retek supplier id
	Vendor order id	Char(15)		External vendor order no (if available)
	Old order written date	Char(14)		Old date order created YYYYMMDDHH24MISS
	New order written date	Char(14)		Changed date order created YYYYMMDDHH24MISS
	Old Currency Code	Char(3)		Old order currency_code (ISO standard)
	New Currency Code	Char(3)		Changed order currency_code (ISO standard)
	Old Shipment Method of payment	Char(2)		Old ship_pay_method
	New Shipment Method of Payment	Char(2)		Changed ship_pay_method
	Old Transportation Responsibility	Char(2)		Old fob_trans_res
	New Transportation Responsibility	Char(2)		Changed fob_trans_res
	Old Trans. Resp.	Char(45)		Old fob_trans_res_desc

	Description			
	New Trans. Resp. Description	Char(45)		New fob_trans_res_desc
	Old Title Passage Location	Char(2)		Old fob_title_pass
	New Title Passage Location	Char(2)		Changed fob_title_pass
	Old Title Passage Description	Char(45)		Old fob_title_pass_desc
	New Title Passage Description	Char(45)		Changed fob_title_pass_desc
	Old not before date	Char(14)		Old not_before_date YYYYMMDDHH24MISS
	New not before date	Char(14)		Changed not_before_date YYYYMMDDHH24MISS
	Old not after date	Char(14)		Old not_after_date YYYYMMDDHH24MISS
	New not after date	Char(14)		Changed not_after_date YYYYMMDDHH24MISS
	Old Purchase type	Char(6)		Old Purchase type
	New Purchase type	Char(6)		New Purchase type
	Backhaul allowance	Number(20)		Backhaul allowance
	Old terms description	Char(240)		Old terms description from terms table
	New terms description	Char(240)		New terms description from terms table
	Old pickup date	Char(14)		Old pickup date YYYYMMDDHH24MISS
	New pickup date	Char(14)		New pickup date YYYYMMDDHH24MISS
	Old ship method	Char(6)		Old ship method
	New ship method	Char(6)		New ship method
	Old comment description	Char(250)		Old comment description
	New comment description	Char(250)		New comment description
	Supplier DUNS number	Number(9)		Supplier DUNS number
	Supplier DUNS location	Number(4)		Supplier DUNS location
TITEM	File record descriptor	Char(5)		Item info
	Line id	Char(10)		Unique line id
	Transaction id	Char(10)		Unique transaction id
	Item Number Type	Char(6)		Item_number_type
	Item	Char(25)		Item (If a pack item, this will be the pack number)
	Old Ref Item Number type	Char(6)		Item_number_type for old ref_item
	Old Ref Item	Char(25)		Old Ref Item
	New Ref Item Number type	Char(6)		Item_number_type for new ref_item
	New Ref Item	Char(25)		Changed Ref Item
	Vendor catalog number	Char(30)		Supplier_item (VPN)
	Free Form Description	Char(100)		item_desc
	Supplier Diff 1	Char(80)		Supplier's diff 1
	Supplier Diff 2	Char(80)		Supplier's diff 2
	Pack Size	Number(12)		Supplier defined pack size
TPACK	File record descriptor	Char(5)	TPACK	Pack component info
	Line id	Char(10)		Unique line id
	Transaction id	Char(10)		Unique transaction id
	Pack id	Char(25)		Packitem_breakout.pack_no (same as item for the pack item)
	Inner pack id	Char(25)		Inner pack identification
	Pack Quantity	Number(12)		Packitem_breakout.pack_item_qty (4 implied decimal places)
	Component Pack Quantity	Number(12)		Packitem_breakout.comp_pack_qty (4 implied decimal places)
	Item Parent Part Quantity	Number(12)		Packitem_breakout.item_parent_pt_qty (4 implied decimal places)

	Item Quantity	Number(12)		Packitem_breakout.item_qty (4 implied decimal places)
	Item Number Type	Char(6)		Item number type
	Item	Char(25)		Item
	Ref Item Number Type	Char(6)		Ref_item_number_type
	Ref Item	Char(25)		Ref_item
	VPN	Char(30)		Supplier item (vpn)
	Supplier Diff 1	Char(80)		Supplier's diff 1
	Supplier Diff 2	Char(80)		Supplier's diff 2
	Item Parent	Char(25)		Required when Pack Template is not NULL
	Pack template	Char(8)		Pack template associated w/style (packitem_breakout.pack_tmpl_id)
	Template description	Char(40)		Description of pack template (if present) sups_pack_tmpl_desc.supp_pack_desc
TSHIP	Record type	Char(5)	TSHIP	Describes file record-shipment info
	Line id	Char(10)		Unique file line number
	Transaction id	Char(10)		Unique transaction number
	Location type	Char(2)		'ST' store or 'WH' warehouse
	Ship to location	Number(10)		Location value form ordloc (store or wh)
	Old unit cost	Number(20)		Old unit cost (4 implied decimal places)
	New unit cost	Number(20)		New unit cost (4 implied decimal places)
	Old quantity	Number(12)		Old qty_ordered or qty_allocated (4 implied decimal places)
	New quantity	Number(12)		Changed qty_ordered or qty_allocated (4 implied decimal places)
	Old outstanding quantity	Number(12)		Old qty_ordered-qty_received (4 implied decimal places)(or qty_allocated-qty transferred, for an allocation)
	New outstanding quantity	Number(12)		Changed qty_ordered-qty_received (4 implied decimal places)(or qty_allocated-qty transferred, for an allocation)
	Cancel code	Char(1)		
	Old cancelled quantity	Number(12)		Previous quantity cancelled (4 implied decimal places)
	New cancelled quantity	Number(12)		Changed quantity cancelled (4 implied decimal places)
	Quantity type flag	Char(1)		'S'hip to 'A'llocate
	Store or warehouse indicator	Char(2)		'ST' (store) or 'WH' (warehouse)
	Old x-dock location	Number(10)		Alloc_detail location (store or wh)
	New x-dock location	Number(10)		Alloc_detail location (store or wh)
	Case length	Number(12)		Case length (4 implied decimal places)
	Case width	Number(12)		Case width (4 implied decimal places)
	Case height	Number(12)		Case height (4 implied decimal places)
	Case LWH unit of measure	Char(4)		Case LWH unit of measure
	Case weight	Number(12)		Case weight (4 implied decimal places)
	Case weight unit of measure	Char(4)		Case weight unit of measure
	Case liquid volume	Number(12)		Case liquid volume (4 implied decimal places)
	Case liquid volume unit of measure	Char(4)		Case liquid volume unit of measure
	Location DUNS number	Number(9)		Location DUNS number
	Location DUNS loc	Number(4)		Location DUNS loc
	New unit cost init	Number(20)		New unit cost init (4 implied decimal places)
	Old unit cost init	Number(20)		Old unit cost init (4 implied decimal places)
	Item/loc discounts	Number(20)		Item/loc discounts (4 implied decimal places)
TTAIL	Record type	Char(5)	TTAIL	Describes file record – marks end of order
	Line id	Char(10)		Unique file line id
	Transaction id	Char(10)		Unique transaction id
	#lines in transaction	Number(10)		#lines in transaction
FTAIL	Record type	Char(5)	FTAIL	Describes file record – marks end of file
	Line id	Char(10)		Unique file line id
	#lines	Number(10)		Total number of transaction lines in file (not

				including FHEAD and FTAIL)

Output Specifications**Scheduling Considerations**

Processing Cycle: PHASE 4 (may also be schedule ad hoc to run multiple times per day)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Locking Strategy

N/A

Restart/Recovery

Driving cursor:

```

SELECT ROWIDTOCHAR(oh.rowid),
       oh.order_no,
       to_char(oh.supplier),
       to_char(oh.written_date,'YYYYMMDDHH24MISS'),
       to_char(ohr.written_date,'YYYYMMDDHH24MISS'),
       to_char(ohr.not_before_date,'YYYYMMDDHH24MISS'),
       to_char(oh.not_before_date,'YYYYMMDDHH24MISS'),
       to_char(ohr.not_after_date,'YYYYMMDDHH24MISS'),
       to_char(oh.not_after_date,'YYYYMMDDHH24MISS'),
       oh.vendor_order_no,
       ohr.currency_code,
       oh.currency_code,
       ohr.ship_pay_method,
       oh.ship_pay_method,
       ohr.fob_trans_res,
       oh.fob_trans_res,
       ohr.fob_trans_res_desc,
       oh.fob_trans_res_desc,
       ohr.fob_title_pass,
       oh.fob_title_pass,
       ohr.fob_title_pass_desc,
       oh.fob_title_pass_desc,
       oh.pre_mark_ind,
       oh.last_sent_rev_no,
       ohr.purchase_type,
       oh.purchase_type,
       oh.backhaul_type,
       NVL(oh.backhaul_allowance,0) * POWER(10,:pi_qty_dec),
       oh.exchange_rate,
       ohr.terms,
       oh.terms,
       to_char(ohr.pickup_date,'YYYYMMDDHH24MISS'),
       to_char(oh.pickup_date,'YYYYMMDDHH24MISS'),
       ohr.ship_method,
       oh.ship_method,
       ohr.comment_desc,
       oh.comment_desc,

```



```

        s.duns_number,
        s.duns_loc
FROM   ordhead oh,
        ordhead_rev ohr,
        sups s,
        v_restart_supplier v
WHERE  ohr.order_no (+) = oh.order_no
        AND oh.status = 'A'
        AND oh.edi_sent_ind = 'N'
        AND oh.edi_po_ind = 'Y'
        AND oh.supplier = s.supplier
        AND (s.edi_po_chg = 'Y'
            OR (s.edi_po_chg = 'N'
                AND oh.last_sent_rev_no IS NULL))
        AND ohr.origin_type (+) = 'V'
        AND ohr.rev_no(+) = oh.last_sent_rev_no
        and v.driver_name = :ps_restart_driver_name
        and v.driver_value = oh.supplier
        and v.num_threads = :pi_restart_num_threads
        and v.thread_val = :pi_restart_thread_val
ORDER BY 2 , 3;

```

Restart/recovery capability will be used in this program to provide restart capability.

Restartability is implied because the program updates ordhead.edi_sent_ind as records are written out.

EDI sales and stock on hand report download [edidlprd]

Design Overview

The following guideline describes Retek's implementation of the ANSI 852 Product Activity Data transaction set.

The productivity data report is a sales audit summary that will be sent to specified EDI vendors, giving sales details, current stock on hand for all location, and current in transit quantities for each of the Items primarily supplied by that vendor.

This program polls all suppliers that require activity reporting. Those that require daily reports will have daily sales information selected from the edi_daily_sales tables, while those requiring weekly reports will pull information from the ITEM_LOC_HIST and ITEM_MASTER tables to get the weekly sales data. The store level table ITEM_LOC is queried for current stock levels, and the quantities that are in-transit are derived from the transfer tables (via a package call).

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
ITEM_LOC	No	Yes	No	No	No
ITEM_LOC_HIST	No	Yes	No	No	No
ITEM_MASTER	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	No	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Post program will delete old edi_daily_sales records

Threading Scheme: N/A (single threaded)

Restart Recovery

Will be threaded by supplier and the logical unit of work should be supplier.

Supplier Query

```
EXEC SQL SELECT TO_CHAR(s.supplier),
                s.sup_name,
                s.edi_sales_rpt_freq,
                FROM sups s, v_restart_supplier rv
                WHERE s.edi_sales_rpt_freq = 'D'
```

```

AND s.supplier = :ps_restart_driver_val
AND rv.num_thread = :pi_restart_num_thread
AND rv.thread_val = :pi_restart_thread_val
AND s.supplier > NVL(:ps_restart_supplier, -999)
ORDER BY s.supplier;

```

OR

```

EXEC SQL SELECT TO_CHAR(s.supplier),
                s.sup_name,
                s.edi_sales_rpt_freq,
                FROM sups s, v_restart_supplier rv
WHERE s.edi_sales_rpt_freq in ('D', 'W')
      AND s.supplier = :ps_restart_driver_val
      AND rv.num_thread = :pi_restart_num_thread
      AND rv.thread_val = :pi_restart_thread_val
      AND s.supplier > NVL(:ps_restart_supplier, -999)
ORDER BY s.supplier;

```

Program Flow

N/A

Shared Modules

ITEMLOC_QTY_SQL.GET_TRANSFER_QTY –find quantity to transfer

Function Level Description

Init()

In addition to standard restart/recovery logic, this function should retrieve reporting date information. Use the cal_to_454_ldow function to determine the last day of the current week. The current vdate should be retrieved from period. If the vdate = current end of week variable then both daily and weekly reporting will occur, otherwise only daily processing will occur. The edi_sales_rpt_lag value should be retrieved from system_variables, further the current vdate – the lag days will be the reporting date for daily activity reports.

A count of all stores and warehouses should be done to determine the size of the item/location array and the output strings. The output strings (for on hand, in transit, and sales) will have to be dynamically sized to accommodate all location quantities. The string sizes should be total_locations (store + warehouse) * multiplier (macro defined as 20). The size_array() function should be called to size the program arrays.

Open output file and write file header record.

Process()

The driving cursor of this program will be based on supplier. All edi suppliers should be selected according to reporting frequency. If the current date is the end of the reporting week (current 454 day is 7), then both the weekly and the daily reporting suppliers should be selected, otherwise only the daily suppliers should be selected. Call write_tran_head and write_tran_tail to write transaction head and tail records. Call process_items() function to process item detail records.

Process_items()

This function loops through the items on `item_supplier` that are primarily supplied by the current supplier. The `process_report_type` function is called to do updates for reporting suppliers (to stock on hand and in transit quantities on `edi_daily_sales`, for a daily report) and to write to the output file.

`Process_report_type()`:

Writes the TITEM item detail line to the output file via the `write_TITEM` function. If reporting frequency is daily, calls `update_daily_stock` to do database updates and `write_daily_items` to write output information. If reporting frequency is weekly, calls `write_weekly_stock` and `write_weekly_sales` functions.

`Update_daily_stock()`

Stock information for the current item is fetched from the `item/location` tables. The In-transit quantity is derived using the `GET_TRANSIT_QTY` function. The stock information is inserted into the `edi_daily_sales` table with the current `vdate` as the transaction date. If the record already exists on the table (could have been inserted during daily processing of `posupld`), then update the record.

`Write_daily_items()`

All the records on `edi_daily_sales` should be selected into arrays for the current sku and daily reporting date (`vdate - lag days`). Loop through the arrays; for each item location in the loop call the `write_TQUTY` function to write information for each activity type. (Summing to get total quantities will now be done by Gentran, not within this program).

`Write_weekly_sales`

Get weekly sales information from `item_loc` and call `write_TQUTY` to write to output file.

`Write_weekly_stock()`

Get weekly stock information and call `write_TQUTY` to write to output file.

`Write_tran_head()`

Write transaction header to output file. Three dates are included: `vdate-lagdays` for a supplier that reports daily, and beginning or reporting week and end of reporting week for suppliers that report weekly.

Vendor items will be written if exists for non-fashion item, otherwise, Retek item number will be written. Items should have the `item_parent` written if it exists. If it doesn't exist then the Retek ITEM should be written.

`Write_TITEM:`

Writes item number and description information to output file.

Items will have their differences appended to their `item_parent` description.

`Write_TQUTY`

Write location, quantity, and quantity type (in-transit, sales, etc) information to output file.

`Size_arrays()`

Size the following arrays with a `#define` value: supplier array (from driving cursor) and `item_supplier` array (items for supplier w/`primary_supplier_ind = Y`). Sales and stock array (select sales information from `edi_daily_sales/item_loc_hist` tables) should be dynamically sized according the number of stores and warehouses in system.

I/O Specification

Output file structure:

FHEAD file identification REQUIRED. One line per file.

THEAD Dates and supplier REQUIRED. One line per transaction.

TITEM Item information REQUIRED. Multiple lines per transaction possible.

TQUTY Location and quantity information REQUIRED. Multiple liens per transaction possible.

TTAIL End of transaction marker. REQUIRED. One line per transaction.

FTAIL End of file marker. REQUIRED. One line per file.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File record descriptor	Char(5)	FHEAD	Describes record type
	Line number	Number(10)	0000000001	Sequential file line number
	File source	Char(5)	DLPRD	File source id for Gentran
	File create date	Char(8)	Period.vdate	Current date YYYYMMDD
Transaction header	File record descriptor	Char(5)	THEAD	Identifies record type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Report date	Char(8)		Vdate-lag days between reporting (vdate if supplier reports weekly) YYYYMMDD
	supplier	Number(10)		Supplier
Item detail	File record descriptor	Char(5)	TITEM	Identifies file record type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Item	Char(25)		Item
	Item_Num_Type	Char(6)		Item Number Type
	Ref_Item	Char(25)		Reference item
	Ref_Item_Num_Type	Char(6)		Reference Item Number Type
	Vendor catalog number	Char(30)		VPN
	Item description	Char(100)		Item description (item desc)

Quantity detail	File record descriptor	Char(5)	TQUTY	Identifies record type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Quantity descriptor	Char(15)		'On-hand' (stock)/'Sold' (sales)/'In transit'
	Location type	Char(2)		'Store' or 'WH' warehouse
	Location	Number(10)		Retek store or warehouse number
	Unit cost	Number(20)		Unit cost (4 implied decimal places) from item_supplier table (in supplier currency)
	Quantity	Number(12)		Quantity - 4 implied decimal places
Transaction trailer	File record descriptor	Char(5)	TTAIL	Identifies record type
	Line number	Number(10)		Sequential file line number
	Transaction lines	Number(6)		Number of lines for this transaction
File trailer	File record descriptor	Char(5)		Identifies record type
	Line number	Number(10)		Total # lines in file
	Number of transaction lines	Number(10)		Number of transaction lines in file

Technical Issues

N/A

EDI purge [ediprg]

Design Overview

This program will purge rejected EDI “new items” or cost changes on the edi temporary tables. New or changed item information as well as cost change information sent by suppliers will be approved or rejected on-line. The items that are rejected will be removed from the system when they exist for longer than the number of days specified on the system_variables table.

TABLE	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	Yes	No	No	No
EDI_NEW_ITEM	No	No	No	Yes
EDI_COST_CHANGE	No	No	No	Yes

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

This program will use the commit_max_ctr on the restart_control table to periodically commit SQL delete operations.

Program Flow

N/A

Shared Modules

Restart_init() - initialize restart recovery tables

Restart_close() - commit changes and update restart recovery tables

Function Level Description

Init()

This function should call restart_init function to initialize the restart_recovery process. The g_l_restart_max_counter is assigned by the restart_init function from the max_commit_ctr on the restart_control table. This counter will be used in the process function to perform periodic commits. This function will also query the system_variables table to fetch the current values of edi_new_item_days and edi_cost_change_days, and will query the period table for the current vdate.

Process()

This function should delete rejected records from the edi temp tables that are older than the number of days specified by the edi_new_item_days and the edi_cost_change_days variables. A while loop is used to force periodic commits of the deletes based on the value of g_l_restart_max_counter. This is done to ensure that rollback segments will not be exceeded in the event of considerable volume

Final()

This function should call the restart close function to update the restart recovery tables with the final program status (completed if finished successfully, aborted if handled exceptions occur).

I/O Specification

N/A

Technical Issues

N/A

EDI supplier order acknowledgements and changes [ediupack]

Design Overview

This program has four functions: to acknowledge vendor receipt of a buyer generated order without changes, to acknowledge vendor receipt of a buyer generated order with date, cost or quantity modifications, to notify buyer of a vendor-generated order, and to acknowledge order cancellations.

All acknowledgements will update the ordhead table with acknowledgement information.

When the supplier sends the acknowledgement with modifications, they can send the entire purchase order or only the changes. The file details will be matched to the current order. If the Not Before Date, Not After Date, Quantity, Price, and item all match the current order, then no changes were submitted. If one of the variables is blank, for example the price, assume that no pricing changes were made. As soon as one of the variables does not match, the order has been changed. These changes will not be written directly to the order; they will be written to the revision tables. Revisions will be accepted in the ordering dialog and changed orders will be resubmitted via edidlord. The revision number will be the incremented value of the last (or maximum) revision number for revisions of the order on the revision tables.

Vendor generated orders will create new orders by inserting new records on the EDI temporary order tables.

This program uses linked list processing. The linked list should hold one transaction at a time (THEAD, transaction header, through TTAIL, transaction tail). If there is an error with a transaction set, that transaction will be written to the reject file so it can be fixed and reprocessed later, then the program will continue to the next transaction set.

Vendor generated orders:

Item, location, supplier, and cost information are all required for this transaction type. Retek only supports single shipping locations for vendor replenishment. Once all required segments are read and validated, order information will be written to the edi_ord_temp table. In addition, in a multichannel environment the supplier's channel ID must correspond to a virtual warehouse in the physical being shipped to.

Acknowledgement without change:

Items that are read will be read and validated against the current order information. The ordhead table will be updated to reflect that the PO acknowledgement was received.

Acknowledgment with change:

The following fields can be modified on an existing buyer generated order: the not before date, not after date, item cost, qty ordered, and order location (for multiple shipping locations only). Date information that is sent will be compared to the dates stored on the ordhead table. If a difference is detected, a revision will be written to the ordhead_rev table for the order. Cost information that is sent will be compared to the cost stored on the ordloc tables. If a difference is detected then revisions will be written for the item and all of its shipping locations with the new cost information. Quantity information will be compared with the information on the ordloc

table. If a difference is detected in the sent quantity and the stored quantities, then a record is inserted/updated on ordloc_rev with the new order quantity.

New items and shipping locations can also be sent. New item information is added to the ordsku_rev and the ordloc_rev tables. New locations for existing items are inserted to the ordloc_rev tables.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
EDI ORD TEMP	No	Yes	Yes	Yes	No
ITEM MASTER	Yes	Yes	No	No	No
ITEM SUPPLIER	Yes	Yes	No	No	No
ORDHEAD REV	Yes	No	Yes	Yes	No
ORDSKU REV	Yes	No	Yes	Yes	No
ORDLOC REV	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: Ad Hoc in Phases 1,2, or 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Before Open to Buy processing

Threading Scheme: N/A

Restart Recovery

The files will not have enough volume to warrant the implementation of restart recovery for commit/rollback considerations but minimal restart/recovery capability will be added.

Program Flow

N/A

Shared Modules

NEW_ITEM_LOC

NEW_PACK_LOC

PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL

SUPP_ITEM_ATTRIB_SQL.GET_ATTRIB_FOR_ORDSKU

Function Level Description

Init:

Get vdate from period; open input file and reject file.

Process

This function reads and validates the transaction header. Then the item information lines and shipment location lines are read and processed, if present.

The ordhead table will be updated to reflect that a PO acknowledgement has been received after the transaction has been processed.

`get_line:`

Read a line from the input file and add it to the linked list.

`dump_to_reject:`

Read in the rest of a rejected transaction and write it to the reject file.

`validate_THREAD`

This function processes the THREAD line for purchase order acknowledgement. Variables on the line will be validated.

If the acknowledgement is a vendor generated order (vendor replenishment) set acknowledgement type and changed order flags. No more validation or processing will occur in this function for vendor generated orders.

Buyer order acknowledgement, with or without changes, retrieve the supplier number and the acknowledgement flag from ordhead. The edi acknowledgement must not have been received, and the supplier will be validated with the supplier value supplied in the file (if present). If the order is being changed (type AC) then the next revision number is retrieved from the ordloc_rev table for the order.

The supplier number read in this function should be validated against the supplier number retrieved from the tables given the order number. This validation will not occur for vendor generated orders because the supplier number is not fetched in that case.

The dates retrieved will be used for vendor generated orders or order date changes. Vendor generated orders and date changes require two dates, the not before date and the not after date.

`validate_TITEM`

This function processes the item data for purchase order acknowledgements. Item IDs are determined (ITEM, VPN & REF_ITEM). This function will call the process_item function to get required item and VPN information regarding order items.

Also, this function processes the pricing information for purchase order acknowledgements. If the order is vendor generated then no further processing is required. If it is a buyer generated order then further validation and program logic may be required. If the item is new and it has a single shipping location then the add_new_item function is called to add ordsku_rev and ordloc_rev data (If the order has multiple shipping locations then the add_new_item() function will be called from the validate_TSHIP function.) New ordsku_rev and ordloc_rev records will be added for the item/location and other ordloc_rev records will be added via the add_new_loc function for further locations for the new item). If the item already exists, then the check_cost_chg function will be called to determine the sent cost information differs from the current order cost information. If this is the case the insert_record_rev function will be called from the check_cost_chg function.

`validate_TSHIP`

This function processes the destination quantity line for purchase orders. After location and quantity location information is read, the add_new_item function will be called if the item is new to the order. Otherwise the check_qty_chg function is called to see if the sent item/location

quantity differs from the item/location quantity currently held on ordloc for the order. If the quantities differ the update_rev_qty function will be called from the check_qty_chg function to update insert new information on the ordloc_rev table.

cancel_order:

This function inserts into the ordhead_rev, ordloc_rev, and ordsku_rev tables. A zero quantity is written for the sku to be cancelled.

process_item

This function is called after the item information is read from the file. Sub-transaction level items should retrieve the transaction level item number and VPN items (Vendor Catalog) should retrieve the transaction level item number for the VPN and supplier values in the EDI file. Buyer generated orders will check to see if current item is on the ordsku table for the order. If the item does not exist, it is flagged as a new item.

Check_dates()

This function will be called by validate_THREAD. The dates read from the input file will be compared with the dates on the ordhead table. If either date has changed, the call the update_rev_dates() function to add a new revision with the modified not_before_date and not_after_date to the ordhead_rev table.

Check_cost_chg()

This function will be called by validate_TITEM for existing items. The edi cost value read from the input file will be compared with order costs on the ordloc table. If any item/location has a different cost than the passed cost segment element then call the insert_record_rev() to adjust the cost amounts and cost_source on the ordloc_rev records.

Check_qty_chg()

This function will be called the validate_TSHIP function. The sent quantity will be compared with the quantity stored on the ordloc table for the item/location. If the quantities differ or if the record does not exist (meaning the location does not exist for the order) then the update_rev_qty function will be called to update/insert the proper quantities on the ordloc_rev table.

Update_rev_dates()

This function will be called from the check_dates() function when passed a not_before_date and/or not_after_date that differ from the dates on the ordhead table. The function should insert a new record into ordhead_rev with the revised dates (if only one date has been revised, get the original value of the other date off the ordhead table and insert that value instead).

Insert_record_rev()

This function will be called from the check_cost_chg() function when a passed cost amount differs from any item/location cost amount on the ordloc table. The function should insert the order/rev_no/item/location combination into the ordloc_rev table. The cost_source field should be set to 'MANL.'

Update_rev_qty

This function will be called from the check_qty_chg() function when a passed qty amount for an item/location differs from the qty amount on the ordloc table for that order, item, & location. The function should update the revisions tables in the following manner:

- If the `ordsku_rev` entry has not been inserted for the current `order_no/rev_no/origin_type/item` combination, the function `insert_ordsku_rev()` will be called.
- If there is already an entry on `ordsku_rev` for the current `order_no/rev_no/origin_type/item/location` combination, the `qty_ordered` and `qty_received` will be updated for that entry. If there is no entry, a new one will be inserted.

Update_replenish()

This function will be called to add records to the vendor replenishment tables. If the order is a vendor generated then new records will be inserted onto `edi_ord_temp` (or updated on this table if present). Information should be derived from the input file and the item tables.

Add_new_item()

This function will insert new records on the `ordsku_rev` table for an item new to the existing Retek buyer order. Insert new information to the `ordsku_rev` table with the following values: new revision number, 'R' (revision type), edi order number, blank case id, edi sku, edi upc, edi upc supplement, sku description from `win_skus`, null colour description, `order_qty` from input file, 0 for `qty_received`, cost source is 'MANL' if processed CTP segment for item, otherwise it's 'NRML', edi code type is null, prepack indicator is null, assortment indicator is null, and the `rev_date` is the `vdate`.

The `add_new_loc` function should be called to insert new `ordloc_rev` records.

Add_new_loc()

The function will insert records on the `ordloc_rev` tables for new locations on an order. It will be called by the `add_new_item` function and from the `update_rev_qty` function. Determine whether the new location is a store or a warehouse. New information should be added from the edi order. The cost will be the passed cost if the cost was in the file for the item, otherwise it should be the item's unit_retail.

Get_location()

This function will assign values to store and warehouse variables depending on the value of the location value read from the edi input file. The store and warehouse tables should be hit to verify the location's existence.

Get_item_info

This function will retrieve item information for inserts to `edi_ord_temp` and the revisions tables. It is called from `add_new_item`, `add_new_loc`. Item description, dept, and `pack_ind` will be retrieved from the `item_master` table. Cost and retail are retrieved from the appropriate item tables.

get_item_info_wrap:

Call `get_item_info` and also `make_new_loc` if necessary.

make_new_loc:

Call the `NEW_ITEM_LOC` procedure to create a new item/location relationship

get_pack_retail

Call `PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL` to get the unit retail price for a pack

`free_record_memory`

Wipe clean the linked list containing all records on a transaction (stored in case the transaction needs to be rejected)

`add_detail_record_to_list:`

Add a record to the linked list

`get_pack_ind`

Get the `simple_pack_ind` from packhead for a given pack.

`final:`

Restart/recovery close; close input and reject files.

I/O Specification

Input file:

FHEAD file identification REQUIRED—one line per file

THEAD order and acknowledgment info REQUIRED—One line per acknowledgment

TITEM Item information OPTIONAL---multiple lines per acknowledgement possible

TSHIP location and quantity information OPTIONAL—multiple lines per item possible

TTAIL acknowledgment end marker REQUIRED—one line per acknowledgement

FTAIL file end marker REQUIRED—one line per file

Record Name	Field Name	Field Type	Default value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Retek file ID	Char(4)	ORAK	Describes file type
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Acknowledgment type	Char(2)		AP-product replenishment AK-no detail or change (use for UK receive, also) (NEW:CA-cancel order (no detail))
	Order number	Char(15)		May be external order number (vendor order number) OR Retek order number
	Written_date	Char(8)		Written date YYYYMMDD format
	Supplier	Number(10)		Retek supplier number

	number			
	Not before date	Char(8)		Not_before_date YYYYMMDD
	Not after date	Char(8)		Not_after_date YYYYMMDD
	Purchase type	Char(6)		Specifies type of purchase – may be blank
	Pickup date	Char(8)		Pickup_date YYYYMMDD – may be blank
TITEM	Record descriptor	Char(5)	TITEM	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	ITEM	Char(25)		Item **At least one of Item, Ref_item
	Ref_item	Char(25)		Reference Item (for example UPC)
	Vendor catalog number	Char(30)		Vendor item number (VPN)
	Unit cost value	Number(20)		Unit_cost (4 implied decimal places)
	Loc_type	Char(2)		‘ST’ for store, ‘WH’ for warehouse
	Location	Number(10)		If NULL, apply to all locations for this item.
	Pickup location	Char(45)		Location to pick up item – may be blank
TSHIP	Record descriptor	Char(5)	TSHIP	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Store/wh indicator	Char(2)		‘ST’ for store, ‘WH’ for warehouse
	Ship to location	Number(10)		Store or warehouse number
	Quantity	Char(12)		Quantity ordered (4 implied decimal places)
TTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Lines in	Number(6)		Total number lines in

	transaction			this transaction
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number(total # lines in file)
	Number of transactions	Number(10)		Number

Technical Issues

N/A

EDI supplier address upload [ediupadd]

Design Overview

The ediupadd.pc batch program will read vendor/supplier sent EDI 838 Profile Data Files. These files will be processed by vendor/supplier and used to update Retek supplier address information.

Five different types of supplier addresses can be changed via this EDI interface: business, postal, returned to, pick up and payment mailing address. This program always assumes that address information is primary for the address type.

If there is an error with a transaction set, write that transaction to the reject file so it can be fixed and reprocessed later, then the program will continue to the next transaction set. See EDI856 for an example.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
STATE	No	Yes	No	No	No
SUPPLIER	No	Yes	No	No	No
SUPS_ADD	No	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (file based processing—single thread only)

Restart Recovery

Retek restart/recovery capability is minimal. The program uses non-fatal error handling to process input files. There is not enough volume to warrant the use of restart recovery. A commit will not occur until the end of file processing and therefore if fatal errors are encountered updates will not have been committed and the program can be restarted without recovery.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init()

Open input & output files. Validate file header line. Call Retek API restart_init to initialize restart/recovery process.

Process()

Loop through each line of input file. Call validate_FDETL to validate and format variables..
Write to reject file if variables are not valid. Call update_supplier to update the database with the new addresses.

Update_supplier()

Update sups_add table with new, modified address information. This program presumes that address modification information is regarding a “primary” address. If the action specified is “add”, then any existing primary address of that type will be changed so that it is no longer primary, and the new address will be inserted as a primary address of the specified type into the supplier address table. If the action is “update”, then the existing primary address of the specified type will be modified to the new passed information (if one does not exist for that type, it will be inserted).

validate_FDETL—validate supplier and address information

I/O Specification

Input file:

FHEAD File type identification

FDETL Supplier address info

FTAIL End of file marker

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Gentran_id	Char(5)		Identifies the file type
	File create date	Char(14)		YYYYMMDDHH24M ISS format
FDETL	File record descriptor	Char(5)	FDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Add or Update	Char(1)		‘A’dd or ‘U’pdate address
	Address type	Char(2)		Will be translated into sups_add.address_type: 01 – Business 02 – Postal 03 – returns 04 – Pick Up (Order) 05 – Payment
	Supplier	Number(10)		Sups.supplier
	Address line 1	Char(30)		Sups_add.address_1
	Address line 2	Char(30)		Sups_add.address_2

	Address line 3	Char(30)		Sups_add.address_3
	Contact name	Char(20)		Sups_add.contact_name
	Contact Phone	Char(20)		Sups_add.contact_phone
	Contact fax	Char(20)		Sups_add.contact_fax
	City	Char(20)		Sups_add.city
	State	Char(3)		Sups_add.state
	Postal code	Char(10)		Sups_add.post_code
	Country	Char(3)		Sups_add.country_id
FTAIL	File record descriptor	Char(5)		Describes file record type
	Line number	Number(10)		Sequential file line number (total # lines in file)
	Number of transactions	Number(10)		Number of transactions in file

Technical Issues

N/A

Supplier availability for contracts upload [ediupavl]

Design Overview

This program uploads supplier availability.
(The only change needed is to rename the program).

Scheduling Constraints

Processing Cycle: Daily, Phase 1

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A –file based processing

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

EDIUK_SQL.SUPPLIER_EXISTS—check for existence of item on sku_supplier table

EDIUK_SQL.SUP_AVAIL_INSERTS—do inserts and updates to the supp_avail table

Function Level Description

Init:

Open files

- Call restart_file_init

File_process:

Loop:

```

    read from input file into structure array
    validate header and detail fields with validate_record
    process records with process_record
    call restart_file_write and restart_commit
end loop

```

Validate_record:

- Parse record into variables and validate
- Check that item exists on sku-supplier tables by calling EDIUK_SQL.SUPPLIER_EXISTS

Process_record:

Call EDIUK_SQL.SUP_AVAIL_INSERTS to do inserts and updates to supp_avail table

Final:

- Call restart_file_write
- Close files
- Remove temp file

I/O Specification

Upload file format:

FHEAD—file identification

FDETL—supplier, item, available quantity

FTAIL—total number of detail records sent

All character variables should be right-padded with blanks and left_justified; all numerical variables should be left-padded with zeroes and right-justified. All dates should be in YYYYMMDD24MISS format.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	File type	Char(4)	SPAV	
	Create date	Char(14)		File create date YYYYMMDDHH24 MISS format
FDETL	Record descriptor	Char(5)	FDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(14)		Sequential transaction number
	Supplier	Char(10)		
	Item type	Char(3)		'SKU', 'UPC', or 'VPN' item type
	Item id	Char(30)		Actual SKU/UPC/VPN number
	Item supplement	Char(5)		UPC supplement
	Available quantity	Char(12)		Available quantity*10000 (4 implied decimal places
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type

	Line number	Number(10)		Sequential file line number (total # lines in file)
	Number of detail records	Number(10)		Number of FDETL lines in file

Technical Issues

N/A

New and changed upload from supplier [ediupcat]

Design Overview

The purpose of the ediupcat batch program is to update the edi_new_item and edi_cost_change tables. This will allow the users to view and implement the vendor changes online instead of manually viewing and inserting information.

EDIUPCAT will read in a file and strip out the appropriate information. For each line item, the supplier has the option of sending one or all of the following as an item identifier: item, ref_item, and VPN. If an item is sent, this implies that the item exists in Retek. This value is validated against the item tables. Ref_item and VPN are also validated if present. If the item is not present in the file, the program searches Retek for the item. If no item is found, the line item is considered a new item. If either Reference Item or Case Reference Item is provided, its Reference Item Type must be presented as well. To update an existing item in the Retek, the Retek item number or VPN of the item must be presented. The only exception for updating an item using Reference Item number is that the Reference Item number exists in RMS tables.

The supplier can also provide item parent information including Item Parent or Parent VPN to specify the relationship of the new item to the existing Retek item. The item parent's item description and item parent number type are then retrieved from the internal Retek system and inserted to the edi_new_item table.

A new parent VPN may be sent as a regular VPN record. After validating the parent VPN information, it is updated or inserted to the edi_new_item table based on the data processed. In the online form, this record can then be created as a parent item. It is permissible for new items to be sent with parent VPNs that are new to the system, but only if the new parent VPN is also present in the file as a separate VPN record (this constraint is for the purposes of creating a Retek item parent in the EDI Item online form, which will then be applied to all items with the associated parent VPN).

A case pack will be created or updated in the online form, if the supplier provides the Case Reference Item and its associate case information in the EDI file in addition to the item information. For a new item and case pack input, if case cost is not in the input file, it will be calculated by multiplying the item unit cost and the case pack quantity. Otherwise, if item unit cost is not presented in the input file while case cost is provided, the item unit cost will be calculated by dividing the case cost by case pack quantity.

To increase the flexibility of input new items, it is permissible to upload new item information without the unit cost. However, these items will stay at the EDI new item staging table – edi_new_item until the unit_cost is available. The unit_cost can be provided later by the next EDI input file or inserted in the online EDI item form.

All input file information is validated. Any erroneous data will cause the entire transaction to be written to a run-time rejection file that can be reprocessed once the appropriate adjustments are made.

The batch program will have the ability to process multiple transactions per file.

The input file format will be in a Retek standard file format, rather than EDI format. The translation from EDI 888 and EDI 879 (unit cost and case cost) to this standard format will be done by customers using an EDI translation product such as the Gentran translator.

Note: The following text of this design specific to cost change functionality in this program is not included in the March 31, 2001, pre release of RMS10.0, EDI New Item:

For an item that exists in the Retek System (item_supp_country table), the Cost Change of the item will be updated in the edi_cost_change table and then further processed in the online Cost Change Form. Otherwise, no cost information will be updated.

Scheduling Constraints

Processing Cycle: Daily, Phase 2

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: (File-based processing, don't use multithreading)

Restart Recovery

The batch program will use restart/recovery initialization, close, and intermittent commits (restart_commit)

Program Flow

N/A

Shared Modules

SQL_LIB.BATCH_MSG—to write error messages

CURRENCY_SQL.CONVERT_BY_LOCATION—convert unit cost and unit retail

COUNTRY_VALIDATE_SQL.EXISTS_ON_TABLE—validates origin_country_id

SYSTEM_OPTIONS_SQL.GET_ALL_DEFAULTS—retrieves default standard_uom, dimension_uom, weight_uom and packing_method.

UOM_SQL.GET_CLASS—retrieves the class that the UOM exists in

Function Level Description

init()

- get vdate

Restart/recovery initialization

- open input file and read file header
- open output file (run-time reject file)

process()

- Read transaction header Loop:
 - Read transaction detail
 - Call `validate_fdetl` to validate each detail record provided by the input file
 - Call `process_item`:
 - If new item or change to existing item, insert into `edi_new_item`
 - If cost change, update `edi_cost_change`

`format_FDETL()`:

- This function will be modified to format additional columns that are added to the input file (reference the input file for details).

`validate_FDETL()`

- Validate that the input file has at least one of the item, VPN and ref_item fields populated. If none of the above fields exists, issue an error message and return `NON_FATAL`.
- Validate supplier by calling `validate_supplier`.
- When item parent passed in from the input file is not null, call `validate_item_parent`.
- Call `validate_parent_VPN` to validate that the parent VPN exists in the system and find the parent_item according to the parent_vpn.
- If both item parent and parent VPN are not null, compare the input item parent with the item parent retrieved from function `validate_parent_VPN`, if they are different, log an error and return `NON_FATAL`. Otherwise, if the input item parent is null, the item parent retrieved from function `validate_parent_VPN` should be used.
- Call functions `validate_origin_country_id` and `validate_uom`.
- When both ref_item_type and ref_item are presented, call function `check_ref_item` and passing the ref_item and ref_item_type to the function.
- If the item field has value,
 - call function `validate_item`;
 - if the item does not equal ref_item, call function `validate_ref_item`;
 - if the item VPN is not null, call `validate_vpn`.
- If the record's item field does not have a value, process as follows:
 - Call `get_item`
- If item parent is not null, item parent description or item parent number type is null, call function `get_item_info()`. Pass in item parent number and variables to hold the item_parent_desc and item_parent_number_type. Note dummy variables are needed to hold other parameters.
- If both VPN and ref_item are not null and their corresponding item exists in RMS, call function `validate_VPN_vs_ref_item` to make sure that the item is not above the transaction level. Since an item that is above the transaction level could not have a ref_item. (Similar to the scenario in RMS9.0 that a style could not have a UPC). If the function call doesn't return true, return whatever the function returns.
- If the case_ref_item field is not null, call function `process_case`.

`validate_supplier()`:

- First check if the supplier number has value.
 - If it has value, open the cursor `c_val_supp` to validate the supplier as the current code does. If the supplier is found successfully, return true.

- If it doesn't have value, and there are duns number and duns loc in the input, create a cursor `c_val_supp_duns` to select the supplier number from the SUPS table according to the duns_number and the duns_loc. If the supplier number is found, return true. If no supplier number is found, log an error message to state so and return NON-FATAL. This record will then be rejected. If an error happened, return fatal.
- Otherwise, return NON-FATAL to reject the record. An error message should be written to the error file to state the reject reason.

`validate_item()`

Check to see if the item is in the system. If it is not, non-fatal error.

- Create a cursor to validate that the item exists in the item_master table, and is not a sub-transaction item. If item_parent is not null, it needs to be added as a validation criteria. At the same time, retrieve item_parent, item_grandparent, item_number_type, item_level, tran_level and pack_ind in the cursor. If the validation returns No Data Found, issue an error message stating that either the item, or the item/item_parent relation doesn't exist in the system. Return a NON_FATAL error. If the item is a valid Retek item, set the item exists indicator to 1.

`validate_ref_item()`

- Since we know that the ref_item does not equal the item, then the ref_item could either be a sub-transaction item or not exist in the Retek system.
- If the ref_item is a sub_transaction level item, the item_parent found for the ref_item from the item_master table should equal the item that passed in from the input file.
- Create a cursor to perform the above validation. The cursor should select item_parent from item_master table where the item equals the ref_item. If NO DATA FOUND, return true. This means the ref_item might not be stored in RMS. If the item_parent retrieved from the cursor equals the item passed in from the input file, return true. Otherwise, log and error stating that the transaction level item found for the ref_item does not match the item in the record, return NON_FATAL error.

`check_ref_item()`

- Validate for reference item type of UPC-A, UPC-E, EAN8, EAN13 and ISBN.
 - If the reference item type is other than listed above, no validations will be given and function should return success.

`validate_parent_VPN()`

Validate the parent_VPN against the item_supplier and edi_new_item tables.

- If item is found in the item_supplier table, store the value in item_parent and return successfully. Make sure the item_parent returned is unique.
- If item doesn't found in the item_supplier table, further check the parent_vpn against the edi_new_item table where supplier equals ps_supplier and VPN equals the record's parent_vpn. If data is found, return true. Otherwise, issue an error message stating that the parent_VPN does not exist in the system, therefore, the item/item_parent relationship can't be established. Return NON_FATAL.

`validate_origin_country_id()`

- If origin country id on file, call `country_validate_sql.exists_on_table` to validate the origin country. If origin country does not exist, return NON_FATAL error.

`validate_uom()`

- Call function `validate_each_uom()` to validate the following unit of measures when they have values:
 - Standard UOM;
 - Dimension UOMs of case, pallet and item unit;
 - Weight UOMs of case, pallet and item unit;
 - Volume UOMs of case, pallet and item unit.

Passing UOM object (example: case, pallet, etc.), UOM type (standard, dimension, weight, volume) and UOM value to the function. If the call to function `validate_each_uom()` returns fatal or non fatal error, return so.

- Otherwise, if the standard UOM is null, or any of the case, pallet or unit's dimension or weight has value, while their unit of measures are null, default them to the Retek system default UOMs. Call `SYSTEM_OPTIONS_SQL.GET_ALL_DEFAULT_UOM` to get the default unit of measures.
- If the case liquid volume or unit liquid volume has a value, but their unit of measure is null, return NON-FATAL error.

`validate_each_uom()`

This function will accept UOM object, UOM type and UOM value as input parameters. It will call package function `UOM_SQL.GET_CLASS` to validate the passed in UOM value. Check the following conditions:

- If the passed in UOM type is standard, the UOM class is 'PACK' or 'MISC', issue an error message and return a NON-FATAL error.
- If the passed in UOM type is dimension, make sure the UOM class is 'DIMEN'. If it is not 'DIMEN', issue an error message and return a NON-FATAL error.
- If the passed in UOM type is weight and the UOM class found is not 'MASS', issue an error message and return a NON-FATAL error.
- If the passed in UOM type is volume and the UOM class found is not 'VOL' or 'LVOL', issue an error message and return a NON-FATAL error.

`validate_vpn()`

- Validate the vpn against the `item_supplier` table. If the inputted vpn is not found on the table with the item and supplier, return a NON-FATAL error.

`Validate_item_parent()`

- This function will valid the input record's `item_parent` exists in the `item_master` table. It will select `item_desc`, `item_number_type` from `item_master` table where item equals the item parent that passed in from the input file.
- If the `item_parent` doesn't exist, log an error and return NON_FATAL. Otherwise, return true.

`Find_item_by_ref_item()`

- The function will find the transaction level item that corresponding to the `ref_item` (item `ref_item` or case `ref_item`) passed in. It will take `ref_item`, `item` and `item_exists` as parameters.
- Since a `ref_item` could actually be a transaction level item or be a sub_transaction level item, crease a cursor `c_item_by_ref_item`, do a decode selection from `item_master` table to select item from `item_master` table if an item equals the passed in `ref_item` and `item_level` equals `tran_level`, or to select `item_parent` if the item equals the `ref_item` and the `item_level` = `tran_level + 1`.

- If data is found set the item_exist to 1 and store the found item in the passed in variable. Otherwise, set the item_exist to 0. If no error occurred, return true. Otherwise, return fatal.

get_item()

- If item has a diff, we must have the ref_item – if not, non-fatal error
- Pass ref_item to the function find_item_by_ref_item() and also pass in variables to hold the item and the item exists indicator that will be retrieved from the function.
- If the item is not found and VPN is on file, validate the VPN on the item_supplier table
- If the item was retrieved
 - Call get_item_info() to retrieve the item's parent, grandparent, type, description, item level, tran level, and pack indicator.
- If the item was not retrieved, check the edi_new_item table
- If the item was not retrieved, it is a new item

Get_item_info()

- This function will accept an item as input parameter. It'll retrieve the item_parent, item_grandparent, item_number_type, item_desc, item_level, tran_level and pack_ind from the item_master table for the item.

convert_currency()

- Call currency_sql.convert_by_location to convert unit_cost and case_cost into primary currency.

process_item()

- Check the edi_new_item table for the existence of item/supplier/origin_country combo.
- Call convert_currency() to convert currency into primary currency for edi_new_item table.
- If item is not on edi_new_item table
 - If item exists
 - Call process_cost_change() to update/insert edi_cost_chg table.
 - Call insert_new_item() to insert into edi_new_item table – do not insert if item is a pack item.
- If item is on edi_new_item table
 - If item exists
 - Call process_cost_change() to update/insert edi_cost_chg table.
 - Call update_item_info() to update edi_new_item table – do not insert if item is a pack item.

insert_new_item()

The function inserts the item into the edi_new_item table, using the values in the transaction detail record. Unit_cost and case_cost should only be inserted for items not in RMS.

update_item_info()

The function updates the edi_new_item table when a record has not been approved and still in the edi_new_item table. The function updates the following columns:

vdate – processed date

NVL(item_desc, edi_new_item.item_desc)

NVL(short_desc, edi_new_item.short_desc)

NVL(case_cost, edi_new_item.case_cost) – for new items only

NVL(unit_cost, edi_new_item.unit_cost) – for new items only

```

NVL(packing_method, edi_new_item.packing_method)
NVL(gross_unit_weight, edi_new_item.gross_unit_weight)
NVL(net_unit_weight, edi_new_item.net_unit_weight)
NVL(unit_weight_uom, edi_new_item.unit_weight_uom)
NVL(unit_length, edi_new_item.unit_length)
NVL(unit_width, edi_new_item.unit_width)
NVL(unit_height, edi_new_item.unit_height)
NVL(unit_lwh_uom, edi_new_item.unit_lwh_uom)
NVL(unit_liquid_volume, edi_new_item.unit_liquid_volume)
NVL(unit_liquid_volume_uom, edi_new_item.unit_liquid_volume_uom)
NVL(gross_case_weight, edi_new_item.gross_case_weight)
NVL(net_case_weight, edi_new_item.net_unit_weight)
NVL(case_weight_uom, edi_new_item.case_weight_uom)
NVL(case_length, edi_new_item.case_length)
NVL(case_width, edi_new_item.case_width)
NVL(case_height, edi_new_item.case_height)
NVL(case_lwh_uom, edi_new_item.case_lwh_uom)
NVL(case_liquid_volume, edi_new_item.case_liquid_volume)
NVL(case_liquid_volume_uom, edi_new_item.case_liquid_volume_uom)
NVL(gross_pallet_weight, edi_new_item.gross_pallet_weight)
NVL(net_pallet_weight, edi_new_item.net_pallet_weight)
NVL(pallet_weight_uom, edi_new_item.pallet_weight_uom)
NVL(pallet_length, edi_new_item.pallet_length)
NVL(pallet_width, edi_new_item.pallet_width)
NVL(pallet_height, edi_new_item.pallet_height)
NVL(pallet_lwh_uom, edi_new_item.pallet_lwh_uom)
NVL(lead_time, edi_new_item.lead_time)
NVL(min_ord_qty, edi_new_item.min_ord_qty)
NVL(max_ord_qty, edi_new_item.max_ord_qty)
NVL(uom_conversion_factor, edi_new_item.uom_conversion_factor)
NVL(standard_uom, edi_new_item.standard_uom)
NVL(supp_diff_1, edi_new_item.supp_diff_1)
NVL(supp_diff_2, edi_new_item.supp_diff_2)
NVL(supp_pack_size, edi_new_item.supp_pack_size)
NVL(inner_pack_size, edi_new_item.inner_pack_size)

```

Validate_VPN_vs_ref_item():

- This function will validate that the VPN doesn't correspond to an item that is above the transaction level. Compare the item_level with the tran_level (the item tran_level and item_level should have been retrieved in the previous processes), if the item_level is less than the tran_level (item_level above the tran_level), log an error stating that an item above transaction level can't have a ref_item, return NON_FATAL. Otherwise, return true.

process_case()

- First, check if this is a new case pack. Call function find_item_by_ref_item to find the pack no that corresponding to the case_ref_item. Note this indicator will be used to populate the edi_new_item table's new_case_pack_ind field if the case_ref_item is valid. Pass in the case_ref_item to the function and also the variables to hold the pack no and the pack exists indicator. If the pack no is not found in RMS, check to make sure a type for the case_ref_item was specified in the input file. If not, log an error and return NON_FATAL. If pack no is found in the RMS, find the component item from the

packitem table for the pack_no. Compare the pack component item found from the cursor with the item that from the input file, if they are different, log an error and return NON_FATAL.

- Next, compare the case pack exist indicator and the item exist indicator:
 - If both case pack and item are new to RMS, if case_cost is null and unit_cost is provided by the input file, calculate the case_cost by multiplying the unit_cost and the pack_size. Otherwise, if unit_cost is null and the case_cost is presented in the input file, divided the case_cost by the pack_size to populate the unit_cost field.
- Finally, if both of the case_ref_item and case_ref_item_type are not null, call function check_ref_item and pass in the case_ref_item and case_ref_item_type. If the function doesn't return successfully, return whatever is returned from the function. Otherwise, return true.

final()

- restart/recovery close, close files

I/O Specification

Input file structure: (reject file will have same file structure)

FHEAD file header

FDETL item info

FTAIL file trailer

Input Files

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Type Definition	Char(4)	UCAT	Identifies program to use
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'.
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	Transaction sequence	Number(10)		Sequential transaction #
	Supplier	Number(10)		Supplier id#
	Sup Name	Char(32)		Supplier name
	Duns Number	Number(9)		Dun and Bradstreet number identifies the supplier. Note the Duns Number and Duns Loc together, uniquely identifies a supplier.
	Duns Loc	Number(4)		Dun and Bradstreet number identifies the location of the supplier.
	item	Char(25)		Retek item (blank if none)
	Ref item	Char(25)		Reference Item. For example, UPC (blank if none).

	Ref item type	Char(6)		Reference item type. Valid reference types are stored in the code_detail table under Code Type of 'UPCT' and listed as follows: ITEM - Retek Item Number UPC-A - UPC-A UPC-AS - UPC-A with Supplement UPC-E - UPC-E UPC-ES - UPC-E with Supplement EAN8 - EAN8 EAN13 - EAN13 EAN13S - EAN13 with Supplement ISBN - ISBN NDC - NDC/NHRIC - National Drug Code PLU - PLU VPLU - Variable Weight PLU SSCC - SSCC Shipper Carton UCC14 - SCC-14 (blank if none).
	Item Parent	Char (25)		Retek Item Parent which uniquely identifies the item/group at the level above the item.
	Parent VPN	Char(30)		Vendor style id
	VPN	Char(30)		Vendor product number (blank if none) Must be in all capitals
	Supplier item differentiator 1	Char(80)		Item differentiator description. For example, color, size, descriptions. This field is displayed later when entering the item into Retek to use as a basis for choosing an appropriate differentiator within Retek.
	Supplier item differentiator 2	Char(80)		Item differentiator description. For example, color, size, descriptions. This field is displayed later when entering the item into Retek to use as a basis for choosing an appropriate differentiator within Retek.
	Item description	Char(100)		Item description
	Short description	Char(20)		Item short description for point of sales.
	Effective date	Char(14)		Effective date, YYYYMMDDHH24MISS
	Min order qty	Number(12)		Minimum order quantity (4 implied decimal places)
	Max order qty	Number(12)		Maximum order quantity (4 implied decimal places)
	Lead time	Number(4)		Days from PO receipt to shipment
	Unit cost	Number(20)		Unit cost, 4 implied decimal places
	Gross unit weight	Number(12)		Gross unit weight (4 implied decimal places). The gross numeric value of weight per unit.
	Net unit weight	Number(12)		Net unit weight (4 implied decimal places). The net numeric value of weight per unit.
	Unit weight UOM	Char(4)		Item unit weight unit of measure
	Unit length	Number(12)		Item unit length (4 implied decimal places)
	Unit width	Number(12)		Item unit width (4 implied decimal places)
	Unit height	Number(12)		Item unit height (4 implied decimal places)
	Unit lwh UOM	Char(4)		Item unit dimension unit of measure.
	Unit liquid volume	Number(12)		Item unit liquid volume or capacity (4 implied decimal places)
	Unit liquid volume UOM	Char(4)		Unit of measure of the item liquid volume/capacity
	Case ref item	Char(25)		Case reference number. For example: case UPC code.

	Case ref item type	Char(6)		Case reference number type. Valid case reference item types are stored in the code_detail table under Code Type of 'UPCT' and listed as follows: ITEM - Retek Item Number UPC-A - UPC-A UPC-AS - UPC-A with Supplement UPC-E - UPC-E UPC-ES - UPC-E with Supplement EAN8 - EAN8 EAN13 - EAN13 EAN13S - EAN13 with Supplement ISBN - ISBN NDC - NDC/NHRIC - National Drug Code PLU - PLU VPLU - Variable Weight PLU SSCC - SSCC Shipper Carton UCC14 - SCC-14 (blank if none).
	Case item desc	Char(100)		Case item description
	Case cost	number(20)		Case Cost (4 implied decimal places)
	Gross case weight	Number(12)		Gross weight of the case (4 implied decimal places)
	Net case weight	Number(12)		Net weight of the case (4 implied decimal places)
	Case weight UOM	Char(4)		Unit of measure of the case weight
	Case length	Number(12)		Case length (4 implied decimal places)
	Case width	Number(12)		Case width (4 implied decimal places)
	Case height	Number(12)		Case height (4 implied decimal places)
	Case lwh UOM	Char(4)		Case dimension unit of measure.
	Case liquid volume	Number(12)		Case liquid volume or capacity (4 implied decimal places)
	Case liquid volume UOM	Char(4)		Unit of measure of the case liquid volume/capacity
	Gross pallet weight	Number(12)		Gross pallet weight (4 implied decimal places)
	Net pallet weight	Number(12)		Net pallet weight (4 implied decimal places)
	Pallet weight UOM	Char(4)		Unit of measure of the pallet weight
	Pallet length	Number(12)		Pallet length (4 implied decimal places)
	Pallet width	Number(12)		Pallet width (4 implied decimal places)
	Pallet height	Number(12)		Pallet height (4 implied decimal places)
	Pallet lwh UOM	Char(4)		Pallet dimension unit of measure.
	Ti	Number(12)		Shipping units (cases) in one tier of a pallet (4 implied decimal places)
	Hi	Number(12)		Number of tiers in a pallet (height). (4 implied decimal places)
	Pack Size	Number(12)		Supplied pack size. I.e., Number of eaches per case pack. This is the quantity that orders must be placed in multiples of for the supplier for the item.
	Inner pack size	Number(12)		Supplied inner pack size. I.e., Number of eaches per inner container.
	Origin Country ID	Char(3)		Supplied origin country ID.
	Standard UOM	Char(4)		Unit of measure in which stock of the item is tracked at a corporate level.

	UOM Conversion Factor	Number(20)		Conversion Factor, 10 implied decimal places. Conversion factor between an "Each" and the standard_uom when the standard_uom is not in the quantity class (e.g. if standard_uom = lb and 1 lb = 10 eaches, this factor will be 10). This factor will be used to convert sales and stock data when an item is retailed in eaches but does not have eaches as its standard unit of measure.
	Packing Method	Char(6)		Packing Method code (HANG,FLAT)
	Location	Number(10)		RETEK location that the supplier distributes to or this may be a number used by the supplier to identify a non-RETEK location.
	Location Type	Char(1)		This field will contain the type of location ('S' for store and 'W' for warehouse).
	Bracket Value 1	Number (12,4)		This will contain the primary bracket value of the supplier.
	Bracket UOM 1	Char(4)		This field will contain the unit of measure of the primary bracket.
	Bracket Type 1	Char (6)		This field will contain the UOM class.
	Bracket Value 2	Number (12,4)		This will contain the secondary bracket value for the supplier.
	Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket.
	Case Bracket Value 1	Number (12,4)		This will contain the primary bracket value of the supplier for a case UPC.
	Case Bracket UOM 1	Char(4)		This field will contain the unit of measure of the primary bracket for a case UPC.
	Case Bracket Type 1	Char (6)		This field will contain the UOM class for a case UPC.
	Case Bracket Value 2	Number (12,4)		This will contain the secondary bracket value for the supplier for a case UPC.
	Case Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket for a case UPC.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Numeric ID(10)		Number of records/transactions processed in current file (only records between head & tail)

Test Conditions

Conditions	Expected Results	Programmer Sign-off
No records	no processing	
Missing required information	write TDETL line to reject file	
<i>Process a valid input file:</i>		
for a new item	Insert edi_new_item record – include the unit retail and cost	
for existing items	Insert edi_new_item record, if changes to other than cost. Only insert into edi_cost_change if cost change present	

for a new item with existing edi_new_item and edi_cost_change records	Update the edi_new_item and edi_cost_change tables	
<i>Input file contains item, ref_item, and VPN:</i>		
invalid ref_item	write TDETL to reject file	
invalid vpn	write TDETL to reject file	
<i>Input file contains ref_item and VPN:</i>		
invalid ref_item	write TDETL to reject file	
invalid vpn	write TDETL to reject file	
item with no ref_item (in Retek) but a valid VPN	Insert/update edi_new_item and edi_cost_change tables	

EDI invoice upload [ediupinv]

Design Overview

This program will upload invoice and credit memo information from an ASCII file (which will be translated into Retek standard format by the client from an EDI file) into the invoice matching tables.

Scheduling Constraints

Pre/Post Logic Description

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	Yes	Yes	No	No	No
PERIOD	Yes	Yes	No	No	No
ORDHEAD	Yes	Yes	No	No	No
UPC_EAN	No	Yes	No	No	No
INVC_HEAD		Yes	Yes	No	No
INVC_DETAIL		No	Yes	No	No
INVC_XREF		No	Yes	No	No
INVC_MATCH_QUEUE		No	Yes	No	No
INVC_MATCH_WKSH T		No	Yes	No	No
INVC_NON_MERCH		No	Yes	No	No
INVC_MERCH_VAT		No	Yes	No	No
TERMS		Yes	No	No	No
VAT_CODES		Yes	No	No	No
RTV_HEAD		Yes	No	No	No

Processing Cycle: Phase I (daily)

Scheduling Diagram:

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A –file based.

Restart Recovery

Logical Unit of Work (recommended Commit check points)
Driving Cursor

The logical unit of work is the invoice.
Commits should occur every 2500 records (subject to experimentation).

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

INVC_SQL.NEXT_INV_C_ID—get a new invoice id

INVC_SQL.APPLY_XREF—given an order number, insert that order and received shipments (physical shipments) associated with the order into invc_xref

Function Level Description

All database interactions required and error handling considerations

Set up structures for each different type of file line.

Input and reject file names should be accepted from the command line at runtime.

init()

Restart/recovery initialization

Fetch vdate from period and vat_ind and invc_match_qty_ind from system_options.

Vdate will be used as the create date.

Open input file and reject file

Read in FHEAD line (or seek to the right place in the file, if doing a restart)

process()

Read in transactions from file (use get_line). As lines are read in, save to a linked list for rejection if necessary. For each transaction:

- 1) Deal with header-level information (THEAD line)—format the line (format_THEAD), do basic validation (validate_THEAD), get a new invoice number (get_new_inv_c_id) and insert into invc_head, invc_match_queue, and invc_non_merch tables.
- 2) Deal with VAT information (TVATS lines—format_TVATS, validate_TVATS). If VAT is turned on in the system, call insert_merch_vat. (if turned off, read through the lines but don't process).
- 3) Deal with cross reference-level information (TXREF lines – format_xref, validate_xref). If order numbers exist, call insert_order_xref to insert the order number into invc_xref table. If ASN numbers exist, loop through all shipment numbers containing the ASN number in the ext_shipment field on the shipment table and call insert_shipment_xref to insert the shipment numbers into the invc_xref table.
- 4) Deal with item_level information (TDETL line). Format, validate, insert into invc_detail table.
- 5) Deal with ASN/carton information (TSHIP line). Match ASN to shipment in RMS. Format, validate, insert into invc_match_wksht. Carton is now optional.
- 6) When get to TTAIL line, check number of lines in transaction from file against number of lines read in by program. If they don't match, reject the transaction.

- 7) If a transaction failed validation anywhere, roll back to a savepoint set at the beginning of the transaction and write the transaction to the reject file (write_to_reject)
- 8) When transaction has been processed, call restart commit logic if necessary.

get_new_inv_id()

Call the invc_sql.next_inv_id package to get a new invoice number

process_THEAD()

Format and validate header variables and do necessary table inserts.

process_TXREF()

Format and validate cross-reference variables and do necessary table inserts into the invc_xref table.

process_TDETL()

Format and validate item-level variables and do necessary table inserts.

process_TVATS()

Format and validate TVATS variables and insert into invc_merch_vat.

process_TSHIP()

Format and validate TSHIP variables and insert into invc_match_wksht.

process_FTAIL()

Validate FTAIL line from input file and create new FTAIL line for reject file (the total number of transaction lines will be different for the reject file).

format_header_fields()

parse THEAD line into variables

format_tvats_fields()

parse TVATS line into variables

format_txref_fields()

parse TXREF line into variables

format_detail_fields()

parse TDETL line into variables

Things that must be validated in validate_THEAD, validate_TVATS, validate_TXREF, and validate_TDETL: (if validation fails, reject transaction but continue processing file)

- item valid (check item_master) (If only given Ref. Item, get transaction –level item from item_master)
- partner/partner type valid if given (use validate_partner_id, and check against partner table)
- merchandise invoices cannot be associated with a partner, and must have supplier
- credit notes from a partner cannot have item records attached to it unless partner type is in (S1,S2,S3).
- order number valid if given (check ordhead)
- reference invoice id valid if given (check invc_head)

- RTV order number valid if given (check against `rtv_head`)
- if total qty given, individual item qtys sum to total (only need to check this if `invc_match_qty_ind` is 'Y' on `system_options`)
- total merchandise cost on THEAD line matches sum of costs from TDETL lines (sum of unit cost * qty)

(keep a running sum of individual qtys/costs; if don't match at end of transaction, reject transaction and rollback to savepoint)

- Credit notes cannot have both a reference invoice id and a reference RTV
- If reference invoice and reference RTV are null, then no terms are required. If they are included, set due date to later of system date and `sup_invoice_date` (use `set_due_date`).
- If Item is not given, must have Ref. Item.
- dates are valid dates, numbers are all numeric
- If terms not given, default to supplier terms (from `sup` table)
- terms must exist on terms table
- VAT code must exist on `vat_codes` (validate on THEAD, and TVATS lines—use a separate function `validate_VAT` for this and pass in the VAT code)
- `Addr_key` must exist in the `addr` table for the partner or supplier given, if not given, will default to primary address of partner or supplier
- `Paid_ind` needs to be Y or N, if not given will default to that on the `sup` table corresponding to the supplier. If only a partner is given, `paid_ind` will default to N.

validate_THEAD()

Validate THEAD line variables as needed

validate_TVATS()

Validate VAT code from TVATS line.

validate_TXREF()

validate TXREF line variables as needed.

validate_TDETL()

validate TDETL line variables as needed.

If Ref. Item is given but Item isn't, get the transaction-level Item from the `item_master` table

validate_order(order_no)

Check that a given order no exists on the `ordhead` table

validate_shipment(ship_no)

Match ASN to shipments in RMS.

validate_carton(ship_no, carton_no)

Check that the given `ship_no`/`carton_no` exists on the `shipsku` table. Use shipments that matched from the ASN. Note that carton is now optional in RMS.

validate_vatcode(vat_code)

Check that a given VAT code exists on the `vat_codes` table

validate_partner_type

Check that given `partner_type` is legal.

validate_addr_key

Check that given addr_key is legal.

get_supplier_terms

Fetch terms from the sups table for the given supplier, if terms not given in file.

get_addr_key

Fetch addr_key from the addr table, if not given in file.

get_paid_ind

Fetch paid_ind from the sups table, if not given in file.

set_due_date()

Sets due date to later of system date and sup_inv_date.

validate_terms()

Check that terms exists on terms table.

insert_header()

Insert a record into the invc_head table.

A credit memo is treated as matched from the moment it is created (since it must refer to another invoice or a return to vendor number). Regular merchandise invoices and non-merchandise invoices are created in unmatched status.

Need to check for duplicate invoices (duplicate supplier invoice number/(supplier)/(partner type/partner id)/supplier invoice date) . If a duplicate is found, reject the duplicate transaction, but continue processing the rest of the file.

The due date should be calculated as follows if terms are given:

Select duedays from the terms table for the terms from the file. Due_date = invoice date + duedays.

The terms discount percent should be taken from the terms table if it is not in the file.

Values to insert into invc_head should be as follows:

INVC_ID	from invc_sql.next_inv_id
INVC_TYPE	'I' for merchandise invoice, 'C' for credit note, 'N' for non-merchandise invoice (from file)
SUPPLIER	supplier from file
EXT_REF_NO	supplier invoice id from file
STATUS	'U' for merchandise and non merchandise invoices, 'M' for credit invoices
EDI_INVC_IND	'Y'
MATCH_FAIL_IND	'N'
REF_INVC_ID	null for invoice, reference invoice number for credit note if given
REF_RTV_ORDER_NO	null for invoice, RTV order number for credit note if given
REF_RSN_CODE	null
TERMS	terms from file (or terms table if not given in file)
DUE_DATE	calculated as above
PAYMENT_METHOD	from file

TERMS_DSCNT_PCT	calculate as above
TERMS_DSCNT_APPL_IND	from file
FREIGHT_TERMS	from file
CREATE_ID	'EDIUPINV'
CREATE_DATE	vdate
INVC_DATE	from file
MATCH_ID	(null)
MATCH_DATE	(null)
APPROVAL_ID	(null)
APPROVAL_DATE	(null)
FORCE_PAY_IND	'N'
FORCE_PAY_ID	(null)
POST_DATE	(null)
CURRENCY_CODE	from file
EXCHANGE_RATE	from file if given, null otherwise
TOTAL_MERCH_COST	from file
TOTAL_QTY	from file if given
EDI_SENT_IND	'N'
COMMENTS	(null)
PARTNER_TYPE	from file
PARTNER_ID	from file
ADDR_KEY	from file if given, addr table otherwise
PAID_IND	from file if given, sups table otherwise

insert_detail()

Insert a record into the invc_detail table.

Default invoice unit cost to original unit cost , invoice qty to original qty, invoice vat rate to original vat rate ("original" comes from the invoice file)

Values should be as follows:

INVC_ID	created by program
ITEM	from file
REF_ITEM	only if given in file
INVC_UNIT_COST	original cost from file
INVC_QTY	original qty from file
INVC_VAT_RATE	original vat rate from file (if given)
STATUS	'U' (except for credit memo, which will be 'M')
ORIG_UNIT_COST	from file
ORIG_QTY	from file
ORIG_VAT_RATE	from file
COST_DSCRPNY_IND	'N'
QTY_DSCRPNY_IND	'N'
VAT_DSCRPNY_IND	'N'
PROCESSED_IND	'N'
COMMENTS	(null)

insert_wksht()

Insert a record into the invc_match_wksht table.

Values should be as follows:

INVC_ID	created by program
ITEM	from file
INVC_UNIT_COST	from file
SHIPMENT	matched from ASN(s) in file
CARTON	from file (carton is now optional)
INVC_MATCH_QTY	(null)
MATCH_TO_COST	If no ORDLOC_INVC_COST records exist for the order/item/loc, this field will contain the SHIPSKU cost. Otherwise it will be the ORDLOC_INVC_COST cost.
MATCH_TO_QTY	If no ORDLOC_INVC_COST records exist for the order/item/loc, this field will contain the SHIPSKU qty. Otherwise it will be the ORDLOC_INVC_COST qty.
MATCH_TO_SEQ_NO	If no ORDLOC_INVC_COST records exist for the order/item/loc, this field will be NULL. Otherwise it will contain the corresponding sequence number from ORDLOC_INVC_COST.

insert_order_xref(order_no)

(Only call this function if an order number was given—only call once for each distinct order number for each invoice)

Call INVC_SQL.APPLY_XREF, which will insert the order number and all received shipments with that order number into the invc_xref table. The apply_to_future_ind parameter should be 'Y'.

insert_ship_xref(shipment)

Make sure the shipment has been received. (If not, reject transaction). Check to make sure that a record with this shipment and invoice does not already exist on the invc_xref table; if it does not, insert (apply_to_future_ind should be 'N', everything else except invoice_id and shipment will be null).

insert_match_queue()

Insert a record into the invc_match_queue table with the new invoice ID. (Don't call this function for credit memos; they don't need to go into the match queue.)

insert_non_merch()

If a sales tax is given on the header line, insert it into the invc_non_merch table (with a non_merch_code of 'T'); if a freight charge is given, insert it into invc_non_merch with a non_merch_code of 'F'; if a miscellaneous charge is given, insert with a code of 'M'. If VAT codes are given, insert them.

insert_merch_vat()

Insert a record into the invc_merch_vat table. Processing should continue with no error if duplicate values are found.

write_to_reject()

Write record to reject file (use linked list)

get_line()

Read in a line from input file and determine the line type.

reset_variables()

Set counters back to zero.

final()

Restart/recovery close
Close files, remove temporary file.

I/O Specification

All files layouts input and output

Input file format:

FHEAD (start of file)

THEAD (Invoice level info)

TVATS (Summary level –cost at a VAT rate)

TXREF (Summary level – ASN and Order numbers)

TDETL (items within an invoice)

TSHIP (ASN and carton (optional) numbers within an invoice)

Note: all TSHIP records for a single item must follow immediately after the TDETL record.

TTAIL (marks end of invoice)

FTAIL (marks end of file)

An input file can contain multiple invoices.

Reject file will have an identical format. If no records are rejected, it will consist of only the FHEAD and FTAIL lines.

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

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	UPINV	Identifies which translation Gentran uses
	Current date	Char(14)		File date in YYYYMMDDHH24MISS format
THEAD	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction type flag	Char(1)		I=Invoice C=Credit note N= non-merchandise invoice
	Supplier invoice	Char (30)		Supplier's identifier for this invoice

	number			
	Cross-reference invoice number	Number(10)		Invoice that a credit note is for (blank for an invoice)
	Cross-reference RTV number	Number(6)		RTV order that a credit note is for (blank for an invoice)
	Partner Type	Char(6)		BK=Bank AG=Agent FF=Freight Forwarder IM=Importer BR=Broker FA=Factory AP=Applicant CO=Consolidator CN=Consignee S1=Supplier hierarchy level 1 (e.g. manufacturer) S2=Supplier hierarchy level 2 (e.g. distributor) S3=Supplier hierarchy level 3 (e.g. wholesaler)
	Partner ID	Char(10)		Credit note or non-merchandise invoice partner.
	Supplier	Number(10)		Invoice supplier
	Supplier invoice date	Char(14)		Inv_head.inv_date YYYYMMDDHH24MISS Date invoice was issued by supplier
	Terms	Char(15)		Inv_head.terms If not given in file, take from supplier terms
	Terms discount percentage	Number(12)		Discount percent if invoice paid on time 4 implied decimal places (optional—if not in file, take from terms table)
	Discount apply indicator	Char(1)		‘Y’—discount HAS been applied to total_merch_cost ‘N’—it has not
	Discount apply non merch ind	Char(1)		‘Y’—discount HAS been applied ‘N’—it has not
	Payment method	Char(6)		Inv_head.payment_method LC=letter of credit OA= Open Account WT= wire transfer

				(other values possible)
	Currency code	Char(3)		Currency code of invoice currency
	Exchange rate	Number(20)		From the invoice currency to the primary currency. 10 implied decimal places. Optional.
	Total merchandise cost	Number(20)		Total invoice cost, including all items on invoice (4 implied decimal places) (NOT including tax) VAT EXCLUSIVE
	Total quantity	Number(12)		Total quantity of items on invoice (4 implied decimal places)
	Freight terms	Char(30)		Freight terms associated w/invoice
	Sales tax	Number(20)		Sales tax on invoice (4 implied decimal places) Optional
	Sales tax VAT code	char(6)		VAT code for sales tax (optional)
	Freight charges	Number(20)		Freight charges for invoice (4 implied decimal places) Optional
	Freight charge VAT code	char(6)		VAT code for freight charges (optional)
	Miscellaneous charge	Number(20)		4 implied decimal places Optional
	Misc. charge VAT code	char(6)		VAT code for miscellaneous charge—optional
	Address key	Number(6)		Address key for the supplier or partner. (optional)
	Paid_ind	Char(1)		Paid index for the invoice. 'Y' or 'N'. (optional)

TVATS	File record descriptor	Char(5)	TVATS	Marks costs at VAT rate line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	VAT code	Char(6)		VAT code that applies to cost
	Cost at this VAT code	Number(20)		4 implied decimal places
TXREF	File record descriptor	Char(5)	TXREF	Marks cross-reference info line
	Line id	Char(10)		Sequential file line number
	Transaction number (10)	Number(10)		Sequential transaction number
	ASN number	Char(30)		Suppliers shipment number
	Order Number	Number(8)		RMS Order Number
TDETL	File record descriptor	Char(5)	TDETL	Marks item info line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Item	Char(25)		Item (optional but at least one of Item/Ref. Item must be present)
	Ref Item	Char(25)		Ref. Item (optional, see item)
	Original Invoice quantity	Number(12)		How many of this item on invoice (4 implied decimal places)
	Original Unit cost	Number(20)		Unit cost of item as reported by invoice 4 implied decimal places
	Original VAT rate	Number(20)		VAT rate for this item (10 implied decimal places)
TSHIP	File record descriptor	Char(5)	TSHIP	Marks ASN/carton info line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	ASN number	Char(30)		Vendor ASN number used to identify shipment
	Carton number	Char(20)		Carton ID if exists
TTAIL	File record descriptor	Char(5)	TTAIL	Marks end of transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction	Number(6)		Total number of TXREF,

	lines			TDETL, TVATS, and TSHIP lines in transaction
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Technical Issues

The VAT rate column on `invc_detail` may be changed to a VAT code column, at which point this program would need to be modified to read in a VAT code rather than a VAT rate.

EXTRA INFO

- use linked list to hold lines in transaction (so can write whole transaction to reject file at once)
- remember this is file-based—use `restart_file_init`, `restart_file_commit` (and `restart_file_write` for the reject file)
- Similar programs to look at: `ediupasn.pc`, `rcvupld.pc`, `ediupack.pc`

General structure of process function:

```

    Inside a while loop:
        Set a savepoint
        Read in a line (should be transaction header line. If it's a FTAIL line, break. If neither
header not FTAIL, set error flag)
        Validate transaction header line; set flag if fails validation
        Read in a line. If it's transaction tail line (TTAIL) break; else validate line and process
        If error occurred, rollback to savepoint and write transaction to reject file
        Call restart commit logic
    End outer (invoice) loop
    Call process_FTAIL to handle last line of file

```

Forecast sales rebuild [fcstrbld]

Retek Demand Forecasting Interface

Module Affected

Fcstrbld.pc

Design Overview

Fcstrbld.pc is designed to roll-up new or updated forecasted unit sales data from the item_forecast table. This data will be summarized into the subclass, class and department level sales forecast tables.

Stored Procedures / Shared Modules (Maintainability)

N/A

Input/Output Specifications

N/A

Program Flow

Process

- Populate_subclass
- Populate_class
- Populate_dept
- Delete_rec

Function Level Description

Process() : select the domain_id and rowid from forecast_rebuild. Pass the domain_id into populate_subclass(), populate_class(), and populate_dept() (in that order). Pass the rowid into delete_rec(). Perform standard commit logic.

Populate_subclass(domain_id): Take all the data from the same domain_id from the item_forecast table and roll it up to the subclass_sales_forecast table. Sum forecast sales for items in the same subclass with the same eow_date and location. Select from the item_master, item_forecast, and store tables.

Populate_class(domain_id): Take all the data from the same domain_id from the subclass_sales_forecast table and roll it up to the class_sales_forecast table. Sum forecast sales for all subclasses in a class.

Populate_dept(domain_id): Take all the data from the same domain_id from the class_sales_forecast table and roll it up to the dept_sales_forecast table. Sum forecast sales for all classes in a department.

Delete_rec(rowid): Delete the domain_id from forecast_rebuild once all the data for that domain_id is rolled. This way, if the program crashes it will restart at the last domain_id that wasn't rolled up completely.

Input Specifications

N/A

Output Specifications

N/A

Scheduling Considerations

Cycle = Weekly

Post processing to truncate forecast_rebuild table

Locking Strategy

Restart/Recovery

The logical unit of work is dept, class. The commit (update subclass_sales_hist table) should occur every 10,000 records.

Performance Considerations

Security Considerations

Unit Test Considerations

Forecasted unit sales upload [fcsilupld]

Design Overview

The Forecasted Unit Sales Upload (fcsilupld) is designed to load forecasted unit sales/issues data at the item location week level. The data will be loaded into the ITEM_FORECAST_XX table (where XX is the domain) by writing the data to a text file that will be loaded into the table using SQL loader.

Each forecasted sales/issues record will be validated before it is written to the text file. The validation performed will be as follows:

- Validate that the date of the record is greater than or equal to the current end of week (EOW) date and that it is a valid EOW date.
- No validation is made against the items or locations themselves. It is assumed that the items and locations (and their relationships) in the input file are valid in the RMS.

If the record passes validation, it will be written to a SQL loader text file.

Since one or more files may be present due to multiple domains, these files will be run in separate threads determined by the domain ID.

Scheduling Constraints

Processing Cycle: Weekly

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Domain ID

Restart Recovery

The logical unit of work is item and store records. The commit should occur every 10,000 records.

A driving cursor will not exist since this is a file-based processing module.

Program Flow

N/A

Shared Modules

C454_TO_CAL and CAL_TO_454 are called to verify that the calendar date passed in the input file into an end of week date.

Function Level Description

Threading set-up:

Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as

the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

The batch program should perform the following steps:

- Extract login ID, password from input arguments.
- Extract thread value from input file name. The input file will be fcslupld.*nn* where *nn* will equal the domain (i.e. the thread value).
- Read first record into local variables.
- The procedures CAL_TO_454 and C454_TO_CAL should be used to verify that the date passed to the program is a valid EOW date. If not, reject the record and write error to sqllog, date must be a valid end of week date.
- Validate that the end of week date is greater than or equal to the current EOW date. If not, reject the record and write error to sqllog, date must be greater than or equal to the current eow date.
- Write each record to the sql loader file (fcsales.dat.*nn*). Files will be comma delimited.
- After all the records have been processed, write the domain to the forecast_rebuild table (if it already exists, continue). The Forecast Rebuild program will run against this table to determine which domains have to be rebuilt.

I/O Specification

Input file fcslupld.*nn* (where *nn* is the domain ID)

Item	Numeric	25	
Location	Numeric	10	
Date	Date	8	(yyyymmdd)
Sales	Numeric	12	(12,4 – decimal implied)
Std. Dev.	Numeric	12	(12,4 – decimal implied)

Example:

```
123456789012345678901234567890123456789012345678901234567890
10000000000000000000000000391000000031200002270000000245997000000120000
```

Output file fcsales.dat.*nn* (comma delimited, where *nn* is the domain ID)

Item	Numeric	25	
Location	Numeric	10	
Eow date	Date	8	(yyyymmdd)
Forecast Sales	Numeric	12,4	99999999.9999
Std. Dev.	Numeric	12,4	99999999.9999
Domain	Numeric	2	

Example:

```
1000000000000000000000000039,1000000031,20000227,24.5997,12.0000,1
```

Technical Issues

N/A

Forecast by subclass upload [fcslupld_sbc]

Design Overview

The module receives a flat file coming from the PAS system and creates a comma delimited flat file to be SQL*LOADED, using the Perl script sbc_upld, to table subclass_sales_forecast.

The source flat file is sbc_fcslupld.x, with *x* being the partition or thread number.

The output flat file, sbc_fcsales.dat.x is created, with *x* being the partition or thread number.

The module will access information from the tables in the following matrix.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
CALENDAR	No	Yes	No	No	No

The structure of the output flat file is as follows:

```

eow_date,
dept,
class,
subclass,
location,
sales,
thread_val (or domain)

```

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)
PHASE AD-HOC (weekly)

Scheduling: After completion of PAS process

Pre-Processing: N/A

Post-Processing:

Threading Scheme: N/A (single threaded)

Restart Recovery:

Restart/recovery is based on the values stored in restart_bookmark from the last commit prior to failure. The values are for the last record.

Program Flow

N/A

Shared Modules

N/A

Function Level Description**init()**

- Opens input file
- Declares and opens output files
- Initializes restart/recovery
- Fetch system variables
- Calls get_cur_eow()
- Calls build_calendar()

file_process()

- Allocates memory for structure input variables.
- Calls read_records()
- Calls validate_records()
- Calls process_records()
- Calls insert_rebuild()
- Loop through array to process records until finished.

final()

- Closes input and output files
- Closes restart/recovery

Get_cur_eow()

- Gets the end of week date using stored procedures.

build_calendar()

- Creates an array of end of week dates.

read_records()

- Fetches records containing the following values:
dept,
class,
subclass,
domain_id,
loc,
eow_date,
forecast_sales,
thread_val (or domain id)

validate_records()

- Verifies end of week date in input file is an “end of week” date.

process_records()

- Writes values obtained from the input file to the output file.

insert_rebuild()

- Writes the domain id for records to forecast_rebuild table.

I/O Specification

N/A

Technical Issues

N/A

Forecasting/replenishment [fcstprg]

Design Overview

The forecast sales information at the item and hierarchy level is no longer stored on history tables. In order to speed the performance of the data loads into RMS and subsequent hierarchy forecast builds, the tables can be built either as partitioned views or as Oracle 8 partitioned tables. If the data objects are to be partitioned, they should be partitioned by domain.

When new forecasts are created and loaded into RMS from RDF, the entire domain will be re-built and re-loaded. For this reason, a purge program will be used to truncate the partitions, truncate the tables, or delete the data by domain so that they can be re-loaded by domain. This module will take the domain value that is being rebuilt and will perform the purge using dynamic sql.

This program must be run as either the RMS schema owner, or be run by a user that has been granted the following system privileges:

- 'drop any table'
- 'alter any table'

Scheduling Constraints

Pre/Post Logic Description:

Processing Cycle: before fcslupld.

Scheduling Diagram:

Pre-Processing: Disable indexes, but only if partitioned tables.

Post-Processing: Rebuild indexes.

Threading Scheme: threaded by passed parameter.

Restart Recovery

Logical Unit of Work (recommended Commit check points)
Driving Cursor

N/A

No restart/recovery libraries should be included at all.

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

Function Level Description

All database interactions required and error handling considerations

Main function()

This function will require one additional input parameter. The domain value will be passed into the function along with the username/password@database string.

Process()

The truncate_table function should be called 4 times, each time passing the table name to be truncated, the table_owner and a value to indicate whether or not the table is partitioned. The partition value is determined using the partition_check function. A character string should be locally declared to hold the table names. The four tables will be named:

```

item_forecast_X
dept_sales_forecast_X
class_sales_forecast_X
subclass_sales_forecast_X

```

where X is the domain value parameter.

Truncate_table():

This function will use the partition indicator to create a truncate table command as a character string and will use dynamic sql to truncate the table. The truncate table command should indicate which table is to be truncated and the table owner value. This will also determine what type of statement is necessary to remove the data.

Partition_check()

This function uses the table_name and the table_owner determined from the synonym_trace function to set an indicator that is passed back to process().

I/O Specification

All files layouts input and output

N/A

Volumetrics (table and flow volumes and characteristics)

Technical Issues

This program must be run as either the RMS schema owner, or be run by a user that has been granted the following system privileges:

- 'drop any table'
- 'alter any table'

Forecast item sales, issues download [fdaydnld]

Functional Area

RDF Interfaces

Module Affected

fdaydnld.pc

Design Overview

This module will query the store sales and warehouse issues histories for forecastable items as frequently as once a day. Four flat files for use in RDF modules will be created based on this data: one each for regular, promotion and clearance sales, and a single file for warehouse issues.

Stored Procedures / Shared Modules (Maintainability)

NA

Input Specifications

'Table-To-Table'

This program contains three distinct driving cursors, one each at department, class, and subclass domain levels for warehouse issues and store sales. Only one cursor (based on the system_options.domain_level value) will actually be used in a given instance of the program.

Store Sales Driving Cursors:

I the domain_type = 'D', then the driving cursor is as follows:

```
SELECT tdh.store,
       tdh.wh,
       tdh.item,
       tdh.tran_date,
       SUM(tdh.units),
       tdh.sales_type,
       tdh.tran_code
FROM   tran_data_history tdh,
       item_master im,
       item_loc_soh ils,
       domain_dept dd
WHERE  im.forecast_ind = 'Y'
       AND ((tdh.wh = -1 and tdh.tran_code = 1) /* Net Sales tran_code */
           or (tdh.store = -1 and tdh.tran_code = 32)) /*Transfers Out tran_code */
       AND (tdh.store      = ils.loc
           OR tdh.wh       = ils.loc)
       AND tdh.item       = im.item
       AND tdh.item       = ils.item
       AND ((tdh.tran_date > NVL(ils.last_hist_export_date, tdh.tran_date - 1)
           AND dd.load_sales_ind = 'N')
           OR (dd.load_sales_ind = 'Y'))
       AND im.dept        = dd.dept
       AND dd.domain_id   = TO_NUMBER(:ps_restart_thread_val)
       AND (tdh.item > NVL(:ps_restart_item,-999) OR
           (tdh.item = :ps_restart_item AND
            ils.loc > :ps_restart_location))
GROUP BY tdh.store,
         tdh.wh,
         tdh.item,
         tdh.tran_date,
```



```

        tdh.sales_type,
        tdh.tran_code
ORDER BY 3, 2, 1; /* item, wh, store */

```

if the domain_type = 'C', then the driving cursor is as follows:

```

SELECT tdh.store,
       tdh.wh,
       tdh.item,
       tdh.tran_date,
       SUM(tdh.units),
       tdh.sales_type,
       tdh.tran_code
FROM   tran_data_history tdh,
       item_master im,
       item_loc_soh ils,
       domain_class dc
WHERE  im.forecast_ind = 'Y'
      AND ((tdh.wh = -1 and tdh.tran_code = 1) /* Net Sales tran_code */
          or (tdh.store = -1 and tdh.tran_code = 32)) /*Transfers Out tran_code */
      AND (tdh.store = ils.loc
          OR tdh.wh = ils.loc)
      AND tdh.item = im.item
      AND tdh.item = ils.item
      AND ((tdh.tran_date > NVL(ils.last_hist_export_date, tdh.tran_date - 1)
          AND dc.load_sales_ind = 'N')
          OR (dc.load_sales_ind = 'Y'))
      AND im.dept = dc.dept
      AND im.class = dc.class
      AND dc.domain_id = TO_NUMBER(:ps_restart_thread_val)
      AND (tdh.item > NVL(:ps_restart_item,-999) OR
          (tdh.item = :ps_restart_item AND
           ils.loc > :ps_restart_location))
GROUP BY tdh.store,
         tdh.wh,
         tdh.item,
         tdh.tran_date,
         tdh.sales_type,
         tdh.tran_code
ORDER BY 3, 2, 1; /* item, wh, store */

```

if the domain_type = 'S', the driving cursor is as follows:

```

SELECT tdh.store,
       tdh.wh,
       tdh.item,
       tdh.tran_date,
       SUM(tdh.units),
       tdh.sales_type,
       tdh.tran_code
FROM   tran_data_history tdh,
       item_master im,
       item_loc_soh ils,
       domain_subclass ds
WHERE  im.forecast_ind = 'Y'
      AND ((tdh.wh = -1 and tdh.tran_code = 1) /* Net Sales tran_code */
          or (tdh.store = -1 and tdh.tran_code = 32)) /*Transfers Out tran_code */
      AND (tdh.store = ils.loc
          OR tdh.wh = ils.loc)
      AND tdh.item = im.item
      AND tdh.item = ils.item
      AND ((tdh.tran_date > NVL(ils.last_hist_export_date, tdh.tran_date - 1)
          AND ds.load_sales_ind = 'N')
          OR (ds.load_sales_ind = 'Y'))
      AND im.dept = ds.dept
      AND im.class = ds.class
      AND im.subclass = ds.subclass
      AND ds.domain_id = TO_NUMBER(:ps_restart_thread_val)
      AND (tdh.item > NVL(:ps_restart_item,-999) OR
          (tdh.item = :ps_restart_item AND

```

```

        ils.loc      > :ps_restart_location))
GROUP BY tdh.store,
        tdh.wh,
        tdh.item,
        tdh.tran_date,
        tdh.sales_type,
        tdh.tran_code
ORDER BY 3, 2, 1; /* item, wh, store */

```

Output Specifications

Output Files

The four output files will be named as follows:

Regular sales: rdsal nn .dat
Promotion sales: pdsal nn .dat
Clearance sales: cdsal nn .dat
Warehouse issues: dlyiss nn .dat

where nn is a two-digit number corresponding to the domain_id from which the data was derived, e.g. the regular sales output file from data in domain 13 would be named rdsal13.dat, the warehouse issue output file from data in domain 8 would be named dlyiss8.dat, and so on.

Output File Format:

Record Name	Field Name	Field Type	Default Value	Description
	Location	Char(20)		Store or Warehouse ID. Left-justified.
	Item	Char(25)		Item. Left-justified.
	Date	Char(8)		Transaction date. Left-justified. ('YYYYMMDD')
	Quantity	Char(13)		Sales/Issues. Contains up to four decimal places. If value is fractional, the quantity will be printed with decimal character '.' included, e.g. "1023.25". Right-justified.

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 files. The files should be named as described in Output Specifications. init() will also call a cursor to get the vdate from the period table and the domain_level from system_options.

format_buffer():

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

init_arrays():

Allocates the necessary memory (sized to the `commit_max_counter`) to the arrays used in processing.

process():

This function makes a call to `format_buffer()` and `init_arrays()`, `process_sales_issues()`, and `free_arrays()`.

process_sales_issues():

This function contains the three sales/issues driving cursors. Use the `domain_level` value fetched in `init()` to determine which cursor to use. Open the appropriate driving cursor. In a while loop, perform an array fetch. If the cursor is exhausted, set a flag to indicate this. Use a for loop to loop through each of the records in the array. For each, call `write_daily_sales()`, passing the record's `sales_type`, `store`, `item`, `tran_date` and `units` as parameters. Also, call `write_daily_issues` passing `store`, `item`, `tran_date` and `units` as parameters. After each set of dates is processed by the for loop, make a call to `retek_force_commit()`. If the exhausted cursor flag is set, break out of the while loop.

write_daily_sales():

This function takes five strings as parameters: `sales_type`, `store`, `item`, `tran_date`, and `units`. If the `sales_type` is equal to "R" (regular sale), then write the data to the `rdsalnn.dat` file. If equal to "P" (promotion sale), write to the `pdsalnn.dat` file. If equal to "C" (clearance sale), write to the `cdsalnn.dat` file. All output will be written to files using the string formatted in `format_buffer()`.

write_daily_issues():

This function takes four strings as parameters: `wh`, `item`, `tran_date`, and `units`. Write this data to the `dlyissnn.dat` file using the string formatted in `format_buffer()`.

final():

Take care of file clean up, freeing of memory, and complete the restart recovery process by calling `retek_close()`.

Scheduling Considerations

This program should run weekly in phase 4.

It must be run before the weekly sales/issues download programs: `fsadnld.pc`.

It may also run daily in phase 4, if daily sales/issues download are necessary.

Locking Strategy

Restart/Recovery

This program will use restart recovery. The logical unit of work is each unique Item/Location combination.

Performance Considerations

Security Considerations

Design Assumptions

Outstanding Design Issues

References

Daily sales forecast upload [fdayupld]

Functional Area

Enterprise interfaces – interface daily forecasts between RMS and RDF

Module Affected

Fdayupld - Perl script to load daily forecast data into daily_item_forecast_nn.

Design Overview

The forecast sales information tables have been built either as partitioned views or as Oracle 8 partitioned tables. The data objects are all partitioned by domain. When new forecasts are created and loaded into RMS from RDF, the entire domain will be re-built and re-loaded.

This perl script will accept a daily item forecast input file from RDF and load it into the correct RMS daily_item_forecast partition using sqlloader. The correct partition will be determined by using the input file's extension. The script will create a control file to truncate and load the data from the input file into the domain's partition.

Output Specifications

'File-To-Table'

Table/View	Select	Insert (sqlload)	Truncate Table	Truncate Partition
DAILY_ITEM_FORECAST		X		X

Function Level Description

The input file will be in this format:

Input file filename.nn (where nn is the domain ID)

Item	numeric	25	
Location	numeric	10	
Date	date	8	(yyyymmdd)
Sales	numeric	12	(12,4 - 4 implied decimal places)
Std. Dev.	numeric	12	(12,4 - 4 implied decimal places)

Example record:

```
10000266
100000000220000116000000009691000000108542
```

```
item:10000266
loc:1000000002
date:20000116
sales:000000009691 (or .9691 after decimals are put back in)
```

```
std dev:000000108542 (or 10.8542 after decimals are put
back in)
```

The output control file should have the below form where *NN* is the domain of the input file.

```
LOAD DATA
TRUNCATE INTO TABLE daily_item_forecast_ NN
(
ITEM          position(1:25)  INTEGER EXTERNAL,
LOC           position(26:35) INTEGER EXTERNAL,
DATA_DATE     position(36:43) DATE "YYYYMMDD",
FORECAST_SALES position(44:55) DECIMAL EXTERNAL
":FORECAST_SALES/10000",
FORECAST_STD_DEV position(56:67) DECIMAL EXTERNAL
":FORECAST_STD_DEV/10000",
DOMAIN_ID     CONSTANT      NN
)
```

The input file should be used to get the domain_id. Once the domain is retrieved, the control file should be created. Finally, the data file should be loaded into the database using the generated control file and the userid/password@database from the command line.

The script will accept two parameters, a connect string and the file name of the daily forecast input file:

```
fdayupld userid/passwd@database datafile
```

Assumption:

The files sent from RDF will contain an entire domain. Domains cannot be split between multiple files or data will be lost.

Scheduling Considerations

Fdayupld should be run in phase 2. It has no pre or post dependencies except that it should not be run during the replenishment run (phase 3).

Financials general ledger download 1 [fifglnd1]

Design Overview

This process will extract detailed stock ledger information for certain transaction types on a daily basis in order to bridge the information to an interfaced financial application. The detailed information will retain certain reference information, such as stock adjustment reason codes, which are lost during the normal daily roll-up process (saldly.pc).

This process reads from the `if_tran_data` table, instead of the actual `tran_data` table itself. Hence, it requires the 'PRE-SALDLY' and 'PRE-FIFGLDN1' run of `prepost.pc` to have occurred prior to its initiation. It can run in parallel to `fifglnd2.pc`.

New Codes Table Information

Concerning GL data, there are two factors that need to be addressed:

1. Some transaction types/amount types need to be bridged at a low level to contain detailed reference information, while others do not.
2. There are only certain transaction types/amount types that need to be bridged for a given client. For instance, net sales at retail may be needed for one client, while net sales cost by be needed by another.

This process should allow the flexibility to bridge the required information without coding efforts. Since users can manipulate the codes detail table through a dialogue, this would be a good tool to allow this maintenance.

The new code type that will be created for this process is the General Ledger – Single Transaction 'GLST' type. This will include transaction types/amount types to go through the sku-level detailed processing if the feed to GL is to be daily. The code for this will be a 3-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the sku-detail level.

There will be two other codes types in the total GL interface: General Ledger – Rolled Transaction 'GLRT' and General Ledger – Monthly Transaction 'GLMT'. These are part of the FIFGLDN2 and FIFGLDN3 designs.

Note that there are several risks in the maintenance of these codes types:

1. The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This will result in not sending the code 12 GL transactions.
2. '12R' could be entered in more than one codes type. This will result in duplicate processing, although at different levels of roll-up. There is no cross-checking to verify that a code entered in one code type is not in the other.

Daily-level GL Feed

New processes:

The new program `fifglnd1.pc` will read the `if_tran_data` table for each transaction type/amount type in the codes table, and post it to a Retek general ledger staging table (`fif_gl_data`) at the sku

detail level, complete with reference fields. These reference fields will be bridged to the financial application for segregation of stock adjustment codes, mapping transfers to transfer numbers, etc., depending on the client need.

The new program fifgldn1.pc can run in parallel to the saldly process, but must run after the if_tran_data load process, as it will process rows on if_tran_data.

The local currency will be retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Daily, possibly in parallel to fifgldn2.pc, after PREPOST(PRE-SALDLY) and PREPOST(PRE-FIFGLDN1).

Scheduling Diagram:

Pre-Processing: Prepost for PRE-SALDLY. Prepost for PRE-FIFGLDN1 (this creates indexes)

Post-Processing

Threading Scheme: Based on store/wh.

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

This program will be fully restartable. The LUW will be based upon dept, class, subclass.

```

Select distinct
    Td.sku,
    Td.Dept,
    Td.Class,
    Td.Subclass,
    Td.Store,
    Td.Wh,
    Td.Tran_date,
    Td.Tran_code,
    Td.Adj_code,
    Td.Total_cost,
    Td.Total_retail,
    Td.Ref_no_1,
    Td.Ref_no_2,
    St.currency_code,
    Wh.currency_code,
    cd.code
from    if_tran_data td,
        code_detail cd,
        store st,
        wh
        v_restart_store_wh
where (cd.code_type = 'GLST')
       and (substr(cd.code, 1, 2) = td.tran_code)

```



```

and (td.store = st.store (+)
    and td.wh = wh.wh (+) )
and v.driver_name = :os_restart_driver_name
and v.driver_value = :td.store
and v.num_threads = :oi_restart_num_threads
and v.thread_val = :oi_restart_thread_val
and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
    (td.dept = :ora_restart_dept AND
        (td.class > :ora_restart_class OR
            (td.class = :ora_restart_class AND
                (td_subclass > :ora_restart_subclass)))))
order by dept,
        class,
        subclass;

```

Program Flow

The high-level processing for this program is as follows:

1. Read rows
2. Obtain its cost component from code_detail (substr(code, 3,1)). If it is 'C' for Cost, insert a row to fif_gl_data for the cost amount. If it is 'R', insert a row for the retail amount. If it is 'B' for both, insert one row for each.
3. If the processing/insert was successful, continue, else write a row to the if_errors interface error table.

init() -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the FIF_GL_DATA table (dept, class, subclass, location, loc_type, tran_date, tran_code, adj_code, amount, cost_retail_flag, ref_no_1, ref_no_2, sku, currency_code in the structure).
- Normal array sizing should occur [size_arrays()]

process() -

- open the driving cursor
- WHILE(1)
 - fetch g_l_restart_max_counter records from driving cursor into the array
 - if NO_DATA_FOUND, this means there were fewer than g_l_restart_max_counter records fetched or none were fetched. Set a flag to eventually break from the loop.
 - Calculate the num_records_to_process in the array by subtracting previously processed records from NUM_RECORDS_PROCESSED (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
 - If restart_string is NULL (happens on a fresh start only)
- Populate it with the first LUW information
 - For 0 to < num_records_to_process, loop
- If LUW has NOT changed
 - If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - If the 3rd byte of code is 'R' or 'B', write_array() for retail
- ELSE
 - Write_to_table() to fif_gl_data
 - Call restart_commit()
 - Set restart_string to new LUW
 - If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - If the 3rd byte of code is 'R' or 'B', write_array() for retail
- Add num_records_to_process to the previously processed records
 - If loop breaking flag is set
- break
- END WHILE

- Call `write_to_table()` -- this will insert the final LUW from the `insert_array`

final() -

- Close off restart recovery processing

size_arrays() -

- Allocates the memory for the two array structures.

write_array(array_index, cost_retail_flag) -

- write cost/retail depending on flag passed in
- increment `array_size` variable
- if `array_size = g_l_restart_max` then call `write_to_table()`

write_to_table() -

- For `insert_array_size` records, an array insert into `fif_gl_data`
- Set `insert_array_size` counter = 0.
- If there are insertion errors, the program should return fatally.

Common Functions

Listing of all externally referenced functions and Stored procedures and description of usage

None.

Functional Level Description

All database interactions required and error handling considerations

All required stock ledger data must be processed to the GL interface table within Retek. Errors will be handled in the normal batch method.

I/O Specification

All files' layouts input and output

None.

Technical Issues

None.

Financials general ledger download 2 [fifglndn2]

Design Overview

This process will summarize stock ledger data from the transaction staging table `if_tran_data`, based on the level of information required, and post it to a staging table within Retek for transfer to a financial application's general ledger.

This process reads from the `if_tran_data` table, instead of the actual `tran_data` table itself. Hence, it requires the 'PRE-SALDLY' and 'PRE-FIFGLDN1' run of `prepost.pc` to have occurred prior to its initiation. It can run in parallel to `fifglnd1.pc`.

New Codes Table Information

The new code type that will be created for this process is the General Ledger – Rolled Transaction 'GLRT' type. This will include transaction types/amount types to go through the roll-up processing if the feed to GL is to be daily. The code for this will be a 3-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the sku-detail level.

Note that there are several risks in the maintenance of these codes types:

3. The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This will result in not sending the code 12 GL transactions.
4. '12R' could be entered in more than one codes type. This will result in duplicate processing, although at different levels of roll-up. There is no cross-checking to verify that a code entered in one code type is not in the other.

Daily-level GL Feed

New processes:

The new program `fifglndn2.pc` will read and summarize the `if_tran_data` table for all transaction types/amount types in the codes table, and post it to a Retek general ledger staging table (`fif_gl_data`) at the department, class, or subclass level.

The new program `fifglndn2.pc` can run in parallel to the `saldly` process, as well as the `fifglndn1.pc` process, but must run after the `if_tran_data` load process, as it will process rows on `if_tran_data`.

The local currency will be retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Daily, possibly in parallel to `saldly`, after `PREPOST`('pre-saldly' and 'pre-fifglndn1').

Scheduling Diagram:

Pre-Processing:

Post-Processing

Threading Scheme: store/wh

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

This program will be fully restartable. It will use three different driving cursors, based on the input parameter (DEPT, CLASS, SUBCLASS):

If 'DEPT' is the input parameter, the following cursor will be used:

```

Select  td.Dept,
        -1,
        -1,
        td.Store,
        td.Wh,
        td.tran_date,
        Td.Tran_code,
        sum(Td.Total_cost),
        sum(Td.Total_retail),
        cd.code,
        ';' || to_char(td.dept)
from    if_tran_data td,
        code_detail cd,
        v_restart_store_wh v
where   (cd.code_type = 'GLRT')
        and (substr(cd.code,1,2)=td.tran_code)
        and v.driver_name = :os_restart_driver_name
        and v.driver_value = :td.store
        and v.num_threads = :oi_restart_num_threads
        and v.thread_val = :oi_restart_thread_val
        and td.dept > NVL(:ora_restart_dept, td.dept-1)
group by td.store,
        td.wh,
        td.dept,
        td.tran_code,
        td.tran_date,
        cd.code
order by td.dept

```

If 'CLASS' is the input parameter, the following cursor will be used:

```

Select  td.Dept,
        Td.class,
        -1,
        td.Store,
        td.Wh,
        td.tran_date,
        Td.Tran_code,
        sum(Td.Total_cost),
        sum(Td.Total_retail),
        cd.code,
        ';' || to_char(td.dept) || ';' || to_char(td.class)

```

```

from if_tran_data td,
    code_detail cd,
    v_restart_store_wh v
where (cd.code_type = 'GLRT')
    and (substr(cd.code,1,2)=td.tran_code)
    and v.driver_name = :os_restart_driver_name
    and v.driver_value = :td.store
    and v.num_threads = :oi_restart_num_threads
    and v.thread_val = :oi_restart_thread_val
    and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
        (td.dept = :ora_restart_dept AND
         (td.class > :ora_restart_class)))
group by td.store,
        td.wh,
        td.dept,
        td.class,
        td.tran_code,
        td.tran_date,
        cd.code
order by td.dept,
        td.class

```

If 'SUBCLASS' is the input parameter, the following cursor will be used:

```

Select td.Dept,
        Td.class,
        Td.subclass,
        td.Store,
        td.Wh,
        td.tran_date,
        Td.Tran_code,
        sum(Td.Total_cost),
        sum(Td.Total_retail),
        cd.code,
        '||to_char(td.dept)||';'||to_char(td.class)||';'||to_char(td.subclass)
from if_tran_data td,
    code_detail cd,
    v_restart_store_wh v
where (cd.code_type = 'GLRT')
    and (substr(cd.code,1,2)=td.tran_code)
    and v.driver_name = :os_restart_driver_name
    and v.driver_value = :td.store
    and v.num_threads = :oi_restart_num_threads
    and v.thread_val = :oi_restart_thread_val
    and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
        (td.dept = :ora_restart_dept AND
         (td.class > :ora_restart_class OR
          (td.class = :ora_restart_class AND
           (td.subclass > :ora_restart_subclass)))))
group by td.store,
        td.wh,
        td.dept,
        td.class,
        td.subclass,
        td.tran_code,
        td.tran_date,
        cd.code
order by td.dept,
        td.class,
        td.subclass

```

Program Flow

The high-level processing for this program is summarized in the following 3 steps:

1. Read the row
 2. Obtain its cost component from `code_detail (substr(code, 3,1))`. If it is 'C' for Cost, insert a row to `fif_gl_data` for the cost amount. If it is 'R', insert a row for the retail amount. If it is 'B' for both, insert one row for each.
 3. If the processing/insert was successful, continue, else write a row to the `if_errors` interface error table.
- This program will take the value of `system_options.gl_rollup`. There will be a value of 'D' for department, 'C' for class, or 'S' for subclass.
 - To facilitate the dept/class/subclass nature of this program, three integer global bind vars will be declared: `Dept_rollup`, `class_rollup`, and `sclass_rollup`. Depending on the `system_options` value described above, the appropriate variable will be set to 1 and the others to 0 (zero). If/then calls such as `"if (dept_rollup) { <statement> };"` will allow for easy readability of the code.
 - In addition to the insert array structure (with fields mirroring the `fif_gl_data` table structure), one structure (mirroring the **subclass** cursor field list plus a field for the LUW info retrieved from the cursor) should be declared. Depending on the level of rollup desired, the class or class & subclass fields will be populated with '-1', but this method is cleaner than having 3 structures to track.

init() -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the `FIF_GL_DATA` table (`dept`, `class`, `subclass`, `location`, `loc_type`, `tran_date`, `tran_code`, `adj_code`, `amount`, `cost_retail_flag`, `ref_no_1`, `ref_no_2`, `sku`, `currency_code` in the structure).
- Normal array sizing should occur [`size_arrays()`]
- Get value from `system_options` (as described above). If it is a department level rollup, initialize only the `ora_restart_dept` var. If it's class, initialize only `ora_restart_dept` and `ora_restart_class`, etc.

process() -

- open the appropriate driving cursor
- `WHILE(1)`
- fetch `g_l_restart_max_counter` records from appropriate driving cursor into the array
- if `NO_DATA_FOUND`, this means there were fewer than `g_l_restart_max_counter` records fetched or none were fetched. Set a flag to eventually break from the loop.
- Calculate the `num_records_to_process` in the array by subtracting previously processed records from `NUM_RECORDS_PROCESSED` (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
- If `restart_string` is `NULL` (happens on a fresh start only)
 - Populate it with the first LUW information
- For 0 to `< num_records_to_process`, loop
 - If LUW has NOT changed
 - If the 3rd byte of code is 'C' or 'B', `write_array()` for cost

- If the 3rd byte of code is 'R' or 'B', write_array() for retail
- ELSE
 - Write_to_table() to fif_gl_data
 - Call restart_commit()
 - Set restart_string to new LUW
 - If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - If the 3rd byte of code is 'R' or 'B', write_array() for retail
- Add num_records_to_process to the previously processed records
- If loop breaking flag is set
 - break
- END WHILE
- Call write_to_table() -- this will insert the final LUW from the insert_array

size_arrays() -

- Allocates the memory for the two array structures. (the fetch array and the insert array)

write_array(array_index, cost_retail_flag) -

- Fetch currency code from store or wh table.
- write cost/retail depending on flag passed in
- increment array_size variable
- if array_size = g_l_restart_max then call write_to_table()

write_to_table() -

- For insert_array_size records, an array insert into fif_gl_data. If -1's are in class and/or subclass, insert them as normal.
- Set insert_array_size counter = 0.
- If there are insertion errors, the program should return fatally.

final() -

- Close off restart recovery processing

Functional Level Description

All database interactions required and error handling considerations

All required stock ledger data must be processed to the GL interface table within Retek. Errors will be handled using the normal batch techniques.

I/O Specification

All files' layouts input and output

None.

Technical Issues

None.

Financials general ledger download 3 [fifgldn3]

Design Overview

This process will summarize stock ledger data from the monthly stock ledger table `month_data`, based on the level of information required, and post it to a staging table within Retek for transfer to a financial application's general ledger.

Since this process reads from the `month_data` table, it requires the sales monthly process SALMTH to have run. It can run any time after the close of the last day of the month.

New Codes Table Information

The new code type that will be created for this process is the General Ledger – Monthly Transaction 'GLMT' type. This will include transaction types/amount types to go through the roll-up processing if the feed to GL is to be monthly. The code for this will be a 3-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the department, class, or subclass level.

Since the monthly amounts are no longer maintained as transaction codes, this process will need to map certain amounts back to `tran_codes`. Some amounts are calculated from existing amount fields, and are not present on the `tran_data` level. These amounts will be given `tran_codes` specifically for bridging to a general ledger. The following list is the field mapping for the process:

Month_data field	Tran code	Cost/ Retail	Comment
OPN_STK_RETAIL	50	R	Monthly Calculated Figure
OPN_STK_COST	50	C	Monthly Calculated Figure
STOCK_ADJ_RETAIL	22	R	
STOCK_ADJ_COST	22	C	
PURCH_RETAIL	20	R	
PURCH_COST	20	C	
RTV_RETAIL	24	R	
RTV_COST	24	C	
FREIGHT_COST	26	C	
TSF_IN_RETAIL	30	R	
TSF_IN_COST	30	C	
TSF_OUT_RETAIL	32	R	
TSF_OUT_COST	32	C	
NET_SALES_RETAIL	01	R	
NET_SALES_RETAIL_EX_VAT	02	R	
NET_SALES_COST	01	C	
RETURNS_RETAIL	04	R	
RETURNS_COST	04	C	
MARKUP_RETAIL	11	R	
MARKUP_CAN_RETAIL	12	R	
CLEAR_MARKDOWN_RETAIL	16	R	

PERM MARKDOWN RETAIL	13	R	
PROM MARKDOWN RETAIL	15	R	
MARKDOWN CAN RETAIL	14	R	
SHRINKAGE_COST	51	C	Monthly Calculated Figure
SHRINKAGE RETAIL	51	R	Monthly Calculated Figure
EMPL DISC RETAIL	60	R	
WORKROOM AMT	80	R	
CASH DISC AMT	81	R	
CLS STK RETAIL	52	R	Monthly Calculated Figure
CLS STK COST	52	C	Monthly Calculated Figure
GROSS MARGIN AMT	53	R	Monthly Calculated Figure
COST VARIANCE AMT	70	C	
HTD GAFS RETAIL	54	R	Monthly Calculated Figure
HTD GAFS_COST	54	C	Monthly Calculated Figure
INTER_STOCKTAKE_SALES_A MT	55	C	Monthly Calculated Figure
INTER_STOCKTAKE_SHRINK_A MT	56	C	Monthly Calculated Figure
STOCKTAKE_MTD_SALES_AMT	57	C	Monthly Calculated Figure
STOCKTAKE_MTD_SHRINK_AM T	58	C	Monthly Calculated Figure
STOCKTAKE_BOOKSTK_RETAI L	59	R	Monthly Calculated Figure
STOCKTAKE_BOOKSTK_COST	59	C	Monthly Calculated Figure
STOCKTAKE_ACTSTK RETAIL	61	R	Monthly Calculated Figure
STOCKTAKE_ACTSTK_COST	61	C	Monthly Calculated Figure
RECLASS IN COST	34	C	
RECLASS IN RETAIL	34	R	
RECLASS OUT COST	36	C	
RECLASS OUT RETAIL	36	R	

The tran_codes of 50-59 and 61-69 will need to be reserved specifically for this interface.

Note that there are several risks in the maintenance of these codes types:

- The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This will result in not sending the code 12 GL transactions.
- '12R' could be entered in more than one codes type. This will result in duplicate processing, although at different levels of roll-up or different processing times (daily versus monthly). There is no cross-checking to verify that a code entered in one code type is not in the other.

Monthly-level GL Feed

New processes:

The new program fifgldn3.pc will read and summarize the month_data table for all transaction types/amount types in the codes table, and post it to a Retek general ledger staging table (fif_gl_data) at the department, class, or subclass level.


```

        '16R',sum(md.clear_markdown_retail),
        '13R',sum(md.perm_markdown_retail),
        '15R',sum(md.prom_markdown_retail),
        '14R',sum(md.markdown_can_retail),
        '51C',sum(md.shrinkage_cost),
        '51R',sum(md.shrinkage_retail),
        '60R',sum(md.empl_disc_retail),
        '80R',sum(md.workroom_amt),
        '81R',sum(md.cash_disc_amt),
        '52R',sum(md.cls_stk_retail),
        '52C',sum(md.cls_stk_cost),
        '53R',sum(md.gross_margin_amt),
        '70C',sum(md.cost_variance_amt),
        '54R',sum(md.htd_gafs_retail),
        '54C',sum(md.htd_gafs_cost),
        '55C',sum(md.inter_stocktake_sales_amt),
        '56C',sum(md.inter_stocktake_shrink_amt),
        '57C',sum(md.stocktake_mtd_sales_amt),
        '58C',sum(md.stocktake_mtd_shrink_amt),
        '59R',sum(md.stocktake_bookstk_retail),
        '59C',sum(md.stocktake_bookstk_cost),
        '61R',sum(md.stocktake_actstk_retail),
        '61C',sum(md.stocktake_actstk_cost),
        '34C',sum(md.reclass_in_cost),
        '34R',sum(md.reclass_in_retail),
        '36C',sum(md.reclass_out_cost),
        '36R',sum(md.reclass_out_retail),
        0),0),
    cd.code,
    ';' || to_char(md.dept)
from month_data md,
     v_restart_store_wh v,
     code_detail cd
where cd.code_type = 'GLMT'
     and md.half_no = :last_half_no
     and md.month_no = :last_month_no
     and md.currency_ind = 'L'
     and v.driver_name = :os_restart_driver_name
     and v.driver_value = md.store
     and v.num_threads = :oi_restart_num_threads
     and v.thread_val = :oi_restart_thread_val
     and md.dept > NVL(:ora_restart_dept, md.dept - 1)
group by md.store,
         md.wh,
         md.dept,
         cd.code
order by md.dept

```

Also, one for CLASS-level roll-up:

```

SELECT md.dept,
       md.class,
       -1,
       md.store,
       md.wh,
       NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),

```

```

'50C',sum(md.opn_stk_cost),
'22R',sum(md.stock_adj_retail),
'22C',sum(md.stock_adj_cost),
'20R',sum(md.purch_retail),
'20C',sum(md.purch_cost),
'24R',sum(md.rtv_retail),
'24C',sum(md.rtv_cost),
'26C',sum(md.freight_cost),
'30R',sum(md.tsf_in_retail),
'30C',sum(md.tsf_in_cost),
'32R',sum(md.tsf_out_retail),
'32C',sum(md.tsf_out_cost),
'01R',sum(md.net_sales_retail),
'02R',sum(md.net_sales_retail_ex_vat),
'01C',sum(md.net_sales_cost),
'04R',sum(md.returns_retail),
'04C',sum(md.returns_cost),
'11R',sum(md.markup_retail),
'12R',sum(md.markup_can_retail),
'16R',sum(md.clear_markdown_retail),
'13R',sum(md.perm_markdown_retail),
'15R',sum(md.prom_markdown_retail),
'14R',sum(md.markdown_can_retail),
'51C',sum(md.shrinkage_cost),
'51R',sum(md.shrinkage_retail),
'60R',sum(md.empl_disc_retail),
'80R',sum(md.workroom_amt),
'81R',sum(md.cash_disc_amt),
'52R',sum(md.cls_stk_retail),
'52C',sum(md.cls_stk_cost),
'53R',sum(md.gross_margin_amt),
'70C',sum(md.cost_variance_amt),
'54R',sum(md.htd_gafs_retail),
'54C',sum(md.htd_gafs_cost),
'55C',sum(md.inter_stocktake_sales_amt),
'56C',sum(md.inter_stocktake_shrink_amt),
'57C',sum(md.stocktake_mtd_sales_amt),
'58C',sum(md.stocktake_mtd_shrink_amt),
'59R',sum(md.stocktake_bookstk_retail),
'59C',sum(md.stocktake_bookstk_cost),
'61R',sum(md.stocktake_actstk_retail),
'61C',sum(md.stocktake_actstk_cost),
'34C',sum(md.reclass_in_cost),
'34R',sum(md.reclass_in_retail),
'36C',sum(md.reclass_out_cost),
'36R',sum(md.reclass_out_retail),
0),0),

cd.code,
';'||to_char(md.dept)||';'||to_char(md.class)
FROM month_data md,
v_restart_store_wh v,
code_detail cd
WHERE cd.code_type = 'GLMT'
AND md.half_no = :last_half_no
AND md.month_no = :last_month_no
AND md.currency_ind = 'L'
AND v.driver_name = :os_restart_driver_name

```

```

AND v.driver_value = md.store
AND v.num_threads  = :oi_restart_num_threads
AND v.thread_val   = :oi_restart_thread_val
AND (md.dept > NVL(:ora_restart_dept, md.dept - 1) OR
     (md.dept = :ora_restart_dept and
      (md.class > :ora_restart_class)))
GROUP BY md.store,
         md.wh,
         md.dept,
         md.class,
         cd.code
ORDER BY md.dept,
         md.class

```

Lastly, there will be one for SUBCLASS-level roll-up:

```

SELECT md.dept,
       md.class,
       md.subclass,
       md.store,
       md.wh,
       NVL(decode(cd.code, '50R', sum(md.opn_stk_retail),
                    '50C', sum(md.opn_stk_cost),
                    '22R', sum(md.stock_adj_retail),
                    '22C', sum(md.stock_adj_cost),
                    '20R', sum(md.purch_retail),
                    '20C', sum(md.purch_cost),
                    '24R', sum(md.rtv_retail),
                    '24C', sum(md.rtv_cost),
                    '26C', sum(md.freight_cost),
                    '30R', sum(md.tsf_in_retail),
                    '30C', sum(md.tsf_in_cost),
                    '32R', sum(md.tsf_out_retail),
                    '32C', sum(md.tsf_out_cost),
                    '01R', sum(md.net_sales_retail),
                    '02R', sum(md.net_sales_retail_ex_vat),
                    '01C', sum(md.net_sales_cost),
                    '04R', sum(md.returns_retail),
                    '04C', sum(md.returns_cost),
                    '11R', sum(md.markup_retail),
                    '12R', sum(md.markup_can_retail),
                    '16R', sum(md.clear_markdown_retail),
                    '13R', sum(md.perm_markdown_retail),
                    '15R', sum(md.prom_markdown_retail),
                    '14R', sum(md.markdown_can_retail),
                    '51C', sum(md.shrinkage_cost),
                    '51R', sum(md.shrinkage_retail),
                    '60R', sum(md.empl_disc_retail),
                    '80R', sum(md.workroom_amt),
                    '81R', sum(md.cash_disc_amt),
                    '52R', sum(md.cls_stk_retail),
                    '52C', sum(md.cls_stk_cost),
                    '53R', sum(md.gross_margin_amt),
                    '70C', sum(md.cost_variance_amt),
                    '54R', sum(md.htd_gafs_retail),
                    '54C', sum(md.htd_gafs_cost),

```

```

        '55C',sum(md.inter_stocktake_sales_amt),
        '56C',sum(md.inter_stocktake_shrink_amt),
        '57C',sum(md.stocktake_mtd_sales_amt),
        '58C',sum(md.stocktake_mtd_shrink_amt),
        '59R',sum(md.stocktake_bookstk_retail),
        '59C',sum(md.stocktake_bookstk_cost),
        '61R',sum(md.stocktake_actstk_retail),
        '61C',sum(md.stocktake_actstk_cost),
        '34C',sum(md.reclass_in_cost),
        '34R',sum(md.reclass_in_retail),
        '36C',sum(md.reclass_out_cost),
        '36R',sum(md.reclass_out_retail),
        0),0),
    cd.code,
    ';' || to_char(md.dept) || ';' || to_char(md.class) || ';' || to_char(md.subclass)
FROM month_data md,
     v_restart_store_wh v,
     code_detail cd
WHERE cd.code_type = 'GLMT'
      AND md.half_no = :last_half_no
      AND md.month_no = :last_month_no
      AND md.currency_ind = 'L'
      AND v.driver_name = :os_restart_driver_name
      AND v.driver_value = md.store
      AND v.num_threads = :oi_restart_num_threads
      AND v.thread_val = :oi_restart_thread_val
      AND (md.dept > NVL(:ora_restart_dept, md.dept - 1) OR
           (md.dept = :ora_restart_dept and
            (md.class > :ora_restart_class or
             (md.class = :ora_restart_class and
              (md.subclass > :ora_restart_subclass)))))
GROUP BY md.store,
         md.wh,
         md.dept,
         md.class,
         md.subclass,
         cd.code
ORDER BY md.dept,
         md.class,
         md.subclass

```

Program Flow

- This program will take the value of system_options.gl_rollup. There will be a value of 'D' for department, 'C' for class, or 'S' for subclass.
- To facilitate the dept/class/subclass nature of this program, three integer global bind vars will be declared: Dept_rollup, class_rollup, and subclass_rollup. Depending on the system_options value described above, the appropriate variable will be set to 1 and the others to 0 (zero). If/then calls such as "if (dept_rollup) { <statement> };" will allow for easy readability of the code.
- In addition to the insert array structure (with fields mirroring the fif_gl_data table structure), one structure (mirroring the **subclass** cursor field list) should be declared. Depending on the

level of rollup desired, the class or class & subclass fields will be populated with '-1', but this method is cleaner than having 3 structures to track.

- `Fif_gl_data.tran_date` will always be populated with `system_variables.last_eom_date`

init() -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the `FIF_GL_DATA` table (dept, class, subclass, location, loc_type, tran_date, tran_code, adj_code, amount, cost_retail_flag, ref_no_1, ref_no_2, sku, currency_code in the structure).
- Normal array sizing should occur [`size_arrays()`]
- Get value from `system_options` (as described above). If it is a department level rollup, initialize only the `ora_restart_dept` var. If it's class, initialize only `ora_restart_dept` and `ora_restart_class`, etc.
- Get values from `system_vars`: `last_eom_half_no`, `last_eom_month_no`, & `last_eom_date`

process() -

- open the appropriate driving cursor
- `WHILE(1)`
 - fetch `g_l_restart_max_counter` records from appropriate driving cursor into the array
 - if `NO_DATA_FOUND`, this means there were fewer than `g_l_restart_max_counter` records fetched or none were fetched. Set a flag to eventually break from the loop.
 - Calculate the `num_records_to_process` in the array by subtracting previously processed records from `NUM_RECORDS_PROCESSED` (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
 - If `restart_string` is `NULL` (happens on a fresh start only)
 - Populate it with the first LUW information
 - For 0 to < `num_records_to_process`, loop
 - If LUW has NOT changed
 - `fill_array()`
 - ELSE
 - `Write_array()` to `fif_gl_data`
 - Call `restart_commit()`
 - Set `restart_string` to new LUW
 - `fill_array()` for cost
 - Add `num_records_to_process` to the previously processed records
 - If loop breaking flag is set
 - break
- `END WHILE`
- Call `write_array()` -- this will insert the final LUW from the `insert_array`

fill_array(array_index, cost_retail_flag) -

- Fetch currency code from store or wh table.
- write cost/retail depending on flag passed in
- increment `array_size` variable
- if `array_size = g_l_restart_max` then call `resize_array()`

write_array() -

- For insert_array_size records, an array insert into fif_gl_data. If -1's are in class and/or subclass, insert them as normal.
- Set insert_array_size counter = 0.
- If there are insertion errors, the program should return fatally.

size_arrays() -

- Allocates the memory for the two array structures. (the fetch array and the insert array)

resize_array() -

- increase the size of the insert array by another g_l_restart_max slots

final() -

- Close off restart recovery processing

Common Functions

Listing of all externally referenced functions and Stored procedures and description of usage

Functional Level Description

All database interactions required and error handling considerations

All required stock ledger data must be processed to the GL interface table within Retek. Errors with the download will require an abort.

I/O Specification

All files' layouts input and output

None.

Technical Issues

None.

Forecast item sales download [fsadnld]

Functional Area

Retek Demand Forecasting Interface

Module Affected

Fsadnld.pc

Design Overview

This program will extract sales data (current or historical) for all forecastable items to output files. The driving cursors c_dept/c_class/c_subclass will retrieve the forecastable store items/sales information from the item_master, item_loc_soh, item_loc_hist and domain_xxxx tables for items in active status.

Stored Procedures / Shared Modules (Maintainability)

N/A.

Input Specifications

N/A.

Output Specifications

This program produces four output files, all with the same format. One file is created for Regular Sales information, another for Promotional Sales Information, a third for Clearance Sales Information, and a fourth for Issues Sales Information.

Output File Format

Record Name	Field Name	Field Type	Default Value	Description
File Detail	Store	Number(20)	Item_loc_hist. loc	Number of the store in which the item is to be found.
	Item	Varchar2(25)	Item_loc_hist. item	Unique alphanumeric value that identifies the item.
	End of Week Date	Date(8)	Item_loc_hist. eow_date	End of the week date for which the sales totals are gathered.
	Sales_issues	Number(12,4)	Item_loc_hist. sales_issues	Number of units sold for the item/store/date/sales_type combination.

Function Level Description

Main(): Standard retek main() function, calls init(), process(), final() and handles log messaging.

Init(): Calls restart_init(), limits the commit max counter if necessary, sets up temporary output file names and opens these temp output files. Creates final output file names. Gets vdate and next eow date.

Dept_cursor(): This function will open or array fetch the dept driving cursor depending on the action requested from calling function. Signals no data found to calling function if applicable.

Class_cursor(): This function will open or array fetch the class driving cursor depending on the action requested from calling function. Signals no data found to calling function if applicable.

Subclass_cursor(): This function will open or array fetch the subclass driving cursor depending on the action requested from calling function. Signals no data found to calling function if applicable.

Process(): Declares and sizes arrays to hold fetched records and to hold update records. Opens/fetches appropriate driving cursor based on domain level until cursor is empty, calls process_records() for each chunk of data fetched. Calls post_loc() after final chunk of data was fetched.

Format_buffer(): Sets the file write format string.

Process_records(): Calls write_item_files() for each record returned by driving cursor. Handles RR with a LUW of item/loc. Calls restart_commit and post_loc when restart commit is due, also calls restart_file_write for the output files.

Post_loc(): Array updates item_loc_soh based on update array contents and size.

Write_item_files(): Writes data passed in from calling function into the appropriate output file based on sales type.

Size_drive_cur_struct(): Sizes the struct that holds the array fetched records from the driving cursor.

Size_update_struct(): Sizes the struct that holds the item_loc_soh updates.

Final() : Calls final restart_file_write()s as necessary, closes and removes temp output files, calls restart_close().

Scheduling Considerations

N/A.

Locking Strategy

N/A.

Restart/Recovery

At the item/loc level.

Performance Considerations

No change.

Security Considerations

N/A.

Design Assumptions

N/A.

Forecast subclass sales download [fsandnld_sbc]

Functional Area

Allocations, Sales History Download by Subclass

Module Affected

Fsadnld_sbc.pc.

Design Overview

The module creates three output flat files, for regular, promotion and clearance sales, to be used as input to the RPAS system. The module selects records from subclass_sales_hist, using the domain_level from system options. This domain level determines whether dept, class or subclass is the aggregation level. The domain field in any of the above domain tables determines the partition, or thread number.

The flat files, sbc_rsal0xs.dat, sbc_psal%0xs.dat, and sbc_csal%0xs.dat are created with x being the partition or thread number.

Stored Procedures / Shared Modules (Maintainability)

N/A.

Input/Output Specifications

Command Line Usage: Lptotldat logon-string input-file output-file reject-file

- Logon-string: The user's name and password ("username/password"), with an optional database connect string (username/password@database).
- Input-string: The name (with path) of the file from ReSA to be processed. The file name should be suffixed with a dot ('.') and the thread number.
- Output-file: The name (with path) of the file to be SQL*Loaded into LP_TOT_LD_AT. The file name will be suffixed by the program with a dot ('.')

Function Level Description

Main(): Standard retek main() function, calls init(), process(), final() and handles log messaging.

Init(): Calls restart_init(), limits the commit max counter if necessary, sets up temporary output file names and opens these temp output files. Creates final output file names. Gets vdate and next eow date.

Process():

- Determines level for processing: dept, class or subclass
- Calls dept_cursor(), class_cursor() or subclass_cursor
- Loop through array to process records by calling process_records()
until finished.

Dept_cursor():

- Declares department cursor, opens cursor and fetch records.

Class_cursor():

- Declares class cursor, opens cursor and fetch records.

Subclass_cursor():

- Declares subclass cursor, opens cursor and fetch records.

Format_buffer():

- Prepares the buffer for the fetched records

Process_records():

- Calls restart_commit if necessary.
- Writes to regular, clearance and/or promotion files as appropriate.

Write_files():

- Passes fetched record values to the regular, clearance, and/or promotion output buffers as appropriate.

Size_driv_cur_struct():

- Allocates memory for structure variables.

Final() : Calls final restart_file_write()s as necessary, closes and removes temp output files, calls restart_close().

Input Specifications**'Table-To-File'**

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
DOMAIN_DEPT	No	Yes	No	No	No
DOMAIN_CLASS	No	Yes	No	No	No
DOMAIN_SUBCLASS	No	Yes	No	No	No
SUBCLASS_SALES_HIST	No	Yes	No	No	No

Output Specifications**Output File Format**

Record Name	Field Name	Field Type	Default Value	Description
File Detail	Store	Number(20)	Item_loc_his t.loc	Number of the store in which the item is to be found.
	DCS	Number(20)	Item_loc_his t.item	Unique alphanumeric value that identifies the item.
	End of Week Date	Date(8)	Item_loc_his t.eow_date	End of the week date for which the sales totals are gathered.

	Sales_issues	Number(12,4)	Item_loc_hist.sales_issues	Number of units sold for the item/store/date/sales_type combination.

Scheduling Considerations

N/A.

Locking Strategy

N/A.

Restart/Recovery

Restart/recovery is based on the values stored in restart_bookmark from the last commit prior to failure. The values are for department, class, subclass and location.

Performance Considerations

N/A.

Security Considerations

N/A.

Unit Test Considerations

N/A.

Design Assumptions

N/A.

Geocode upload [gcdupld]

Design Overview

The batch program will provide the ability to upload geocodes from an outside source into the RMS tables. The program will allow the deletion of current geocodes and the creation of new geocodes in the system. A flat file will be used to feed the program the additions and deletions to the geocode tables. Validation will occur to determine if duplicate records exist, dependencies exist, and the flat file is in the correct format. If errors occur in the validation of the record, it will be written out to a reject file to allow further investigation of the record.

Assumption: The items that are passed in are only the ones that need to be added or deleted from the system.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: ad hoc

Scheduling Diagram: ad hoc

Pre-Processing: none

Post-Processing: none

Threading Scheme:

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Program Flow

Structure Chart

Main () –

This function will call the appropriate functions within the program

- Init ()
- Process()
- File Process ()
- Final()

Init()

- This function will initialize restart recovery for the program.
- Open a temporary output file in write mode. This file will be used to create the reject file at the end of the program.

- Read in the information that is placed on the flat file. This should be done using the `get_record` command.

Size_arrays()

- This function will size all of the various structures that will be used to insert or delete records from the geocode tables .
- All of the counters that are declared for the array inserts and deletes should be initialized.

Process()

This function drives the population of the geocode temp table. This temp table will be used in the program to facilitate what order the geocodes will be inserted or deleted from the system.

- The input file will have a THEAD line for each geocode in the system. Create a while loop to continue to loop through all of the THEAD lines until the FTAIL is reached in the input file.
- Once a THEAD line is read into the program call the `Validate _THEAD` function to validate that the data being received into the system is valid.
- After the validation of the item has occurred then call the function `insert_temp_records()`. This function will insert a new record into the `geocode_temp` table. The temp table will be used in later functions to determine what geocodes already exist in the system and what geocodes need to be inserted or deleted.

Validate THEAD()-

This function will place all of the values in the THEAD line into local variables such that they can be used throughout the rest of the program.

Insert geocode_temp ()

This function will insert a new record into one of the new temp table. The values that will be inserted are geocode, geocode description, geocode level, and the add/delete indicator.

File_process ()

This function will call the appropriate functions that will determine what geocodes are to be added or deleted from the various geocode hierarchy tables.

The driving cursor for this function will select the geocode, geocode description, level, and the add/delete indicator from the temp table ordered by level, geocode, and the add/delete indicator.

If the add/delete indicator is set to 'A'

- Call the function `add_country_geocode`
- Call the function `add_state_geocode`
- Call the function `add_county_geocode`
- Call the function `add_city_geocode`
- Call the function `add_district_geocode`

- After all of the insert arrays are filled the function insert_geocode() will be called to insert the arrays into the appropriate tables.

If the Flag is set to 'D' then

- Call the function delete_district_geocode
- Call the function delete_city_geocode
- Call the function delete_county_geocode
- Call the function delete_state_geocode
- Call the function delete_country_geocode
- After all of the delete arrays are filled the function delete_geocode() will be called to delete the arrays from the appropriate tables.

Add_district_geocode()

- This function will insert a new district geocode into the district geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country, state, county, and city must all be valid and either exist on the district_geocode, city_geocode, county_geocode, state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_city_geocode()

- This function will insert a new city geocode into the city geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country, state, and county must all be valid and either exist on the city_geocode, county_geocode, state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_county_geocode()

- This function will insert a new county geocode into the county geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country, state, and county must all be valid and either exist on the county_geocode, state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_state_geocode()

- This function will insert a new state geocode into the state geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country must be valid and either exist on the state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_country_geocode()

- This function will insert a new country geocode into the country geocode table. Validate that the country does not exist in the system. If it does then write it out the reject file otherwise add the country geocode into the system.

Insert geocode ()

All of the add functions listed above should fill the following insert arrays for each of the levels: insert_country_geocode, insert_state_geocode, insert_county_geocode, insert_city_geocode, insert_district_geocode. Once all of the arrays are filled this function will insert them into the appropriate tables. Since there are dependencies across the hierarchy the country geocodes must be inserted first, continuing down the hierarchy until the district geocodes are inserted.

Delete_district_geocode()

This function will fill the district delete array with values that need to be removed from the system. District geocodes can always be deleted since they do not have any dependencies.

Delete_city_geocode()

This function will fill the city delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the city exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), and district_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_county_geocode()

This function will fill the county delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the county exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), district_geocodes, and city_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_state_geocode()

This function will fill the state delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the state exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), district_geocodes, city_geocodes, and county_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_country_geocode()

This function will fill the country delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the country exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), district_geocodes, city_geocodes, county_geocodes, and state_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete geocode ()

All of the delete functions listed above should fill the following delete arrays for the various levels: delete_country_geocode, delete_state_geocode, delete_county_geocode, delete_city_geocode, delete_district_geocode. Once all of the arrays are filled this function will delete them into the appropriate tables. Since there are dependencies across the hierarchy the district geocodes must be deleted first, continuing up the hierarchy until the country geocodes are deleted.

Final ()

This function will close all open files and delete all of the information off of the temp files.

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

I/O Specification

All files layouts input and output

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	'GCUP'	Identifies which translation Gentran uses
	Current date	Char(14)		File date in YYYYMMDDHH24MISS format
FDETL	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Country Geocode	Char(4)		Country Geocode
	State Geocode	Char(4)		State Geocode
	County Geocode	Char(4)		County Geocode
	City Geocode	Char(4)		City Geocode
	District Geocode	Char(4)		District Geocode
	Geocode Level	Char(6)		Geocode Level (STATE, CITY, etc)
	Geocode Description	Char(40)		Geocode Description
	Add Delete Ind	Char(1)		Add/delete Indicator
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Generate supplier order numbers [genpreiss]

Functional Area

VMI (Vendor Managed Inventory)

Design Overview

This is a very simple batch program. Based on records on the SUPP_PREISSUE table, it pre-reserves order numbers for suppliers that do VMI by placing these pre-generated order numbers on the ORD_PREISSUE table.

The LUW of this module is a single record from the SUPP_PREISSUE table, which is then used to generate and insert order numbers into ORD_PREISSUE table by calling ORDER_NUMBER_SQL.PREISSUE.

Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init(): Standard Retek init function. Gets vdate and calls retek_init().

Process(): This is the main function that does all the work.

- In a while loop, fetch records from SUPP_PREISSUE table whose NEXT_GEN_DATE value matches the current vdate.
 - For each record from SUPP_PREISSUE table, generate as many order numbers by calling ORDER_NUMBER_SQL.PREISSUE as defined in the fetched SUPP_PREISSUE record's QTY field.
- After order numbers are generated, update the SUPP_PREISSUE table for the current SUPP_PREISSUE record so that the SUPP_PREISSUE.NEXT_GEN_DATE is set to SUPP_PREISSUE.NEXT_GEN_DATE + SUPP_PREISSUE.FREQUENCY.
- Call retek_commit() with the supplier of the current SUPP_PREISSUE record as the argument.

Final(): Standard Retek final function. Calls retek_close().

Scheduling Considerations

This module can be run at any stage in the batch cycle, it is independent of other programs. If a custom program is created to download the pre-issued numbers it will need to be run after genpreiss.pc.

This module is multi-threaded by supplier. See volume testing documentation for optimum thread value. (I suggest 20-30 threads.)

Store grade upload [gradupld]

Design Overview

The Store Grade Upload is designed to load RDF driven Store Grades into the RMS. Data will be loaded into the STORE_GRADE_GROUP, STORE_GRADE and STORE_GRADE_STORE tables. If the store has been assigned to a 'Junk' grade within RDF's Grade the Store's Sister Store Grade assignment will be assigned to the 'Junk' Store.

The Gradconv program will convert RDF's flat file into a standard RMS batch input file.

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: gradconv pre-processes the RDF file.

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

A driving cursor will not exist since this is a file-based processing module.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

The batch program should perform the following steps:

- Extract login ID, password from input arguments.
- Read first record into local variables.
- Validate the Buyer .
- The Store Grade Group will be validated against the STORE_GRADE_GROUP table. If the record does not exist then a record will be inserted into the STORE_GRADE_GROUP table. From here, the remainder of the hierarchy will be new and each Grade and Store records will be inserted into the appropriate tables. If a store has been assigned a 'Junk' Grade ('0') then the grade of its STORE.SISTER_STORE will be retrieved and used as the Store's grade.

I/O SpecificationInput File

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

Record Type	Field	Datatype	Valid Values	Description
FHEAD	Record Type	Char(5)	'FHEAD'	Record Identifier
	Line ID	Number(10)	0000000001	Line Sequence Identifier
	File Name	Char(5)	'GRADU'	File Identifier
FDETL	Record Type	Char(5)	'FDETL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Grade Group ID	Number (8)		Valid Grade Group ID
	Grade Group	Char (20)		Valid Grade Group
	Grade Store	Number (10)		Valid Grade Store
	Grade ID	Number (10)		Valid Grade ID
	Grade Name	Varchar (20)		Valid Grade Name
FTAIL	Record Type	Char(5)	'FTAIL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Line Total	Number(10)		Total number of FDETL lines in the file.

Technical Issues

N/A

Group, department number download [grpdnld]

POS Configuration - General

Module Affected

posgpdlld.pc

Design Overview

Occasionally, retailers need to perform department level sales for merchandise. To accommodate this, a new download program (posgpdlld.pc) is created to download group number/department information to individual Point-of-Sale (POS) systems.

Assumption: The dataset that defines group numbers and departments is small. With this assumption, a full extract of RMS group number/department combinations occurs for every run of the batch program.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: ad hoc

Scheduling Diagram: Run after reclsdly.pc

Pre-Processing: reclsdly.pc

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Logical Unit of Work

Group_no/dept

Driving Cursor

```
SELECT group_no,
       dept,
       ';' || to_char(group_no) || ';' || to_char(dept)
FROM deps
WHERE (group_no > NVL(:ps_restart_group_no, -999)
       OR (group_no = :ps_restart_group_no
           AND dept > :ps_restart_dept))
ORDER BY group_no, dept;
```

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

Function Level Description

All database interactions required and error handling considerations

Global structure of arrays for data processing:

```
struct fetch_rec
{
    char (*as_group_no) [NULL_GROUP_NO];
    char (*as_dept) [NULL_DEPT];
    char (*as_restart_string) [NULL_RESTART]
};
```

Called from init():

```
int get_date()
```

Fetches the vdate (business date) from the period table and places it into a program global variable.

```
int size_array(struct fetch_recs *ioa_data_buffer)
```

Sizes the arrays used in database FETCHES by using the calloc function.

```
void format_strings()
```

Creates format strings used in file outputs.

```
int init()
```

Performs standard Retek restart/recovery initialization. All file handling occurs here, such as declaration of a temp file, and opening of files. The FHEAD line is also written to the file, in the case of a fresh start.

```
int process()
```

The driving cursor is used to fetch records from the database. The group_no and dept are written to the temp file using the FDETL format string defined previously. Once all data has been fetched an FTAIL line is written to the file.

```
int final()
```

Performs closing operations. A call to restart_close() is made, and the temp file is removed from the system.

I/O Specification

Record Type	Field	Datatype	Valid Values	Description
FHEAD	Record Type	Char(5)	'FHEAD'	Record Identifier
	Line ID	Number(10)	0000000001	Line Sequence Identifier
	File Name	Char(4)	'GRPD'	File Identifier
	File Date	Char(14)		Date the file was created in 'YYYYMMDDHHMMSS' format
FDETL	Record Type	Char(5)	'FDETL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Group no	Numeric ID (4) (left-justified character field)		Valid RMS Group Number.
	Dept	Numeric ID (4) (left-justified character field)		Valid RMS Department for the Group Number.
FTAIL	Record Type	Char(5)	'FTAIL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Line Total	Number(10)		Total number of FDETL lines in the file.

Sales history rollup by department, class, and subclass [hstbld]

Design Overview

The sales history rollup routine will extract sales history information for each item from the item_master, and item_loc_hist tables. The history information will be rolled up to the subclass, class, and dept level to be written to: dept_sales_hist, class_sales_hist, and subclass_sales_hist.

For each item, data to be saved includes sales qty, value, gross profit, and sales rate. This data must be collected from several tables including item_master, item_loc_hist and mask_rebuild. Letting the database (server) roll up the totals verse using a loop on the client enhances speed. Using a VIEW that contains all item information for the current week enhances simplicity. Data can then be summed from this single view instead of having to join across all 3 tables in a single select statement.

The rebuild program can be run in one of two ways:

First, if the program is run with a run-time parameter of 'rebuild', the program will read data (dept, class, and subclass) off the manually input mask_rebuild table, which will determine what is rebuilt. This process is used after items are reclassified from one merchandise hierarchy to another. Rebuilding a department will rebuild each class and subclass within the department, thus, only one row is required on mask rebuild for the department. This type of rebuilding process will rebuild data from all dates on the item_master, item_loc_hist table, rolling them to the department, class, and subclass level.

Second, if the program is run with a run-time parameter of 'weekly', the program will build sales information for all dept/class/subclass combinations only for the current end of week date.

mask_rebuild table:

DEPT	CLASS	SUBCLASS	
X	NULL	NULL	Rebuild Department
X	X	NULL	Rebuild Class
X	X	X	Rebuild Subclass

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPT_SALES_HIST	No	Yes	Yes	Yes	No
CLASS_SALES_HIST	No	Yes	Yes	Yes	No
SUBCLASS_SALES_HIST	No	Yes	Yes	Yes	No
ITEM_MASTER	No	Yes	No	No	No
ITEM_LOC_HIST	No	Yes	No	No	No
MASK_REBUILD	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE 3 (weekly) PHASE AD-HOC (weekly)
Scheduling Diagram:	Must run after complete weekly sales have been updated by posupld. Also should be re-run on demand when a sales rollup request has been given for a given dept, class or subclass
Pre-Processing:	N/A
Post-Processing:	hstbld_post() Truncates the mask rebuild table.
Threading Scheme:	N/A (single threaded)

Restart Recovery:

If program is run for reclassifying Items (first run time parameter = 'rebuild'), the driving cursor for the program will be:

```
EXEC SQL DECLARE c_rebuild_dept CURSOR FOR
  SELECT      im.dept,
              im.class,
              im.subclass,
              ilh.loc,
              to_char(ilh.eow_date,'YYYYMMDD'),
              ilh.week_454,
              ilh.month_454,
              ilh.year_454,
              ilh.sales_type,
              NVL(SUM(NVL(ilh.sales_issues,0)),0),
              NVL(SUM(NVL(ilh.value,0)),0),
              NVL(SUM(NVL(ilh.gp,0)),0)
  FROM        item_master im,
              item_loc_hist ilh
  WHERE       im.item = ilh.item
  AND         ilh.sales_type in ('P','R')
  AND         ilh.eow_date = to_date(:os_vdate,'YYYYMMDD')
  AND         ilh.loc_type = 'S'
  AND         (im.dept > NVL(:ov_restart_dept, -999)
  OR (im.dept = :ov_restart_dept
  AND NVL(im.class, 0) > :ov_restart_class))
  GROUP BY    im.dept,
              im.class,
              im.subclass,
              ilh.loc,
              to_char(ilh.eow_date,'YYYYMMDD'),
              ilh.week_454,
              ilh.month_454,
              ilh.year_454,
              ilh.sales_type
  ORDER BY    im.dept,
              im.class,
              im.subclass,
```

```

        ilh.loc,
        ilh.sales_type;

```

If program is run for current end of week (second run time parameter = 'weekly'), the driving cursor for the program will be:

```

EXEC SQL DECLARE c_rebuild_eow CURSOR FOR
  SELECT    im.dept,
            im.class,
            im.subclass,
            ilh.loc,
            to_char(ilh.eow_date,'YYYYMMDD'),
            ilh.week_454,
            ilh.month_454,
            ilh.year_454,
            ilh.sales_type,
            NVL(SUM(NVL(ilh.sales_issues,0)),0),
            NVL(SUM(NVL(ilh.value,0)),0),
            NVL(SUM(NVL(ilh.gp,0)),0)
  FROM      item_master im,
            item_loc_hist ilh,
            mask_rebuild mr
 WHERE      im.item = ilh.item
            AND im.dept = mr.dept
            AND ilh.sales_type in ('P', 'R')
            AND ilh.loc_type = 'S'
            AND (mr.class IS NULL
                  OR (im.subclass = mr.class
                      AND (mr.subclass IS NULL
                          OR im.subclass = mr.subclass)))
            AND (im.dept > NVL(:ov_restart_dept, -999)
                  OR (im.dept = :ov_restart_dept
                      AND NVL(im.class, 0) > :ov_restart_class))
 GROUP BY   im.dept,
            im.class,
            im.subclass,
            ilh.loc,
            to_char(ilh.eow_date,'YYYYMMDD'),
            ilh.week_454,
            ilh.month_454,
            ilh.year_454,
            ilh.sales_type
 ORDER BY   im.dept,
            im.class,
            im.subclass,
            ilh.loc,
            to_char(ilh.eow_date, 'YYYYMMDD'),
            ilh.sales_type;

```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init()

- Initialize restart recovery
- If processing current end of week, call check_eow_date()
- Initialize structure arrays for subclass_sales_hist insert, subclass_sales_hist update, class_sales_hist insert, class_sales_hist update, and sales_history fetch.
- Check_eow_date()
- Check that vdate is a valid end of week date

process()

- Open driving cursor
- Array fetch cursor
- Loop through array to process records.
- Increment dept and class sales variables for running totals of sales, value, gp, and forecast_sales.
- Call process_subclass().
- If dept/class/store/eow_date/sales_type changes, call process_class().
- If dept/store/eow_date/sales_type changes, call process_dept().
- Call process_subclass(), process_class(), and process_subclass() to process last record fetched (last record is not processed within above loop.
- Call insert_class()
- Call update_class()
- Call insert_subclass()
- Call update_subclass

Process_subclass()

- Fetch previous sales, value, gp, and forecast_sales value from subclass_sales_hist
- If fetch is not found (no record exists), add values to subclass_insert array
- If fetch is found,
 - Add values to subclass_update array
- Increment delta variables to hold running totals for difference in subclass' sales and updated sales, value and updated value, gp and updated gp, forecast_sales and updated forecast_sales.

Process_class()

- Fetch rowid from class_sales_hist
- If fetch is not found, add values to class_insert array
- If fetch is found,
 - Add values to class_update array
- Increment delta variables to hold running totals for difference in class' sales, value, gp, and forecast_sales by adding deltas from subclass' sales, value, gp, and forecast_sales.
- Reset subclass delta variables and class running total variables.

Process_dept()

- Perform update of dept_sales_hist
- If update is not found (record does not exist), perform insert into dept_sales_hist

- Reset class delta variables and dept running total variables.

Insert_class()

- Perform array insert of class_sales_hist.

Update_class()

- Perform array update of class_sales_hist.

Insert_subclass()

- Perform array insert of subclass_sales_hist.

Update_subclass()

- Perform array update of subclass_sales_hist.

I/O Specification

N/A

Technical Issues

N/A

Sales history rollup by diff IDs [hstbld_diff]

Design Overview

Item Diff History:

The sales history rollup routine will extract sales history information for each item_parent from the item_loc_hist table. The history information will be rolled up to the item differentiator level to be written to: item_diff_loc_hist and item_parent_loc_hist.

For each ITEM, data to be retrieved includes sales qty and stock. This data must be collected from several tables including item_loc_hist, item_loc, and item_master

The rollup routine will extract sales history information for each ITEM_PARENT from the item_loc_hist table. This history information will be rolled up to the item_diff_loc_hist and item_parent_loc_hist tables. The summation is at the key level: diff_id, store, sales type (P or R) and eow_date.

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_MASTER	Y	N	N	N
ITEM_LOC	Y	N	N	N
ITEM_LOC HIST	Y	N	N	N
ITEM DIFF LOC HIST	Y	Y	Y	N
ITEM PARENT LOC HIST	Y	Y	Y	N
SYSTEM_VARIABLES	Y	N	N	N
PERIOD	Y	N	N	N

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)
PHASE AD-HOC (weekly)

Scheduling Diagram: Must run after complete weekly sales have been updated by posupld or after hstbld. Also should be re-run on demand when a sales rollup request has been given for a given dept, class or subclass (run hstbld) or style/color (run this module).

Pre-Processing: N/A

Threading Scheme: N/A (single threaded)

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init()

- Call retek_init()
- Call check_current_eow()

check_current_eow()

- Open and fetch cursor c_eow_date_unit

process()

- Call process_item()
- Call retek_force_commit()

process_item()

- Wrapper function which calls oracle function *process_item*

final()

- Call retek_close()

I/O Specification

N/A

Technical Issues

N/A

Processing Cursors— contained in package function
hstbld_diff_process.diff_process()

```

MERGE INTO item_diff_loc_hist IDLH
  USING (SELECT im.item_parent,
               NVL(im.diff_1, ' ') AS DIFF_ID,
               ilh.loc,
               TO_CHAR(ilh.eow_date, 'YYYYMMDD') AS
               EOW_DATE,
               ilh.week_454,
               ilh.month_454,
               ilh.year_454,
               ilh.sales_type,
               SUM(NVL(ilh.value, 0)) AS VALUE,
               SUM(NVL(ilh.stock, 0)) AS STOCK
  FROM   item_loc_hist ilh,
         item_loc il,
         item_master im
 WHERE  im.item_parent is NOT NULL
        AND im.diff_1 is NOT NULL
        AND ilh.item = im.item
        AND ilh.item = il.item
        AND ilh.loc = il.loc
        AND ilh.sales_type in ('P', 'R')
        AND ilh.eow_date = I_vdate
 GROUP BY im.item_parent,
          im.diff_1,
          ilh.loc,
          ilh.eow_date,
          ilh.week_454,
          ilh.month_454,
          ilh.year_454,
          ilh.sales_type
 UNION
  SELECT im.item_parent,
         NVL(im.diff_2, ' ') AS DIFF_ID,
         ilh.loc,
         TO_CHAR(ilh.eow_date, 'YYYYMMDD') AS
         EOW_DATE,
```

```

        ilh.week_454,
        ilh.month_454,
        ilh.year_454,
        ilh.sales_type,
        SUM(NVL(ilh.value, 0)) AS VALUE,
        SUM(NVL(ilh.stock, 0)) AS STOCK
    FROM item_loc_hist ilh,
        item_loc il,
        item_master im
    WHERE im.item_parent is NOT NULL
        AND im.diff_2 is NOT NULL
        AND ilh.item = im.item
        AND ilh.item = il.item
        AND ilh.loc = il.loc
        AND ilh.sales_type in ('P', 'R')
        AND ilh.eow_date = I_vdate
    GROUP BY im.item_parent,
        im.diff_2,
        ilh.loc,
        ilh.eow_date,
        ilh.week_454,
        ilh.month_454,
        ilh.year_454,
        ilh.sales_type) S
    ON( IDLH.item = S.item_parent
        AND IDLH.diff_id = S.diff_id
        AND IDLH.location = S.loc
        AND IDLH.eow_date = TO_DATE(S.eow_date, 'YYYYMMDD')
        AND IDLH.sales_type = S.sales_type)
    WHEN MATCHED THEN UPDATE SET IDLH.value = NVL(IDLH.value, 0) +
                                                S.value,
                                IDLH.stock = NVL(IDLH.stock, 0) +
                                                S.stock

    WHEN NOT MATCHED THEN INSERT (item,
                                diff_id,
                                location,
                                loc_type,
                                eow_date,
                                week_454,
                                month_454,
                                year_454,
                                sales_type,
                                value,
                                stock)
        VALUES(S.item_parent,
                S.diff_id,
                S.loc,
                'S',
                TO_DATE(S.eow_date, 'YYYYMMDD'),
                S.week_454,
                S.month_454,
                S.year_454,
                S.sales_type,
                S.value,
                S.stock);

MERGE INTO item_parent_loc_hist IPLH
    USING (SELECT im.item_parent,
        ilh.loc,
        TO_CHAR(ilh.eow_date, 'YYYYMMDD') AS
                                                EOW_DATE,

        ilh.week_454,
        ilh.month_454,
        ilh.year_454,
        ilh.sales_type,
        SUM(NVL(ilh.value, 0)) AS VALUE,
        SUM(NVL(ilh.stock, 0)) AS STOCK
    FROM item_loc_hist ilh,
        item_loc il,
        item_master im

```



```

        WHERE im.item_parent is NOT NULL
              AND ilh.item = im.item
              AND ilh.item = il.item
              AND ilh.loc = il.loc
              AND ilh.sales_type in ('P', 'R')
              AND ilh.eow_date = I_vdate
    GROUP BY im.item_parent,
             ilh.loc,
             ilh.eow_date,
             ilh.week_454,
             ilh.month_454,
             ilh.year_454,
             ilh.sales_type) S
    ON(      IPLH.item = S.item_parent
          AND IPLH.location = S.loc
          AND IPLH.eow_date = S.eow_date
          AND IPLH.sales_type = S.sales_type)
    WHEN MATCHED THEN UPDATE SET IPLH.value = NVL(IPLH.value, 0) +
                                                S.value,
                                                IPLH.stock = NVL(IPLH.stock, 0) +
                                                S.stock
    WHEN NOT MATCHED THEN INSERT (item,
                                   location,
                                   loc_type,
                                   eow_date,
                                   week_454,
                                   month_454,
                                   year_454,
                                   sales_type,
                                   value,
                                   stock)
    VALUES(S.item_parent,
            S.loc,
            'S',
            TO_DATE(S.eow_date, 'YYYYMMDD'),
            S.week_454,
            S.month_454,
            S.year_454,
            S.sales_type,
            S.value,
            S.stock);

```

Purge sales history [hstprg]

Design Overview

Deletes FROM ITEM_LOC_HIST, SUBCLASS_SALES_HIST, CLASS_SALES_HIST, DEPT_SALES_HIST, DAILY_SALES_DISCOUNT where data is more than a specified number of months old. Number of months for retention of fashion style history is specified by unit_options.ITEM_HISTORY_MONTHS.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
ITEM_LOC_HIST	No	No	No	No	Yes
SUBCLASS_SALES_HIST	No	No	No	No	Yes
CLASS_SALES_HIST	No	No	No	No	Yes
DEPT_SALES_HIST	No	No	No	No	Yes
DAILY_SALES_DISCOUNT	No	No	No	No	Yes
SYSTEM_VARIABLES	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Sales history rollup by diff [hstprg_diff]

Design Overview

Diff History:

The tables `item_diff_loc_hist` and `item_parent_loc_hist` need to be purged of records older than a system set date. This date is stored in the `item_history_months` column of `unit_options`.

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)
PHASE AD-HOC (weekly)

Scheduling Diagram: Must run after `hstbld_diff.pc`. Also should be re-run on demand when requested.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery: N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

`init()`

- Fetches system variables.

`process()`

- Deletes records from `item_diff_loc_hist` and `item_parent_loc_hist` which are older than the date calculated backward from `vdate` for the value in `item_history_months`

I/O Specification

N/A

Technical Issues

N/A

Weekly stock on hand, retail and average cost values update [hstwkupd]

Design Overview

Run weekly to update the current stock on hand, retail values and average cost for each item/location on ITEM_LOC_HIST. The program must be run on the last day of the week as scheduled.

Possible values for the inventory history level indicator:

- N - No history information is maintained
- I - Information is only maintained on items sold within the week
- A - Information is maintained on all items with a stock on hand at the end of the week greater than zero, regardless of sales

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: WH
V_restart_store_wh

Restart Recovery

```
EXEC SQL DECLARE c_item_loc CURSOR FOR
  SELECT /*+ INDEX(item_loc) */
    il.item,
    il.loc,
    il.loc_type,
    NVL(ils.stock_on_hand,0),
    NVL(il.unit_retail,0),
    NVL(il.av_cost,0)
  FROM item_loc il,
       item_loc_soh ils,
       v_restart_store_wh rv
 WHERE rv.driver_value = il.loc
    AND il.loc = ils.loc
    AND il.item = ils.item
    AND il.status = 'A'
    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 (il.item > NVL(:ora_restart_item,'') OR
         (il.item = :ora_restart_item AND
          il.loc > :ora_restart_loc))
```

ORDER BY il.item,
il.loc;

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Upload customs tariff files [htsupld]

Design Overview

This batch program will be run whenever an updated US customs tariff file is available (probably twice a year) to upload HTS tariff information from the file into RMS HTS tables. The program will handle both the initial HTS information load as well as mid-year HTS updates that are supplied by the US government. The initial upload is handled by inserting information from the file into the tables; updating information already in the tables is handled by adjusting the effective dates of the existing HTS records and inserting a new set of HTS records into the tables.

Updating HTS records should follow the following guidelines:

- No HTS records with the same HTS and import country should have overlapping effect_from and effect_to dates. Import country is passed as an input parameter to the program, so that the program can support different import countries.
- The new HTS effective dates will never chop up the effective dates of an existing HTS, and there will never be any rollback in dates. Therefore, a new HTS can only start in the middle of an existing HTS or cover a completely different time frame after the existing HTS.
- When loading a new HTS that starts in the middle of an existing HTS, the effect_to date of the existing HTS should be adjusted to one day before the new effect_from date.
- No existing HTS information should be purged by the program. It's the client's responsibility to handle that.

TABLE	SELECT	INSERT	UPDATE	DELETE
HTS	Yes	Yes	Yes	Yes
HTS_TAX	No	Yes	Yes	Yes
HTS_FEE	No	Yes	Yes	Yes
HTS_OGA	No	Yes	Yes	Yes
HTS_TARIFF_TREATMENT	Yes	Yes	Yes	Yes
HTS_TT_EXCLUSIONS	No	Yes	Yes	Yes
TARIFF_TREATMENT	Yes	No	No	No
COUNTRY_TARIFF_TREATMENT	Yes	No	No	No
HTS_CHAPTER	Yes	No	No	No
OGA	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
QUOTA_CATEGORY	Yes	No	No	No
COUNTRY	Yes	No	No	No
HTS_CVD	No	No	Yes	No
HTS_AD	No	No	Yes	No
HTS_REFERENCE	No	No	Yes	No
ITEM HTS	Yes	Yes	Yes	No
ITEM HTS_ASSESS	No	No	Yes	No
ORDSKU HTS	Yes	Yes	Yes	Yes

MOD_ORDER_ITEM HTS	No	Yes	No	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
DUAL	Yes	No	No	No
ORDSKU HTS ASSESS	No	No	No	Yes
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ORDSKU	Yes	No	No	No
CE_CHARGES	Yes	No	No	Yes
CE_ORD_ITEM	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Ad hoc

Scheduling Diagram: Run anytime as needed.

Pre-Processing: after hts upload conversion (**hts2rms – PERL script**).

Post-Processing: None

Threading Scheme: None

Restart Recovery

This program supports Retek standard intermittent commit and file upload restart/recovery. Recommended commit counter is 2000 (commit after every 2000 tariff records are read). Input file names must end in a “.1” for the restart mechanism to properly parse the file name. Since there is only 1 input file to be uploaded, only 1 thread is used. A reject file is used to hold records that have failed processing. The user can fix the rejected records and process the reject file again.

Program Flow

N/A

Shared Modules

ITEM HTS SQL.DELETE_ASSESS – given the item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function deletes the corresponding record from item_hts_assess.

ITEM HTS SQL.DEFAULT_CALC_ASSESS – given the item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into item_hts_assess, it also will potentially call other package functions and update other tables.

LC_SQL.DELETE_LCORDAPP – given the order_no, this function deletes from lc_ordapply table.

OTB_SQL.ORD_UNAPPROVE – given the order_no, this function updates the otb table.

ITEM_ATTRIB_SQL.GET_STANDARD_UOM – given the item_no, item_type and indicator, this function returns the standard_uom, standard_class, and conv_factor.

UOM_SQL.CONVERT – given the to_uom, from_value, from_uom, item, supplier and origin_country, this function returns the to_value.

SQL_LIB.BATCH_MSG – returns error message information.

ORDER HTS_SQL.DELETE_ASSESS -- given the order_no and seq_no, this function deletes from the ordsku_hts_assess table.

ORDER HTS_SQL.DEFAULT_CALC_ASSESS -- given the order_no, seq_no, pack or item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into ordsku_hts_assess, it also will potentially call other package functions and update other tables.

CE_CHARGES_SQL.INSERT_COMPS – given the ce_id, vessel_id, voyage_flt_ind, order_no, item, pack_item, hts, import_country_id, effect_from, effect_to, cvb_code this function inserts into the ce_charges table.

Function Level Description

Main

Standard Retek main function. This program takes in four parameters: userid/passwd, input file, reject file, import country id.

Init

- A global variable is used to hold the import country id that is passed in as a program input parameter. Call check_country to make sure that import country exists on the COUNTRY table; return with fatal error if not. It is used as the import country throughout the program.
- Open input file for read and open reject file for write.
- Call retek_init() for restart/recovery initialization.
- If it is a fresh start, call retek_get_record to read the FHEAD line into the fhead structure.
- Fetch vdate from period table.
- Fetch max_item from hts table and max ct from ordsku_hts and max ct from ce_charges.
- Fetch update_item_hts_ind and update_order_hts_ind from the system_options table
- Call check_spi to make sure that 'C1' and 'C2' exist in the TARIFF_TREATMENT table as SPI's. 'C1' and 'C2' are default tariff treatments for every HTS. Return with fatal error if not.

File_process

- Call function retek_get_record in a while loop to read the THEAD line into the thead structure:
 - if the record type returned is 'FTAIL', exit the loop;

- set a save point.
- If the record type returned is 'THEAD', read the THEAD line into the thead structure that contains V1, V2, V3, V4 fields. The V4 record is not currently used in RMS/RTM.
- If the record type returned is other than 'FTAIL' or 'THEAD', give a fatal error (wrong record type).
- Call function process_THEAD to further process data contained in the THEAD. Set process error flag to indicate non-fatal process error.
- Call function retek_get_record in a while loop to read the TDETL line into the tdetl structure:
 - if the record type returned is 'TTAIL', exit to the outer loop to continue reading THEAD records if any exists;
 - if the record type returned is other than 'TDETL' or 'TTAIL', give a fatal error (wrong record type).
 - Call function process_TDETL to further process data contained in the TDETL. Set process error flag to indicate non-fatal process error.
- If update_item_hts_ind = "Y",
 - If tran_code is "A" or "R", call item_hts_update function. "A" stands for Update only and "R" stands for Replace. In both of these cases (as opposed to the other possibility of "D" for Delete) item tables will need to be updated.
 - If update_order_hts_ind = "Y", call ordsku_hts_search function.
- If process error flag is set. Rollback database process to the save point. Write rejected records to the reject file.
- Call restart_force_commit to perform intermittent commit for restart/recovery.

Process_THEAD

Fill the hts_keys structure with data from THEAD.

After fill in the hts_keys, verify that effect_from < effect_to date. If not, reject the record right away. Call valid_all_numeric function to check effect_from, and effect_to field. If invalid reject the record.

This function processes the information in V1, V2 and V3 records based on the transaction code ("A", "R", "D") in the V1 record. It compares the new effective dates against those of any existing HTS records with the same HTS code and import country.

If the transaction code type is 'A', insert a record into the HTS table; if the transaction code type is 'R', update the HTS record that has the same HTS code, import country id, effect_from and effect_to dates

For transaction code "A":

If new HTS covers a time period different than and after any existing HTS, or no HTS exists for the given HTS/import country, is a valid record for inserting.

If new the HTS record is overlapping with existing record and its effect_from date > existing record and effect_to >= existing effect_to date, it is a valid record. Process is as follows:

1. Insert an HTS record with the same data as the existing overlapping HTS, except that the effect_to date should be 1 day before the effect_from date of the new HTS record;
1. Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.

2. Insert new hts to the related tables.

Detailed technical description:

- Call function `validate_hts_update` to verify that the record is valid for insert/update to the database or reject to the reject file. For the valid record call `hts_child_update` function to prepare child table processing.
- Call `hts_table_insert` function to insert record to the hts table. if any invalid information exists, write to error file.
- Call `hts_oga_insert` function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.
- Call `hts_spi_insert` function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.
- Call `hts_gsp_insert` function to insert record/s to the hts_tt_exclusions table. if any invalid information exists, write to error message log file.

Set process error flag if non fatal error occurs. Return error flag.

For transaction code "R":

- 1) Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
- 2) If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates:
- 3) insert to update the HTS table and re-insert child table information from the input file.

Detailed Technical Description:

- Call function `search_hts_update` to find record that can be updated in the database tables. If one exists, prepare child tables for processing.
- Call `hts_table_insert` function to insert record to the hts table. if any invalid information exists, write to error file.
- Call `hts_oga_insert` function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.
- Call `hts_spi_insert` function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.
- Call `hts_gsp_insert` function to insert record/s to the hts_tt_exclusions table. if any invalid information exist, write to error message log file.

Set process error flag if non fatal error occurs. Return error flag.

For transaction code "D":

1. Search for the HTS with same HTS, import country id, effect_from and effect_to dates.
 2. If a record is found update HTS and all its child records to yesterday.
- (Note: since the dates are still presented in 2-digit year in the 99 tape, we assume that the year coming in as 00-49 means 2000-2049, and 50-99 means 1950-1999. The customs uses '999999' to mean Dec 31st, 2039.)

Detailed Technical Description:

Call function `search_hts_reset` to find updateable record in the hts table. If one exists, insert new hts record. Call function `hts_child_update` to update all the child records, then delete the existing hts record.

`Validate_hts_update`

1) new HTS starts before or on the same day as any existing HTS, or
 new HTS starts after and ends before any existing HTS:
 $\text{effect\_from} \geq \text{new effect\_from}$ OR
 $\text{effect\_from} < \text{new effect\_from}$ and $\text{effect\_to} > \text{new effect\_to}$

This is an invalid record. Write the record to the reject file, write an error message to the message log file, and return to the calling function with a non-fatal error.

2) new HTS starts after and overlaps with an existing HTS:
 $\text{effect\_from} < \text{new effect\_from}$ and $\text{effect\_to} \geq \text{new effect\_from}$ or new HTS starts after old end date and therefore does not overlap at all. The ranges are completely separate.

This is a valid record, and a most likely scenario. Fetch the effect_from and effect_to of the existing HTS. Insert a new record with effect_from date same as existing overlapping hts record and effect_to date is 1 day before the new effect_from date to hts table.

Call function `hts_child_update` function to update effect_to date of all child records to 1 day before the new effect_from date.

Delete the old record from hts table.

Search_hts_update

1. Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
2. If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates:

HTS_TT_EXCLUSIONS
 HTS_TARIFF_TREATMENT
 HTS_OGA
 HTS_TAX
 HTS_FEE

Note: HTS table record cannot be deleted due to the other child tables on HTS: ITEM_HTS, ITEM_HTS_ASSESS, ORDSKU_HTS, HTS_CVD, HTS_AD, HTS_REFERENCE, HTS_CHAPTER. The information on these tables won't be loaded in the HTS upload process.

Seach_hts_reset

1. Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
2. Insert into HTS, all the same information, but inserting yesterday as the new to_date .
3. If a record is found, call `hts_child_update` function to update the records in the child tables with effect_to date to yesterday:

Hts_child_update

This function updates the effect_to date of the existing overlapping HTS record on child tables. Since the child tables have referential constraints on the effective dates of the parent table HTS.

Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.

The following child tables should be updated:

HTS_TARIFF_TREATMENT

HTS_TT_EXCLUSIONS

HTS_AD

HTS_CVD

HTS_OGA

HTS_REFERENCE

HTS_TAX

HTS_FEE

ITEM HTS

ITEM HTS_ASSESS

ORDSKU HTS

CE_CHARGES

Note: since table HTS_TT_EXCLUSIONS has a foreign key on the effect_to date of table HTS_TARIFF_TREATMENT, we cannot update the effect_to date of HTS_TARIFF_TREATMENT directly. Likewise, insert an HTS_TARIFF_TREATMENT record with the new effect_to date first; then update the effect_to date of the HTS_TT_EXCLUSIONS table; at the end delete the HTS_TARIFF_TREATMENT record with the original effect_to date.

Call delete_ord_temp_tables and pass in the value “-1” because there is no known order_no at this point.

Item_hts_update

- 1) Call size_item_array function to allocate space for the items
- 2) Fetch item, origin_country_id and status from item_hts into struct
- 3) If no data found, call free_itemlist and go to the next record. If data is found,
- 4) Loop
 - If tran_code = “A” the item will need to be inserted with the same data as the fetched record but with new effect_to and effect_from dates.
 - Insert dates into item_hts
 - Delete old record from item_hts
 - call the package SQL Delete assess to delete the old records from item_hts_assess.
 - If tran_code = “R”
 - Call SQL Delete_assess to delete the old records from item_hts_assess
 - Call SQL Default_calc_assess to update the item_hts_assess table (ie insert record with new dates and recalculate)
 - Call ECL_CALC_SQL.CALC_COMP to recalculate expenses based on new assesses.
 - Insert into mod_order_item_hts a new record with same data but new dates.
 - Call free_itemlist

Ordsku_hts_search

- 1) Call size_ord_array function to allocate space for the order information

- 2) Fetch values from ordhead, ordsku_hts, ordsku and ordloc into struct (all necessary values to be able to do a complete insert into the mod_order_item, ordsku_hts, and ordsku_hts_assess tables.
- 3) If no data found, call free_ordlist and go to next record. If data is found,
- 4) Loop
 - If order status = "A", (the order needs to be updated) set status from approved back to worksheet by calling SQL functions (LC_SQL.DELETE_LCORDAPP and OTB_SQL.ORD_UNAPPROVE).
 - Insert into mod_order_item_hts table (just the order_no and indicator set to 'Y')
 - Call ordsku_hts_update
 - Call free_ordlist

Ordsku_hts_update

- 1) Call size_ce_array to allocate space for the custom entry information
 - Fetch custom entry values from ce_ord_item, ce_head, item_supp_country into struct
 - If no data found, call ordhts_update. If data is found,
 - If CE status = "W", (worksheet status)
 - Call ordhts_update
 - Loop for each custom entry record
 - Call ce_update
 - If status != "W" then the quantity cleared will need to be compared to the total quantity. In order to do that they will need to be converted to the standard uom format
 - Loop
 - Call uom_convert to get the total quantity.
 - If total_qty < qty_ordered
 - Call ordhts_update
 - Call free_ceordlist

Ordhts_update

- 1) if tran_code = "A" or "D"
 - Delete old record (record with old dates) from ordsku_hts_assess
 - Delete old record (record with old dates) from ordsku_hts
 - Insert record with new dates into ordsku_hts
- 2) Else if tran_code = "A"
 - Insert record with new dates into ordsku_hts
- 3) else if tran_code = "D"
 - Call SQL Delete_assess by calling order_del_assess function
 - Call SQL calc_comp
 - If the item is a pack item check to see if a record already exists on mod_order_item_hts – if it does not, insert one with the pack_item
 - If it is not a pack item, insert with item_no into mod_order_item_hts.
 - Return 0
- 4) Else if tran_code = "R"
 - Call delete_ord_temp_tables and pass in the order_no.
- 5) Call SQL Delete_assess by calling order_del_assess function
- 6) Call ORDER HTS_SQL.DEFAULT_CALC_ASSESS with either the pack_no or item_no depending on if it is a pack or not.
- 7) Call ELC_CALC.CALC_COMP
- 8) If it is a pack item insert into mod_order_hts with the pack_no
- 9) If it is not a pack item, insert into mod_order_item_hts with the item_no

Ce_update

- 1) Delete from ce_charges.
- 2) if it is a “D”, call CE_CHARGES_SQL.INSERT_COMPS

Hts_table_insert

Before inserting into or updating the HTS table,

- 1) Call function check_chapter to make sure that the chapter already exists on the HTS_CHAPTER table. If not, reject the record;
- 2) Call check_valid_all_numeric function to check unit for all numeric value.
- 3) Call function check_uom to make sure that the UOMs (UOM1, UOM2, UOM3) already exist on the UOM_CLASS table. Reject the record if UOM does not exist.
- 4) Call function check_duty to make sure that the duty code already exists on the CODE_DETAIL table. If not, reject the record.
- 5) Call valid_all_numeric function to verify that the quota is all numeric. Then calling function check_quota to make sure that the quota category already exists on the QUOTA_CATEGORY table. If not, reject the record.

Update the existing hts record with the updated hts_desc, chapter, units, units_1, units_2, units_3, duty_comp_code, more_hts_ind, quota_cat, quota_ind, ad_ind, cvd_ind.

Insert the following into the HTS table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- hts_desc: commodity description (V1i)
- chapter: 1st 4 (leftmost) digits of tariff number
- units: number of reporting units (V1g)
- units_1: first unit of measure (V1h) (If the number of reporting units is zero, this should be defaulted to ‘X’)
- units_2: second unit of measure (V1i)—NULL if not given
- units_3: third unit of measure (V1j)—NULL if not given
- duty_comp_code: duty code (V1k)
- more_hts: Y if additional tariff indicator (V2j is ‘R’, N otherwise
- quota_cat: category number (V3h) but only if quota indicator (V3g) is 1
- quota_ind ‘Y’ if there is a quota, ‘N’ otherwise
- ad_ind ‘Y’ if the anti-dumping flag (V3f) is 1, N otherwise
- cvd_ind ‘Y’ if the countervailing duty flag (V2k) is 1, N otherwise

Hts_oga_insert

For each OGA code, call function check_oga to verify that the OGA code exists on the OGA table. If not, reject the record; otherwise, call hts_oga_insert to insert into HTS_OGA.

- Insert the following into the HTS_OGA table:
- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- code: OGA code from OGA codes field (V3f)
- reference_id: NULL

- comments: NULL

Hts_spi_insert

For each SPI, call function check_spi to check if the SPI exists on the tariff_treatment table; if not, reject the record. Call function hts_tariff_treatment_insert to insert into HTS_TARIFF_TREATMENT. In addition to the SPI records in V3, 'C1' and 'C2' are default tariff_treatments for every HTS. So, two extra records should be inserted into HTS_TARIFF_TREATMENT with SPI codes 'C1' and 'C2'. 'C1' takes the special_duty_rate from V1 and Column 1 rates from V2; 'C2' takes Column 2 rates from V2.

Before inserting, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Call valid_all_numeric function to check specific_rate, ad_rate, other_rate for all numeric value. If not, reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

Insert the following into the HTS_TARIFF_TREATMENT table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: SPI code from V3i
- specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)
- av_rate: 0,col1, or col2 ad valorem rate, as appropriate (0 for SPI's)
- other_rate: 0,col1, or col2 other rate, as appropriate (0 for SPI's)

Hts_gsp_insert

For each GSP excluded country, call function check_country_tariff_treatment to check that the country and tariff treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; if not, reject the record.

Insert the following into the HTS_TT_EXCLUSIONS table

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: first SPI code from V3i
- origin_country_id: excluded country code from V3d (GSP excluded countries)

Check_spi

Check to see if SPI exists on TARIFF_TREATMENT table; reject the record if it doesn't.

Check_country

Check to see if country exists on COUNTRY table; reject the record if it doesn't.

Check_chapter

Check to see if chapter exists on the HTS_CHAPTER table and reject the record if it doesn't.

Check_uom

Check to see if uom exists on UOM_CLASS table; reject the record if it doesn't.

Check_duty

Check to see if duty code exists on CODE_DETAIL table (check for the code where code_type='DCMP'); reject the record if it doesn't.

Check_quota

Check to see if the quota_category exists on the QUOTA_CATEGORY table; reject the record if it doesn't.

Check_oga

Check to see if the oga code exists on the OGA table; reject the record if it doesn't.

Check_comb_country_tt

Check to see if the country and tariff_treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; reject the record if it doesn't.

Process_TDEL

Format the tax line information from tdel structure.

Call function process_taxfee, if no non-fatal error in the process_THREAD function.

Process_tax_fee

If tax specific rate or tax ad rate is not null, call hts_taxfee_insert to insert the tax rates into HTS_TAX or HTS_FEE tables. If special rates exist on the tax line, call function hts_tariff_treatment_insert to insert into the HTS_TARIFF_TREATMENT table using the ISO country code as the tariff treatment (SPI). If the SPI given on the tax line already exists for the HTS, the record should be updated, as the tax line special rate takes precedence over the V3 line SPI's rate

Call valid_all_numeric function to check tax_specific_rate, tax_av_rate, fee_specific_rate, fee_av_rate for all numeric value, if not reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

Hts_taxfee_insert

If the tax class code is 016,017,018,or 022 it is a tax; insert into HTS_TAX

If the tax class code is 038,053,054,055,056,057,079,090,103 it is a fee; insert into HTS_FEE

Insert the following into the HTS_TAX or HTS_FEE table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tax_type/fee_type: tax class code (V5h)
- tax_comp_code/fee_comp_code: tax comp code (V5i)
- tax_specific_rate/fee_specific_rate: tax specific rate (V5k)
- tax_av_rate/fee_av_rate: tax ad valorem rate (V5l)

Hts_tariff_treatment_insert

Before calling this function, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Insert the following into the HTS_TARIFF_TREATMENT table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: SPI code from V3i
- specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)
- av_rate: 0,col1 or col2 ad valorem rate, as appropriate (0 for SPI's)
- other_rate: 0,col1 or col2 other rate, as appropriate (0 for SPI's)

Size_item_array

Allocates space for the item array struct

Size_ord_array

Allocates space for the order array struct

Size_ce_array

Allocates space for the custom entry array struct

Free_orditemlist

Frees the space in the array

Free_itemlist

Frees the space in the array

Free_ceordlist

Frees the space in the array

Uom_convert

Calls ITEM_ATTRIB_SQL.GET_STANDARD_UOM

Calls UOM_SQL.CONVERT

Order_del_assess

Calls ORDER HTS_SQL.DELETE_ASSESS

Delete_ord_temp_tables

If an order no is not passed in, look at the hts table and see if there is an order that exists for that hts. If so, loop and for each record see if there is a record to delete on the temp tables by calling ORDER_SETUP_SQL.DELTE_TEMP_TABLES.

If the order number was passed in, call ORDER_SETU_SQL.DELETE_TEMP_TABLES right away.

Final

Restart/recovery close and close input and reject file.

Why HTS lines that have a rate greater than 9999999999 need to be rejected:

For fields specific_rate, av_rate, other_rate, RMS has the data type Number(12,8) and numbers coming in from the customs tape also have 8 implied digits. However, when storing the number into the Retek database, we need to divide the number coming in from the customs tape by 1000000 (left shift 6 digits) instead of 100000000 (left shift 8 digits). This is because Retek stores the percent part of the rate only. In other words, rate 11.5% (0.115) is stored as 11.5 in Retek database, whereas it will come in from the customs tape as 11500000 (=0.115). Therefore, the highest rate that can be represented in Retek is 9999.99999999% (= 99.999999999, or < 100 times). So we need to reject HTS lines that have rate greater than 9999999999.

(Note: this is true for hts spi and hts tax/fee specific_rate and av_rate, except that when 999999999999 (12 nines) are used, it represents a special code for NULL.)

I/O Specification

Here is the layout of the input file to be uploaded:

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Retek file ID	Char(5)	HTSUP	Describes file type
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(14)		Unique transaction id
	HTS Line	Char(320)		V1 through V4 records from the customs HTS file concatenated together
TDETL	Record descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(10)		Unique transaction id
	Tax/fee line	Char(80)		V5 through V9 records from the customs HTS file, each on a separate TDETL line
TTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Detail lines	Number(6)		Number of lines between THEAD and TTAIL
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction	Number(10)		Number of lines between

lines

FHEAD and FTAIL

Here is the layout of the original input file:

Note: The input file contains lines of 720 characters, i.e. the newline character occurs only after every 720 characters. Each 720-character line consists of nine 80-character records. Each 80-character record starts with 'V1' or 'V2' ... or 'V9' or blank if the record is completely empty. For each tariff, records V1 and V2 are mandatory; records V3 through V9 are optional, which means they can be all blank. Record V4 is not currently used in RMS/RTM. Records V5 through V9 contain the tax/fee information for the tariff, and all have the same structure. The lower-case letters in the record name block are as a convenience to cross-reference with the US Customs file description.

Record Name	Field Name	Field Type	Default Value	Description
V1a	Control identifier	Char(1)	V	Identifies start of record
B	Record type	Char(1)	1	Identifies record type
c	Tariff number	Number(10)		unique tariff number—will be left justified and space filled if less than 10 digits
d	transaction code	Char(1)	A, D, R	Valid codes are: A = Add D = Delete R = Replace
e	begin effective date	char(6)		MMDDYY format—date this tariff record becomes effective
f	end effective date	char(6)		MMDDYY format—last date this tariff record becomes effective; can be '999999'
g	number of reporting units	number(1)	0,1,or 2 or 3	Number of census reporting units required
h	1 st reporting unit of measure	char(4)		unit of measure ("X" if none required)
I	2 nd reporting unit of measure	char(4)		unit of measure—may be blank
j	3 rd reporting unit of measure	char(4)		unit of measure—may be blank
k	duty computation code	char(1)		Indicates formula to compute duty
l	commodity description	char(30)		Description of tariff commodity
m	column 1 specific rate of duty	Number(12)		specific rate listed in general column 1 rate of duty 8 implied decimal places
n	base rate	char(1)	'B' or	If B, duty rate involves a

		indicator		blank	base rate. Not used in RMS.
		o	space fill	char(1)	blank
V2	a	Control identifier	char(1)	V	Not used in RMS. Identifies start of record
b		Record type	char(1)	2	Identifies record type
c		tariff number	Number(10)		Unique tariff number
d		general column 1 ad valorem percentage	Number(12)		ad valorem rate 8 implied decimal places
e		column 1 other	Number(12)		specific rate if exists, or other ad valorem percentage if needed for duty computation 8 implied decimal places
f		Column 2 specific rate	Number(12)		specific rate 8 implied decimal places
g		Column 2 ad valorem percentage	Number(12)		ad valorem percentage 8 implied decimal places
h		Column 2 other rate	Number(12)		other specific rate or ad valorem percentage 8 implied decimal places
i		countervailing duty flag	char(1)	blank or 1	subject to CVD if 1; space fill otherwise
j		additional tariff indicator	char(1)	blank or 'R'	space fill if no additional tariff number needed; 'R' if it is
k		space fill	char(6)	blanks	Not used in RMS.
V3	a	Control identifier	char(1)	V	identifies start of record
b		Record type	char(1)	3	identifies record type
c		tariff number	Number(10)		unique tariff number
d		GSP excluded countries	char(20)		Up to 10 2-char ISO country codes
e		OGA codes	char(15)		Up to 5 other gov agency codes
f		anti-dumping flag	char(1)	1 or blank	if 1, may be subject to dumping duties
g		quota indicator	char(1)	1 or blank	if 1, tariff subject to possible quota
h		category number	char(6)		quota category
I		special program indicators	char(20)		up to 10 2-char codes
j		space fill	char(5)	blanks	Not used in RMS.
NEWLINE				\n	
V4		Control identifier	char(1)	V	identifies start of record
a		identifier			Entire V4 record not used

					in RMS.
b	Record type	char(1)	4		identifies record type
c	tariff number	number(10)			unique tariff number
d	value edit code	char(3)			defines value edit
e	value low bounds	number(10)			minimum or lower value to use in edit (can be blank) 5 implied decimal places
f	value high bounds	number(10)			max or higher value to use in edit (can be blank) 5 implied decimal places
g	entry date restriction	number(1)	0,1, or 2		date edit code
h	beginning restriction date	char(4)			MMDD format—beginning/earliest date for edit (can be blank)
I	end restriction date	char(4)			MMDD format—end/latest date for edit (can be blank)
j	entry date restriction 2	number(1)	0,1, or 2		date edit code (can be blank)
k	beginning restriction date 2	char(4)			MMDD format—beginning/earliest date for edit (can be blank)
l	end restriction date 2	char(4)			MMDD format—end/latest date for edit (can be blank)
m	country of origin	char(2)			country of origin ISO code or 01 (must be Col 1 country) or 02 (must be Col 2 country)
n	space filler	char(2)	blanks		
o	quantity edit code	char(3)			defines quantity edit
p	low quantity	number(10)			min/lower qty for edit 5 implied decimal places
q	high quantity	number(10)			max/higher qty for edit 5 implied decimal places
V5	a	Control identifier	char(1)	V	identifies start of record
b	Record type	char(1)	5,6,7,8,9		identifies record type
c	tariff number	number(10)			unique tariff number
d	Country code	char(2)			ISO code of country or 'E ' (E plus space) or 'J ' (J plus space) where special rate applies
e	specific rate	number(12)			Col 1 special rate 8 implied decimal places
f	ad valorem rate	number(12)			Col 1 special ad valorem rate 8 implied decimal places

g	Other rate	number(12)		Col 1 other special rate if needed 8 implied decimal places
h	tax/fee class code	char(3)		Identifies specific tax or user fee
I	tax/fee comp code	char(1)		used to compute amt of tax/fee
j	tax/fee flag	number(1)		1—required 2—may be required
k	tax/fee specific rate	number(12)	blank if no value	Not used in RMS. Specific rate to calculate tax or fee 8 implied decimal places
l	tax/fee ad valorem	number(12)	blank if no value	ad valorem rate to compute tax or fee 8 implied decimal places
m	space fill	char(1)	blank	

V6 through V9 records have the same fields as the V5 record.				

Investment buy calculation [ibcalc]

Design Overview

The ibcalc.pc batch program is the engine of investment buy processing. It identifies investment (IB) buy opportunities and calculates recommended order quantities (ROQs) that will meet the target return-on-investment (ROI)

This module will calculate forward buy opportunities using:

- Carrying costs
- Ordering parameters
- Deals
- Cost changes
- Forecasts
- Inventory levels
- Target ROI (return on investment)

The deals and cost change components will be contained on a FUTURE_COST table. This table will hold a tuple for each future date that has a costing event (i.e. a cost change, deal activation/deactivation). We are assuming default costing bracket and default deal thresholds.

Investment Buy Eligibility

The supplier/country/dept/loc must be IB eligible – or it must exist on SIM_EXPL.

The item must have a cost event (on FUTURE_COST) between now and the look-ahead date.

The item must be forecastable. The item must be a transaction level item. The item must be approved. The item must be active at the WH.

Once eligible item/locs are found, their simple pack (if it exists) is looked up. If the item is on replenishment, the primary replenishment pack on REPL_ITEM_LOC is used to order the IB qty. If the item is not on replenishment, the primary costing pack on ITEM_LOC is used to order the IB qty. If neither of these packs exist, the item itself is used to order the IB qty.

Stored Procedures / Shared Modules (Maintainability)

Program Flow

Function Level Description

Data structures:

Init()

Set system level parameters: vdate, system level investment buy defaults

Initialize the restart recovery.

Process()

This will have the standard process function format –

```

    WHILE driving cursor brings back records

        Array fetch the driving cursor

        FOR each record fetched loop

            Process driving cursor records

        End FOR loop

    End WHILE loop

```

Define the driving cursor:

The driving cursor is based off records on SIM_EXPL. SIM_EXPL hold pre-qualified supplier/country/dept/location combinations that are eligible for IB and their IB parameters.

The driving cursor will be a two-part UNION. The first part retrieves the current cost by getting the record with the greatest date less than or equal to today from FUTURE_COST. This cursor also brings back all of the look up information that is needed to perform the IB calculations and ensures that the item is IB eligible. It joins FUTURE_COST and SIM_EXPL to ITEM_LOC, ITEM_LOC_TRAITS, REPL_ITEM_LOC, ITEM_MASTER, PACKITEM, ITEM_SUPP_COUNTRY, and ITEM_SUPP_COUNTRY_LOC.

The second part of the UNION brings back all future cost events between tomorrow and the look-ahead date. Much of the information brought back by the first UNION is not needed in this cursor as it is already know (i.e. it will have the same shelf life for the current cost record as for all the future cost records -- see *int init_process_struct()*) – allowing us to skip some of the joins. We still need to ensure that only eligible items are brought back. It joins FUTURE_COST and SIM_EXPL to ITEM_LOC, REPL_ITEM_LOC, and ITEM_MASTER.

```

int calc_ib(fetch_struct *ia_driver,
            process_struct *ioa_process,
            solution_struct *ioa_solution)

```

This function determines if the passed in driving cursor belongs to the same item/loc combination as the last group. If it does, it sends the record to *process_cost_event()*. If it doesn't is calls *process_completed_group()* to clean up the last group and calls *init_process_struct()* to initialize the new group.

```

int init_process_struct(fetch_struct *ia_driver,
                       process_struct *iot_process)

```

This function is called with current cost records only, never with future_cost records from the driving cursor (this is why we can avoid some of the joins in the second union in the driving cursor). This function moves a driving cursor record from the fetch array to the process struct. It also initializes the calculation elements of the process struct such that the first future cost event will be processed. It also does the UOM conversions necessary for the storage cost.

```
int process_cost_event(fetch_struct *ia_driver,
                      process_struct *iot_process)
```

Determines the optimum number of days of supply to orders base on the IB algorithm. Group stop is tripped when we have determined the solution days and any additional future_cost records will not have effect and should be ignored:

```
    if (group_stop)
        return 0;
    If current event day <= solution day of previous event
        If current event cost <= item/loc's current cost cost
            Final solution day = current event day + lead times
            Set group stop;
        Else if current event cost >= previous event cost
            Call calc_solution_day() to calculate the solution day based on the
            current event
            If first cost event for group set next event date
            Set prev future_cost and prev solution days
        Else if item/loc's current cost < current event cost < previous event cost
            Call calc_solution_day() to calculate the solution day based on the
            current event
            If current solution day < current event day + lead time
                Final solution day = current event day + lead times
                Set group stop
            Else
                Final solution day = solution day for current event
                Set group stop
        End if
    End if
Else
    Set group stop
End if
```

```
int calc_solution_day(process_struct *it_process,
                     double      id_future_cost,
                     long        *iol_solution_days)
```

This function calculates the max numbers of days that can be bought while exceeding the target ROI. It uses the below formula to calculate the number of days.

$$D = (FC - CC + (CC * CM * ST) - HaC) / (HoC + CC * CM + (CC * TROI / A))$$

D = solution day

TROI = target return on investment

CC = current unit cost

FC = future unit cost

CM = cost of money

ST = supplier terms

HoC = holding costs per unit

*HaC = handling costs per unit

A = annual factor

```
int uom_convert(char *is_item,
               char *is_from_uom,
               char *is_to_uom,
               double *iod_storage_cost)
```

This function converts a passed in value from one UOM to another. It is basically a wrapper for a call to UOM_SQL.CONVERT.

```
int process_completed_group(process_struct *iot_process,
                           solution_struct *ioa_solution)
```

This function is passed in a process struct containing the solution day. First it calculates the ROI by calling get_roi(). Next it gets the SOH and forecast by calling get_soh_forecast(). The ROQ is set to Forecast – ROQ.

It then populates the solution array with the ROQ by calling copy_info_to_solution()

```
int get_roi(process_struct *iot_process)
```

Given a solution day, this function determines the return on investment using the factors shown below

ROI = return on investment
 FC = future unit cost
 CC = current unit cost
 CM = cost of money
 D = solution day
 ST = supplier terms
 HoC = holding costs per unit
 *HaC = handling costs per unit
 A = annual factor

$$ROI = (((FC-CC)-((CC*CM*(D-ST))+HaC))/CC)*(A/D)$$

```
int get_soh_forecast(char *is_item,
                    char *is_dept,
                    char *is_class,
                    char *is_subclass,
                    char *is_ib_loc,
                    long il_days,
                    double *iod_soh,
                    double *iod_forecast)
```

This function is the interface to a group of functions that determine the total stock on hand for the passed in item/loc. These function need to consider wh-link and sub-item information.

The sub-items are looked up at the replenishment wh that the ib wh is linked to by the wh.repl_wh_link field.

The `wh.ib_wh_link` is used to determine which wh locations should be considered in the forecast and soh calculations.

This function opens a cursor to get the sub-items defined at the `repl_wh_link` wh. It then calls `get_sho_forecast_locs` for each sub-item found as well as for the master item. It keeps a running total of the forecast and soh returned.

```
int get_soh_forecast_locs(char *is_item,
                        char *is_dept,
                        char *is_class,
                        char *is_subclass,
                        char *is_use_stock_ind,
                        char *is_use_forecast_ind,
                        char *is_ib_loc,
                        long il_days,
                        double *iod_soh,
                        double *iod_forecast)
```

This function loops through all wh locations that are linked to the passed in location by the `wh.ib_wh_link` field. For each location found, it call `soh_pkg_call()` and `forecast_pkg_call()` (depending on the use indicators) and sums up the results.

```
int get_forecast(fetch_struct *ioa_driver)
```

Wrapper function for `REPL_ORDER_QTY_SQL.GET_FORECAST_FOR_PERIOD`.

```
int get_soh(fetch_struct *ioa_driver)
```

Wrapper function for `ITEMLOC_QUANTITY_SQL.GET_LOC_FUTURE_AVAIL`.

```
void copy_info_to_solution(solution_struct *ioa_solution,
                        process_struct *i_process)
```

Populates a record in the solution array based on the information in the passed in process struct.

```
int flush_ib_solution_array(solution_struct *ioa_solution)
```

Inserts rows into `IB_RESULTS` based on the records in the solution array.

Input Specifications

'Table-To-Table'

Output Specifications

'Table-To-Table'

`IB_RESULTS`

Scheduling Considerations

After rplext.pc and ibexpl.pc
Before rplbld.pc

Locking Strategy

This program will run during the nightly batch schedule.

This program will not update or delete records, it only inserts into IB_RESULTS.

Restart/Recovery

The logical unit of work is item, supplier, origin country, and location combination. The logical unit of work is non-unique.

Performance Considerations

This could potentially be effect a very large number if item/locations every night. Every effort should be make to enhance the performance of this module.

Security Considerations

The program will be run in the batch window when all security is turned off. This program needs full access to the system to operate correctly.

Unit Test Considerations**Design Assumptions**

Processing will only take place on warehouse locations.

Investment buy explosion/pre-quality [ibexpl]

Design Overview

The ibexpl batch program pre-qualifies investment buy (ib) legible wh/dept and IB legible supp/dept/locs.

The wh_dept table holds IB parameters at the WH or at the wh/dept level. If there are IB parameters defined at the wh/dept level, they are used. If there are not IB parameters defined at the wh/dept level, the IB parameters at the WH level are used. If IB parameters are not defined at either level, system level IB parameters (system_options) are used. The first part of this program blows IB parameters to the wh/dept level no matter what level they are held at in the database. The results are written to the wh_dept_expl table.

Next wh_dept_expl (with all legible wh/dept combos) is combined with sup_inv_mgmt to get the final list of all legible sup/dept/locs.

Sup_inv_mgmt determines whether or not a given sup/dept/loc combo is IB legible. The main problem is that this table can store information at different levels depending on the supplier's inv_mgmt_lvl -- SUPS.INV_MGMT_LVL (in parenthesis).

Sup (S)
 Sup/dept (D)
 Sup/loc (L)
 Sup/dept/loc (A)

If the record is not found at the level suggested by the SUPS.INV_MGMT_LVL, it needs to look up the hierarchy as shown below - up to the highest level (sup). If no record exists as the sup level, its not IB legible.

Sup
 sup/dept -> sup
 sup/loc -> sup
 sup/dept/loc -> sup/dept -> sup

The second part of this program explodes the sup_inv_mgmt table down to the sup/dept/loc level by filling in the implied rows. The exploded sup_inv_mgmt info is only done for IB legible wh/dept combos from wh_dept_expl. The results are placed on sim_expl.

Stored Procedures / Shared Modules (Maintainability)

Program Flow

Function Level Description

Main() – standard retek main.

Init() – initialize restart recovery.

Process() – first deactivate the indexes on WH_DEPT_EXPL and SIM_EXPL by calling unusable_indexes(). Next truncate the two tables by calling truncate_table(). Next repopulate the

two tables by calling `pop_wh_dept_expl()` and `pop_sim_expl()`. Finally rebuild to indexes on the two tables by calling `unusable_indexes()`.

Unusable_indexes() – this function will either set all indexes on a table to unusable or rebuild them depending on the passed in parameters.

Truncate_table() – this function will truncate the table that is passed into it as a parameter.

Pos_wh_dept_expl() – this function will populate the `wh_dept_expl` table by doing an insert-select statement based on the `wh`, `wh_dept`, `deps`, and `system_options` tables.

Populate a temp table with all the IB eligible warehouses and their IB parameters exploded to all departments in the system. IB parameters can be held at three levels -- WH/DEPT, WH, SYSTEM. The lowest level always takes precedence. This is accomplished by using three cursors joined together by a UNION:

The first union handles instances where IB parameters are set up at the WH/DEPT level.

The second union handles instances where IB parameters are set up at the WH but not at the WH/DEPT level.

The third union handles instances where IB parameters are set up at the SYSTEM but not WH/DEPT or WH level.

Pop_sim_expl() – this function will populate the `sim_expl` table by doing an insert-select statement on `sup_inv_mgmt`, `sup`, `terms`, and `wh_dept_expl`.

Populate a temp table with all the IB eligible WH/DEPT records from `WH_DEPT_EXPL` exploded to all IB eligible SUP/DEPT/LOCS in the system.

To do this we need to get all investment buy eligible suppliers from the `SUP_INV_MGMT` table. The problem is that this table can store information at different levels depending on the supplier's `inv_mgmt_lvl` -- `SUPS.INV_MGMT_LVL` (in parenthesis).

- sup (S)
- sup/dept (D)
- sup/loc (L)
- sup/dept/loc (A)

If the record is not found at the level suggested by the `SUPS.INV_MGMT_LVL`, it needs to look up the hierarchy as shown below - up to the highest level (sup). If no record exists as the sup level, it's not IB eligible.

- sup
- sup/dept -> sup
- sup/loc -> sup
- sup/dept/loc -> sup/dept -> sup

This is accomplished using an insert-select statement with four parts:

The first part gets records for supplier/dept/locs that do inventory management at the supplier level. It also gets supplier/dept/locs that do inventory management at lower

levels but have not actually set up inventory management parameters at those lower levels.

The second part gets records for supplier/dept/locs that do inventory management at the supplier/dept level. It also gets supplier/dept/locs that do inventory management at lower levels but have not actually set up inventory management parameters at those lower levels.

The third part gets records for supplier/dept/locs that do inventory management at the supplier/loc level and have records set up at supplier/loc.

The fourth part gets records for supplier/dept/locs that do inventory management at the supplier/dept/loc level and have records setup up at supplier/dept/loc.

final() – clean up restart recovery.

Input Specifications

'Table-To-Table'

wh_dept
sup_inv_mgmt

Output Specifications

'Table-To-Table'

wh_dept_expl
sim_expl

Scheduling Considerations

Before ibcalc.pc

Locking Strategy

This program will run during the nightly batch schedule.

Restart/Recovery

Only implemented for bookkeeping.

Performance Considerations

The main reason for this program to exist is for performance considerations. It prequalifies sup/dept/locs for IB consideration to allow this logic to be removed from ibcalc.pc's driving cursor.

This program requires special permissions. It must be run by an oracle user that has been granted the following privileges, or be run by a DBA.

- 'drop any table' AND
- 'alter any index\

Design Assumptions

Processing will only take place on warehouse locations.

Inventory adjustment purge [invaprg]

Design Overview

The Inventory Adjustment Purge (invaprg) module will delete all obsolete inventory adjustment records where a pre-determined number of months have elapsed. The pre-determined number of months is a system option.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
INV_ADJ	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
RESTART_CONTROL	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

In the init() function, select the number of months inventory adjustments are held is selected from the UNIT_OPTIONS table.

Then, in the process() function, all records are delete from the INV_ADJ table where the number of months between the current date and the adjustment date is greater than the number of months that inventory adjustments are held.

I/O Specification

N/A

Technical Issues

N/A

Close unmatched shipments [invclshp]

Design Overview

This batch program will close all shipments that have remained open for a specified number of days and are not associated with any open invoices. This will be accomplished by setting the invc_match_status on the SHIPMENT table to 'C'losed.

Scheduling Constraints

Processing Cycle: ad hoc

Scheduling Diagram: after invpost

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

For all shipments with an invc_match_status != 'C' on SHIPMENT, set the shipment's invc_match_status to 'C' if all of the following conditions is met:

- The shipment is in 'R' (received) status.
- The shipment's receive_date on SHIPMENT is system_options.close_open_ship_days before today's date.
- No invoice matched to the shipment is not in 'P' status.
- The shipment is not associated with any open (unposted) invoices on the INVC_XREF table.

Also, all shipments that are in 'I'nvoice status but are associated with closed orders will also be purged from the system.

Technical Issues

N/A

Invoice post [invcpost]

Design Overview

This batch program will insert records into the IIF (invoice interface to financials) staging tables. It will insert new invoices with a payment status of 'H' (hold payment) so that the AP system can take all current/future liabilities into consideration. It will also insert approved matched invoices or force-paid invoices with a payment status of 'P' (ready to pay). Invoices from suppliers that are marked as pre-paid suppliers will be inserted with a payment status of 'U' (pre-paid unmatched) if unapproved or 'M' (pre-paid matched) if approved.

Approved invoices associated with approved debit/credit memos or credit notes will be inserted with a payment status of 'P' (ready to pay) since issuing a debit/credit memo should release the invoice for payment. However, the invoices and the debit/credit memos or credit notes associated should all be in approved status to be posted in tandem. Invoices with attached credit note requests will only be posted when an approved merchandise credit note has been attached. Credit note requests are never sent to financials. All types of invoices (except credit note requests), including debit/credit memos or credit notes, will be sent to the IIF staging tables even if they are not qualified for being posted.

This program will be modified to also post INVC_DISCOUNT records to the IIF_DISCOUNT table when posting an invoice. Scheduling Constraints.

Processing Cycle: daily

Scheduling Diagram: after ediupinv and invmtch, before interface to financial system

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: None

Restart Recovery

The logical unit of work for the invoice post module will be the invoice transaction. Restart/recovery will be based on the invoice transaction. In order to bundle an invoice and its associates, the invoice transaction here means the invc_id itself for an invoice not referencing another invoice and the ref_invc_id for an invoice that references another invoice. Therefore, one logical unit of work can contain multiple invoices.

The driving cursor for this restart/recovery strategy looks like the following. Note that the second decode() of the ORDER BY clause is used to put an invoice ahead of its associates if it is referenced.

```
SELECT ih.invc_id,
       ih.invc_type,
       ih.supplier,
       ih.partner_type,
       ih.partner_id,
       ih.ext_ref_no,
       ih.ref_invc_id,
```

```

        ih.ref_rtv_order_no,
        ih.ref_price_change,
        ih.ref_rsn_code,
        ih.terms,
        TO_CHAR(ih.due_date, 'YYYYMMDD'),
        ih.payment_method,
        ih.terms_dscnt_pct,
        ih.terms_dscnt_appl_ind,
        ih.freight_terms,
        TO_CHAR(ih.invc_date, 'YYYYMMDD'),
        ih.force_pay_ind,
        ih.post_date,
        ih.currency_code,
        ih.exchange_rate,
        ih.total_merch_cost,
        ih.total_qty,
        ih.comments,
        ih.status,
        ih.direct_ind,
        ih.paid_ind,
        ih.addr_key,
        ih.payment_ref_no,
        ih.payment_date,
        ih.proof_of_delivery_no
FROM   invc_head ih
WHERE  ih.invc_type != 'R'
      AND (ih.status = 'A'
        OR (ih.force_pay_ind = 'Y'
          AND ih.status != 'P'))
      OR ih.post_date is NULL)
      AND decode(ih.ref_invc_id,
                NULL, ih.invc_id,
                ih.ref_invc_id) > NVL(:ps_restart_invc_id, -999
ORDER BY decode(ih.ref_invc_id, NULL, ih.invc_id, ih.ref_invc_id),
          decode(ih.ref_invc_id, NULL, 0, 1));

```

The `commit_max_ctr` field on the `restart_control` table will determine the number of transactions that equal a logical unit of work. It should be set to prevent excessive rollback space usage. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init():

Retrieve system date and VAT indicator from `PERIOD` and `SYSTEM_OPTIONS`:

- `vdate`
- `vat_ind`

process():

In a LOOP, fetch the driving cursor. For each invoice fetched, insert into arrays (for insert into the IIF tables) as follows:

- For the insert into IIF_HEAD, gather the following values for each invoice:
 - invc_id = invc_head.invc_id
 - invc_type = invc_head.invc_type
 - supplier = invc_head.supplier
 - partner_type = invc_head.partner_type
 - partner_id = invc_head.partner_id
 - ext_ref_no = invc_head.ext_ref_no
 - payment_status will be decided by the action taken on this invoice, invc_head.direct_ind, and the prepay indicator:
 - Action is POST if
 - The invoice is being force-paid (force_pay_ind = 'Y')
 - The invoice is in 'A' status and not associated with any other invoices
 - The invoice is in 'A' status, associated with other invoices and all its associates are ready (debit memos/credit memos/credit notes are all in 'A' status, and, if any credit note request exists, debit memo or credit note in 'A' status exist).
 - The invoice is in 'A' status and referencing an invoice in 'P' status (either just posted or previously posted).
 - Action is SEND if the invoice is not qualified for being POSTed but has not been sent yet
 - Action is NONE if the invoice is not qualified for being POSTed and has been sent already
 - The prepay indicator indicates if a supplier is marked for pre-payment. It is fetched from sups.prepay_invc_ind for a supplier invoice; it is always 'N' for a non-supplier invoice
 - Payment_status is 'C' if invc_head.paid_ind is 'Y' for the invoice
 - Payment_status is 'M' if the action is POST and prepay indicator is 'Y'
 - Payment_status is 'P' if the action is POST and prepay indicator is 'N'
 - Payment_status is 'U' if the action is SEND and prepay indicator is 'Y'
 - Payment_status is 'H' if the action is SEND and prepay indicator is 'N'
 - ref_invc_id = invc_head.ref_invc_id
 - ref_rtv_order_no = invc_head.ref_rtv_order_no
 - ref_price_change = invc_head.ref_price_change
 - ref_rsn_code = invc_head.ref_rsn_code
 - terms = invc_head.terms
 - due_date = invc_head.due_date
 - payment_method = invc_head.payment_method
 - terms_dscnt_pct = invc_head.terms_dscnt_pct
 - terms_dscnt_appl_ind = invc_head.terms_dscnt_appl_ind
 - freight_terms = invc_head.freight_terms
 - invc_date = invc_head.invc_date
 - force_pay_ind = invc_head.force_pay_ind
 - post_date = today's date
 - currency_code = invc_head.currency_code
 - exchange_rate = invc_head.exchange_rate
 - total_payment_merch_cost = invc_head.total_merch_cost
 - total_payment_qty = invc_head.total_qty
 - comments = invc_head.comments
 - addr_key = invc_head.addr_key

- `payment_ref_no = invc_head.payment_ref_no`
- `payment_date = invc_head.payment_date`
- `proof_of_delivery_no = invc_head.proof_of_delivery_no`
- `direct_ind = invc_head.direct_ind`
- For the insert into IIF_NON_MERCH, gather the following values for each invoice:
 - `invc_id = invc_non_merch.invc_id`
 - `non_merch_code = invc_non_merch.non_merch_code`
 - `non_merch_amt = invc_non_merch.non_merch_amt`
 - `vat_code = invc_non_merch.vat_code`
 - `service_perf_ind = invc_non_merch.service_perf_ind`
 - `store = invc_non_merch.store`
- For the insert into IIF_MERCH_VAT, gather the following values for each invoice:
 - `invc_id = invc_merch_vat.invc_id`
 - `vat_code = invc_merch_vat.vat_code`
 - `total_cost_excl_vat = invc_merch_vat.total_cost_excl_vat`
- For the insert into IIF_DETAIL, gather the following values for each invoice:
 - `invc_id = invc_detail.invc_id`
 - `item = invc_detail.item`
 - `payment_unit_cost = invc_detail.invc_unit_cost`
 - `payment_qty = invc_detail.invc_qty`
 - `payment_vat_rate = invc_detail.invc_vat_rate`
 - `cost_dscrpncty_ind = invc_detail.cost_dscrpncty_ind`
 - `qty_dscrpncty_ind = invc_detail.qty_dscrpncty_ind`
 - `vat_dscrpncty_ind = invc_detail.vat_dscrpncty_ind`

- For the insert into IIF_MATCH_DETAIL, gather the following values for each invoice (invoice types 'C', 'D' and 'M' will not have records on this table):
 - `invc_id = invc_match_wksht.invc_id`
 - `item = invc_match_wksht.item`
 - `order_no = the order_no from SHIPMENT for invc_match_wksht.shipment`
 - `asn_no = the asn from SHIPMENT for invc_match_wksht.shipment`
 - `shipment = invc_match_wksht.shipment`
 - `rcpt_date = the receive_date from SHIPMENT for invc_match_wksht.shipment`

For invoices matched at the totals level, where there are no `invc_match_wksht` records, the values should be gathered from `shipsku` instead.

- Once all IIF records are inserted into the array for the invoice, update the invoice and receipt header statuses appropriately.
 - Set the `post_date` on `INVC_HEAD` to today's date.
 - If the action taken for the invoice is POST, set the status on `INVC_HEAD` to 'P' (posted).
 - If the invoice was in 'A' status, check each shipment matched to the invoice (retrieve using `match_invc_id` on `SHIPSku`) to see if it is matched to any other invoices not in 'P' status. If it is, or if there are still items on the shipment that are unmatched

(match_inv_id is NULL), leave the shipment's inv_match_status as it is. If it is not, and all SKUs on the shipment are matched, set the shipment's inv_match_status to 'C'.

End LOOP.

Table to Table

Source Table	Source Column	Field Type
INVC_DISCOUNT	inv_id	number(10)
	seq_no	number(6)
	discount_type	varchar2(6)
	discount_value	number(20,4)
	applies_to_amt	number(20,4)

Target Table	Target Column	Calculations
IIF_DISCOUNT	inv_id	None
	seq_no	None
	discount_type	None
	discount_value	None
	applies_to_amt	None

process_inv()

- after the calls to insert_iif_non_merch() and insert_iif_merch_vat(), and before the call to insert_iif_detail(), make a call to the new function insert_iif_discount()

insert_iif_discount()

- perform a select insert to move all data from INVC_DISCOUNT to IIF_DISCOUNT for the invoice in the pa_inv_array current record

Technical Issues

N/A

Automatic invoice matching [invmatch]

Design Overview

This batch program will perform the automatic invoice matching process. For each invoice on the INVC_MATCH_QUEUE table, a summary match will be attempted followed by a detail-level match (if invoice details are available). Successfully matched invoices will be set to Matched status and automatically approved if from automatic approval suppliers. Invoices that fail automatic matching will have their match fail indicator set to 'Y'. On successful completion of this program, all records will be deleted from the INVC_MATCH_QUEUE table.

Scheduling Constraints

Processing Cycle: daily

Scheduling Diagram: after ediupinv, revupld, before invpost

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: supplier

Restart Recovery

The logical unit of work for the automatic invoice matching module will be the invoice transaction. Restart/recovery will be based on the invoice transaction. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

The commit_max_ctr field should be set to prevent excessive rollback space usage. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Program Flow

N/A

Shared Modules

INVC_MATCH_SQL.TOTAL_MATCH – to summary match a given invoice.

INVC_MATCH_SQL.ITEM_MATCH_ALL – to detail match a given invoice.

Driving Cursor:

```
SELECT imq.invc_id,
       ih.status
FROM   invc_head ih,
       invc_match_queue imq,
       v_restart_supplier vrs
WHERE  imq.invc_id = ih.invc_id
       AND ih.supplier = vrs.driver_value
       AND vrs.driver_name = :os_restart_driver_name
       AND vrs.num_threads = :oi_restart_num_threads
```

```

AND vrs.thread_val = :oi_restart_thread_val
AND vrs.driver_value = ih.supplier
AND (ih.invc_id > NVL(:os_restart_invc_id, -999))
ORDER BY ih.invc_id;

```

Function Level Description

init():

Restart/recovery initialization; size arrays for driving cursor to fetch into

process():

In a LOOP, fetch the driving cursor using an array. For each invoice fetched, attempt to match it as follows:

- Check the invoice's status on INVC_HEAD. If the status is not 'U' or 'R', the invoice is already matched. Continue processing with the next invoice (apart from the last step of deleting off INVC_MATCH_QUEUE).
- Call the function check_header_match. This will do the following: 1) call INVC_MATCH_SQL.TOTAL_MATCH to attempt a totals match of the invoice.
- 2) If O_reconciled is FALSE, the invoice cannot be matched. The match_fail_ind on INVC_HEAD should be updated to 'Y' and processing should continue on the next invoice (apart from the last step of deleting off INVC_MATCH_QUEUE).
- 3) If O_match is TRUE, the invoice is matched and processing can continue on the next invoice (apart from the last step of deleting off INVC_MATCH_QUEUE).
- 4) If O_match is FALSE, line item matching will be attempted if line item details exist on INVC_DETAIL. Call the function INVC_MATCH_SQL.ITEM_MATCH_ALL to attempt line item matches of the invoice.
- 5) If O_match is TRUE from ITEM_MATCH_ALL, the invoice is matched and processing can continue on the next invoice (apart from the last step of deleting off INVC_MATCH_QUEUE).
- 6) If O_match is FALSE from ITEM_MATCH_ALL, the invoice is not matched. The match_fail_ind on INVC_HEAD should be updated to 'Y'.
- Once an invoice is processed, delete its record off INVC_MATCH_QUEUE.

End LOOP.

check_header_match(ls_invc_id):

Call INVC_MATCH_SQL.TOTAL_MATCH; if reconciled but not matched, call INVC_MATCH_SQL.ITEM_MATCH_ALL.(see detailed description above).

final():

restart/recovery close

Technical Issues

N/A

Invoice purge [invprg]

Design Overview

This program will purge old (older than order_history_months) posted invoices that have not already been purged by ordprg.pc (which purges invoices associated with an order). This includes all types of invoices—non-merchandise, credit notes, credit note requests, debit memos, and consignment invoices. Regular merchandise invoices will primarily be deleted through ordprg.pc but will be deleted by invprg.pc if they still exist in the system. The age of the invoices will be determined from the match date; if there is no match date, the invoice date will be used.

01/24/2000—Modification of this program to accommodate for the Direct-to-Store functionality is being added. An added check will have to be made in the driving cursor to exclude invoice referenced to the SA_TRAN_HEAD table before deletion. For invoices that are still in the Sales Audit system and meet the following conditions should not be purged.

If INVC_HEAD.PAYMENT_DATE = SA_TRAN_HEAD.TRAN_DATETIME
 And (INVC_HEAD.EXT_REF_NO = SA_TRAN_HEAD.VENDOR_INVC_NO
 Or INVC_HEAD.EXT_REF_NO = SA_TRAN_HEAD.PAYMENT_REF_NO
 Or INVC_HEAD.EXT_REF_NO =
 SA_TRAN_HEAD.PROOF_OF_DELIVERY_NO).

12/20/2001 : Also, invoices that are matched to a shipment (via SHIPSKU.MATCH_INVC_ID) will not be purged.

Scheduling Constraints

Pre/Post Logic Description

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
INVC_HEAD	Yes	Yes	No	No	Yes
INVC_DETAIL	Yes	No	No	No	Yes
INVC_XREF	Yes	No	No	No	Yes
INVC_MATCH_QUEUE	Yes	No	No	No	Yes
INVC_NON_MERCH	Yes	No	No	No	Yes
INVC_MERCH_VAT	Yes	No	No	No	Yes
INVC_MATCH_WKSHT	Yes	No	No	No	Yes
SHIPSKU	No	Yes	No	No	No

Processing Cycle: Ad hoc (monthly)

Scheduling Diagram: This program should run after ordprg.pc (Invoices that are matched to an order are purged by ordprg.pc)

Pre-Processing: ordprg.pc

Post-Processing: N/A

Threading Scheme: NONE

Restart Recovery

The logical unit of work is the invoice. A commit should occur every 2500 records (subject to experimentation)

Driving cursor:

```
SELECT invc_id
  FROM invc_head
 WHERE status = 'P'
    AND MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),NVL(match_date,
invc_date))
        >:oi_hist_months;
```

(implicit restartability—ones you've done will have been deleted, so won't get fetched by the cursor when restarted).

Modify driving cursor (01/24/2000):

```
SELECT invc_id
  FROM invc_head
 WHERE status = 'P'
    AND MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),NVL(match_date,
invc_date))
        >:oi_hist_months
    AND not exists (SELECT 'x'
                    FROM sa_tran_head s
                   WHERE i.payment_date = to_date (to_char (s.tran_datetime, 'YYYYMMDD'),
'YYYYMMDD')
                  AND (i.ext_ref_no = s.vendor_inc_no
                      OR i.ext_ref_no = s.payment_ref_no
                      OR i.ext_ref_no = s.proof_of_delivery_no))
 ORDER BY i.invc_id;
```

Modify driving cursor (12/20/2001):

```
SELECT i.invc_id
  FROM invc_head i
 WHERE i.status = 'P'
    AND MONTHS_BETWEEN(TO_DATE(:ps_vdate, 'YYYYMMDD'), NVL(i.match_date,
i.invc_date)) > :pi_hist_months
    AND NOT EXISTS (SELECT 'x'
                    FROM sa_tran_head s,
                         shipsku k
                   WHERE i.payment_date = TO_DATE(TO_CHAR(s.tran_datetime,
'YYYYMMDD'), 'YYYYMMDD')
                  AND (i.ext_ref_no = s.vendor_inv_no
                      OR i.ext_ref_no = s.payment_ref_no
                      OR i.ext_ref_no = s.proof_of_delivery_no)
                  AND (k.match_inv_id = i.invc_id))
 ORDER BY i.invc_id;
```

Modify driving cursor (2/28/2002):

```
SELECT i.invc_id
  FROM invc_head i
 WHERE i.status = 'P'
```

```

        AND MONTHS_BETWEEN(TO_DATE(:ps_vdate, 'YYYYMMDD'), NVL(i.match_date,
i.invc_date)) > :pi_hist_months
        AND NOT EXISTS (SELECT 'x'
                        FROM sa_tran_head s
                        WHERE i.payment_date = TO_DATE(TO_CHAR(s.tran_datetime,
'YYYYMMDD'), 'YYYYMMDD')
                        AND (i.ext_ref_no = s.vendor_inv_no
                            OR i.ext_ref_no = s.payment_ref_no
                            OR i.ext_ref_no = s.proof_of_delivery_no))
        AND NOT EXISTS (SELECT 'x'
                        FROM shipsku k
                        WHERE k.match_inv_id = i.invc_id)
    ORDER BY i.invc_id;

```

Program Flow

Shared Modules

INVC_SQL.DELETE_INVC—for a given invoice, delete from all associated invoice tables

Function Level Description

init()

Fetch vdate from period and order_history_months from unit_options.
 Restart/recovery initialization.
 Call size_array to set up memory for the array the driving cursor will fetch into.

process()

In a loop:

Array fetch driving cursor (get invoices to delete)
 Call delete_invoices to do actual deletion for each invoice in the array.
 Call restart_commit after the entire array has been processed, making sure to force a commit.

delete_invoices()

Call the invc_sql.delete_inv function to delete invoices. The SKU parameter should be passed in as NULL so that the entire invoice will be deleted.

size_array()

Allocate memory for invoice id array.

final()

Restart/recovery close.

I/O Specification

N/A

Letter of credit download [lcadnld]

Design Overview

The lcadnld program is used to process Letter of Credit (LC) applications, in order to be sent to the bank, based on the letter of credits flagged during the day by the Select LC Applications action on the LC Find form and written to the staging table LC_DOWNLOAD. This program has one output file which contains all records for all LC applications. This program should be included in the standard download phase of the batch cycle. The individual LCs flagged for a given day will be downloaded in full, with application and detail information written to a Retek standardized flat file. Another program will format the information from this file into individualized bank flat files using a S.W.I.F.T. format (SWIFT formats are used when transmitting financial information and are a globally recognized set of standards). After this program has processed all of the applications on LC_DOWNLOAD, these records will be deleted and the LCs will have their status changed to 'E'xtracted on LC_HEAD.

Affected Tables:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
LC_HEAD	Yes	Yes	No	Yes	No
LC_DETAILS	Yes	Yes	No	No	No
LC_DOWNLOAD	No	Yes	No	No	Yes
SUP_IMPORT_ATTR	No	Yes	No	No	No
ADDR	Yes	Yes	No	No	No
PARTNER	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
CODE_DETAIL	Yes	Yes	No	No	No
ITEM_MASTER	Yes	Yes	No	No	No
DOC	No	Yes	No	No	No
OUTLOC	Yes	Yes	No	No	No
ITEM_IMPORT_ATTR	No	Yes	No	No	No
REQ_DOC	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Perl script lcmt700.

Threading Scheme: No threading due to low volume.

Driving Cursor- The cursor includes three parts. The first part joins to the item_maste table and selects the item_parent (if the item's level is 2) or item_grandparent (if the item's level is 3) where the item_master.item is equal to lc_detail.item, a group by is used to sum the qty. The

second part joins item_master and selects the item where item_master.item is equal to lc_detail.item or item is NULL and the item's level is one and it's a transaction level item. The third part joins item_master and selects the item where the item_master.item is equal to lc_detail.item and the pack_ind is 'Y' (a pack item).

```
SELECT ROWIDTOCHAR(h.rowid),
       ROWIDTOCHAR(l.rowid),
       h.issuing_bank,
       NVL(h.advising_bank, ' '),
       h.lc_ref_id,
       NVL(h.bank_lc_id, ' '),
       h.form_type,
       h.lc_type,
       s.revocable_ind,
       h.transferable_ind,
       TO_CHAR(h.application_date, 'YYYYMMDDHH24MISS'),
       TO_CHAR(h.expiration_date, 'YYYYMMDDHH24MISS'),
       h.place_of_expiry,
       h.applicant,
       h.beneficiary,
       h.currency_code,
       h.exchange_rate * :ld_rate_dec,
       NVL(h.origin_country_id, ' '),
       h.presentation_terms,
       h.purchase_type,
       NVL(h.advice_method, ' '),
       NVL(h.issuance, ' '),
       h.amount_type,
       h.amount * :ll_no_dec,
       h.variance_pct * :ll_no_dec,
       NVL(h.specification, ' '),
       NVL(h.credit_avail_with, ' '),
       NVL(h.drafts_at, ' '),
       NVL(h.paying_bank, ' '),
       NVL(h.negotiating_bank, ' '),
       NVL(h.confirming_bank, ' '),
       NVL(h.transferring_bank, ' '),
       h.partial_shipment_ind,
       h.transshipment_ind,
       NVL(h.fob_title_pass, ' '),
       NVL(h.fob_title_pass_desc, ' '),
       h.with_recourse_ind,
       TO_CHAR(h.latest_ship_date, 'YYYYMMDDHH24MISS'),
       TO_CHAR(h.earliest_ship_date, 'YYYYMMDDHH24MISS'),
       NVL(TO_CHAR(h.lc_neg_days), ' '),
       NVL(h.transport_to, ' '),
       ol.outloc_desc,
       h.comments,
       d.order_no,
       DECODE(im.item_level, 2, im.item_parent,
```



```

        3, im.item_grandparent),
    MAX(NVL(d.cost, 0) * :ll_no_dec),
    SUM(NVL(d.qty, 0) * :ll_no_dec),
    MIN(NVL(TO_CHAR(d.earliest_ship_date, 'YYYYMMDDHH24MISS'), ' ')),
    MAX(NVL(TO_CHAR(d.latest_ship_date, 'YYYYMMDDHH24MISS'), ' ')),
    d.merch_desc,
    d.comments,
    im.standard_uom,
    im.desc_up
FROM lc_head h,
    lc_detail d,
    sup_import_attr s,
    lc_download l,
    outloc ol,
    item_master im
WHERE h.lc_ref_id = l.lc_ref_id
    AND d.lc_ref_id = l.lc_ref_id
    AND s.supplier = h.beneficiary
    AND im.item = d.item
    AND ol.outloc_id = h.transport_to
    AND l.lc_ref_id > nvl(:ps_restart_lc, 0)
    AND l.lc_download_type = 'A',
    AND im.item_level = im.tran_level
    AND im.item_level > 1
    AND im.pack_ind = 'N'
GROUP BY h.rowid,
    l.rowid,
    h.issuing_bank,
    h.advising_bank,
    h.lc_ref_id,
    h.bank_lc_id,
    h.form_type,
    h.lc_type,
    s.revocable_ind,
    h.transferable_ind,
    h.application_date,
    h.expiration_date,
    h.place_of_expiry,
    h.applicant,
    h.beneficiary,
    h.currency_code,
    h.exchange_rate,
    h.origin_country_id,
    h.presentation_terms,
    h.purchase_type,
    h.advice_method,
    h.issuance,
    h.amount_type,
    h.amount,
    h.variance_pct,
    h.specification,

```

```

h.credit_avail_with,
h.drafts_at,
h.paying_bank,
h.negotiating_bank,
h.confirming_bank,
h.transferring_bank,
h.partial_shipment_ind,
h.transshipment_ind,
h.fob_title_pass,
h.fob_title_pass_desc,
h.with_recourse_ind,
h.latest_ship_date,
h.earliest_ship_date,
h.lc_neg_days,
h.transport_to,
ol.outloc_desc,
h.comments,
d.order_no,
DECODE(im.item_level, 2, item_parent,
        3, item_grandparent),
d.merch_desc,
d.comments,
rst.standard_uom,
dl.desc_up
UNION ALL
SELECT ROWIDTOCHAR(h2.rowid),
ROWIDTOCHAR(l2.rowid),
h2.issuing_bank,
NVL(h2.advising_bank, ' '),
h2.lc_ref_id,
NVL(h2.bank_lc_id, ' '),
h2.form_type,
h2.lc_type,
s2.revocable_ind,
h2.transferable_ind,
TO_CHAR(h2.application_date, 'YYYYMMDDHH24MISS'),
TO_CHAR(h2.expiration_date, 'YYYYMMDDHH24MISS'),
h2.place_of_expiry,
h2.applicant,
h2.beneficiary,
h2.currency_code,
h2.exchange_rate * :ld_rate_dec,
NVL(h2.origin_country_id, ' '),
h2.presentation_terms,
h2.purchase_type,
NVL(h2.advice_method, ' '),
NVL(h2.issuance, ' '),
h2.amount_type,
h2.amount * :ll_no_dec,
h2.variance_pct * :ll_no_dec,
NVL(h2.specification, ' '),

```

```

        NVL(h2.credit_avail_with, ' '),
        NVL(h2.drafts_at, ' '),
        NVL(h2.paying_bank, ' '),
        NVL(h2.negotiating_bank, ' '),
        NVL(h2.confirming_bank, ' '),
        NVL(h2.transferring_bank, ' '),
        h2.partial_shipment_ind,
        h2.transshipment_ind,
        NVL(h2.fob_title_pass, ' '),
        NVL(h2.fob_title_pass_desc, ' '),
        h2.with_recourse_ind,
        TO_CHAR(h2.latest_ship_date, 'YYYYMMDDHH24MISS'),
        TO_CHAR(h2.earliest_ship_date, 'YYYYMMDDHH24MISS'),
        NVL(TO_CHAR(h2.lc_neg_days), ' '),
        NVL(h2.transport_to, ' '),
        ol2.outloc_desc,
        h2.comments,
        d2.order_no,
        TO_CHAR(d2.item),
        NVL(d2.cost, 0) * :ll_no_dec,
        NVL(d2.qty, 0) * :ll_no_dec,
        NVL(TO_CHAR(d2.earliest_ship_date, 'YYYYMMDDHH24MISS'), ' '),
        NVL(TO_CHAR(d2.latest_ship_date, 'YYYYMMDDHH24MISS'), ' '),
        d2.merch_desc,
        d2.comments,
        im2.standard_uom,
        im2.desc_up
FROM lc_head h2,
     lc_detail d2,
     sup_import_attr s2,
     lc_download l2,
     outloc ol2,
     item_master im2
WHERE h2.lc_ref_id = l2.lc_ref_id
      AND im.pack_ind = 'N'
      AND d2.lc_ref_id = l2.lc_ref_id
      AND s2.supplier = h2.beneficiary
      AND (im2.item = d2.item
           OR (d2.item is NULL
               AND im2.item = (select MIN(item) from item_master)))
      AND ol2.outloc_id = h2.transport_to
      AND l2.lc_ref_id > nvl(:ps_restart_lc, 0)
      AND l2.lc_download_type = 'A'
      AND ((im2.item = d2.item AND d2.item is NOT NULL)
           AND im2.item_level = im2.tran_level
           AND im2.item_level = 1
UNION ALL
SELECT ROWIDTOCHAR(h3.rowid),
       ROWIDTOCHAR(l3.rowid),
       h3.issuing_bank,
       NVL(h3.advising_bank, ' '),

```

```

h3.lc_ref_id,
NVL(h3.bank_lc_id, ''),
h3.form_type,
h3.lc_type,
s3.revocable_ind,
h3.transferable_ind,
TO_CHAR(h3.application_date, 'YYYYMMDDHH24MISS'),
TO_CHAR(h3.expiration_date, 'YYYYMMDDHH24MISS'),
h3.place_of_expiry,
h3.applicant,
h3.beneficiary,
h3.currency_code,
h3.exchange_rate * :ld_rate_dec,
NVL(h3.origin_country_id, ''),
h3.presentation_terms,
h3.purchase_type,
NVL(h3.advice_method, ''),
NVL(h3.issuance, ''),
h3.amount_type,
h3.amount * :ll_no_dec,
h3.variance_pct * :ll_no_dec,
NVL(h3.specification, ''),
NVL(h3.credit_avail_with, ''),
NVL(h3.drafts_at, ''),
NVL(h3.paying_bank, ''),
NVL(h3.negotiating_bank, ''),
NVL(h3.confirming_bank, ''),
NVL(h3.transferring_bank, ''),
h3.partial_shipment_ind,
h3.transshipment_ind,
NVL(h3.fob_title_pass, ''),
NVL(h3.fob_title_pass_desc, ''),
h3.with_recourse_ind,
TO_CHAR(h3.latest_ship_date, 'YYYYMMDDHH24MISS'),
TO_CHAR(h3.earliest_ship_date, 'YYYYMMDDHH24MISS'),
NVL(TO_CHAR(h3.lc_neg_days), ''),
NVL(h3.transport_to, ''),
ol3.outloc_desc,
h3.comments,
d3.order_no,
TO_CHAR(d3.item),
NVL(d3.cost, 0) * :ll_no_dec,
NVL(d3.qty, 0) * :ll_no_dec,
NVL(TO_CHAR(d3.earliest_ship_date, 'YYYYMMDDHH24MISS'), ''),
NVL(TO_CHAR(d3.latest_ship_date, 'YYYYMMDDHH24MISS'), ''),
d3.merch_desc,
d3.comments,
im.standard_uom,
im3.desc_up
FROM lc_head h3,
lc_detail d3,

```

```

sup_import_attr s3,
lc_download l3,
outloc ol3,
item_master im3
WHERE h3.lc_ref_id = l3.lc_ref_id
AND im3.pack_ind = 'Y'
AND d3.lc_ref_id = l3.lc_ref_id
AND s3.supplier = h3.beneficiary
AND ol3.outloc_id = h3.transport_to
AND l3.lc_ref_id > nvl(:ps_restart_lc,0)
AND l3.lc_download_type = 'A'
AND (im3.item = d3.item AND d3.item is NOT NULL)
ORDER BY 3, 5, 44;

```

Restart Recovery

Restart/recovery for this program is set up at the lc_ref_id level.

The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Program Flow



Shared Modules

LANGUAGE_SQL.GET_CODE_DESC

Function Level Description

Init():

This function performs preliminary processing and populates global variables. It will call retek_init() to start up restart-recovery and bring back the bookmark string in case of a restart. It

will also populate vdate, to be used when writing to the output file. Then call write_fhead() to open the output file and write the FHEAD information.

Process():

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from lc_download where the lc_download.lc_ref_lc_id value is greater than zero and lc_download.lc_download_type='A'. (The cursor includes two parts, the first part join the item_master and selects the item_parent or item_grandparent which item_master.item=lc_detail.item, the second part joins the item_master and select the item which item_master.item=lc_detail.item and lc_detail.item is null.)

Once the records are fetched, it sets the variance_pct field = ' ' if the amount_type is 'E'xact. It then calls write_thead() for every lc_ref_id. If lc_head.comments has been populated, then call write_thdcm(), to write the comments field to the output file. Then pass in the lc_ref_id to write_tdocs() to write a line for each description of goods and services, required document or additional condition associated with the LC. Then go on to call write_tdetl(), passing in the application_struct. If the lc_detail.merch_desc field has been populated, then call write_tmerc(), to write the merchandise description field to the output file. If the lc_detail.comments field has been populated, then call write_tdcm(), to write the comments field to the output file. Finally, call write_ttail() if the lc_ref_id has changed (to ensure that the transaction tail is only called when all the driving cursor records for an individual LC have been processed). For every loop call update_lc_head() to update all the statuses on lc_head table and call delete_lc_download() to delete all the records on lc_download table. Restart/Recovery and committing of records is also performed here.

Final():

This function will call write_ftail() first to write the FTAIL information, then call retek_close() to wrap up restart-recovery and populate the final error message, if necessary.

Write_fhead():

This function will write the FHEAD line of the output file.

Write_thead():

This function, which takes in the application_struct, will write the THEAD line of the output file according to the layout presented later in this document. Most of the variables will be fetch from the driving cursor, and can be written to the file as they are, the others will call get_all_desc() and get_all_address().

Write_thdcm():

This function, which takes in the lc_head.comments, will write the THDCM line of the output file. Note: if the lc_head.comments field is not populated, then this function will not be called.

Write_tdocs():

This function, which will take in an individual lc_ref_id, will do a join between the REQ_DOC and the DOC table where the REQ_DOC.DOC_ID = DOC.DOC_ID, the REQ_DOC.MODULE = 'LC', and the REQ_DOC.KEY_VALUE_1 = lc_ref_id. Fetch back the swift_tag, the doc_id and doc_text in a loop, writing an individual TDOC record for each doc_text fetched back. To explain further, each required document associated with an LC (and there can be more than one) will also have a swift_tag associated with it. This swift_tag is related to internationally recognized file format layouts that are used for financial transactions, often between banks. This program downloads the swift tag because it corresponds to fields (ex. field 46A – Required

Documents on form MT 700) on the SWIFT formatted file and will later be used by the scripts that translate this file into the bank required SWIFT format.

Write_tdetl():

This function, which takes in the `application_struct`, will write the TDETL line of the output file according to the layout presented later in this document.

Write_tmmerc():

This function, which takes in the `lc_detail.merch_desc`, will write the TMERC line of the output file. If the `lc_detail.merch_desc` field is not populated, then this function will not be called.

Write_tdtcm():

This function, which takes in the `lc_detail.comments`, will write the THDCM line of the output file. If the `lc_detail.comments` field is not populated, then this function will not be called.

Write_ttail():

This function will write the TTAIL line of the output file.

Write_ftail():

This function will write the FTAIL line of the output file.

Cerate_format_string():

This function formats the strings for the FHEAD, THEAD, THDCM, TDOCS, TDETL, TMERC, TDTCM, TTAIL and FTAIL records in the output file.

Init_arrays():

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

Get_all_desc():

This function will call `get_desc()`, passing a code and a code_type and will get the description for that code.

Form_type	LCFT
Lc_type	LCTP
Place_of_expiry	LCPE
Presenattion_terms	LCPT
Purchase_type	PURT
Advice_method	LCAD
Issuance	LCIS
Amount_type	LCAT
Specification	LCSP
Drafts_at	LCDA
Fob_title_pass	FOBT

Get_desc():

This function takes in a code and a code_type and fetches the description for that code using `LANGUAGE_SQL.GET_CODE_DESC` and passes the returned `O_code_desc` value out of the function.

Get_all_addresses():

This function will call get_address(), taking in Issuing_bank, Advising_bank, Applicant, Beneficiary, Credit_avail_with, Paying_bank, Negotiating_bank, Confirming_bank, Transferring_bank and will fetch back name and address information.

For Bank and Applicant, PARTNER.PARTNER_DESC should be fetched back where the partner_type = 'BK' and the partner_id = the bank (for the banks or the credit_avail_with) or where the partner_type = 'AP' and the partner_id = applicant (for the applicant). Then all the address information (add_1, add_2, add_3, city, state, post, country_id) should be fetched off the addr table. Again, fetch this information using the module = 'PTNR', key_value_1 = 'BK' (for the banks or credit_avail_with) or 'AP' (for the Applicant), and key_value_2 = issuing_bank ID or the applicant ID.

For the Beneficiary, first the supplier name should be fetched back from the SUPS table where Beneficiary = Supplier. Then, off the ADDR table the address information (add_1, add_2, add_3, city, state, post, country_id) should be fetched where the module = 'SUPP' and key_value_1 = beneficiary ID (key_value_2 is NULL).

Get_address():

This function will be called by get_all_addresses() and will return the information of name and address.

Update_lc_head():

This function will updates the status of the Letter of Credits to 'E'xtracted once they have been successfully processed. This is by rowid.

Delete_lc_download():

This function will deletes all records from LC_DOWNLOAD that have been fully processed. Done by rowid.

I/O SpecificationOutput file

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

It should be noted that the TDOCS lines are in no way associated with the TDETL lines. The TDETL lines are details relating to an individual LC and the TDOCS lines are documents associated with an individual LC.

The THDCM line is only present when the comment field at the header level on LC_HEAD is populated. There can be only one THDCM line per THEAD line.

The TDTCM line is only present when the comment field at the detail level on LC_DETAILS is populated. There can be only one TDTCM line per TDETL line.

Record Name	Field Name	Field Type	Default Value	Description
File	File Type	Char(5)	FHEAD	Identifies file record type

Header	Record Descriptor			
	File Line Identifier	Number(10)	line number in file	ID of current line being created for output file.
	File Type Definition	Char(4)	LCAP	Identifies file as 'Letter of Credit Application'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'
File Detail	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file.
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Issuing Bank	Char(10)	lc_head.issuing_bank	Used to sort the LCs into individualized bank SWIFT formatted files (using another program) – bank where LC application is headed.
	Issuing Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = issuing_bank and partner_type = 'BK'.
	Issuing Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Issuing Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Issuing Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Issuing Bank City	Char(20)	addr.city	City bank located in
	Issuing Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Issuing Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Issuing Bank Country	Char(3)	addr.country_id	Country bank located in.
	Advising Bank	Char(10)	lc_head.advising_bank	Used to sort the LCs into individualized bank SWIFT formatted files (using another program) – bank where LC application is headed.
	Advising Bank Name	Char(40)	Partner.partner_desc	The description from the partner table where partner_id = advising_bank and partner_type = 'BK'.
	Advising Bank	Char(30)	Addr.add_1	Mandatory line of address.

	Address 1			
	Advising Bank Address 2	Char(30)	Addr.add_2	Non-mandatory line of address (can be null).
	Advising Bank Address 3	Char(30)	Addr.add_3	Non-mandatory line of address (can be null).
	Advising Bank City	Char(20)	Addr.city	City bank located in
	Advising Bank State	Char(3)	Addr.state	State, if applicable, where bank located in.
	Advising Bank Post Code	Char(10)	Addr.post	Post code, if applicable, where bank located in.
	Advising Bank Country	Char(3)	Addr.country_id	Country bank located in.
	Letter of Credit	Number(8)	lc_head.lc_ref_id	The LC_REF_ID off the LC_HEAD table.
	Form Type	Char(6)	lc_head.form_type	The level of detail that the LC will send to the issuing bank.
	Form Type Description	Char(40)	code_detail.code_desc	Describes the form type: Long or Short.
	Letter of Credit Type	Char(6)	lc_head.lc_type	The type of LC that is being applied for.
	Letter of Credit Type Description	Char(40)	code_detail.code_desc	Describes the LC type: Master, Normal, Revolving.
	Form of Letter of Credit – I	Char(1)	sup_import_attr.revocable_ind	The REVOCABLE_IND off the SUP_IMPORT_ATTR table.
	Form of Letter of Credit – II	Char(1)	lc_head.transferable_ind	Indicates if LC transferable.
	Application Date	Char(14)	lc_head.application_date	Date the LC is created within RTM/RMS, formatted to 'YYYYMMDDHH24MISS'.
	Expiration Date	Char(14)	lc_head.expiration_date	The date the LC expires, formatted to 'YYYYMMDDHH24MISS'.
	Place of Expiry	Char(6)	lc_head.place_of_expiry	Code for the place the LC will expire.
	Place of Expiry Description	Char(40)	desc is retrieved through a decode	The description of the place the LC will expire.
	Applicant	Char(10)	lc_head.applicant	Party on whose behalf the LC is being issued.
	Applicant Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = applicant and partner_type = 'AP'.
	Applicant Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Applicant	Char(30)	addr.add_2	Non-mandatory line of address

	Address 2			(can be null).
	Applicant Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Applicant City	Char(20)	addr.city	City applicant located in
	Applicant State	Char(3)	addr.state	State, if applicable, where applicant located in.
	Applicant Post Code	Char(10)	addr.post	Post code, if applicable, where applicant located in.
	Applicant Country	Char(3)	addr.country_id	Country applicant located in.
	Beneficiary	Number(10)	lc.head.beneficiary	Party in favor of which the LC is being issued.
	Beneficiary Name	Char(32)	sup.supp_name	Beneficiary (supplier) name from the SUPS table.
	Beneficiary Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Beneficiary Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Beneficiary Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Beneficiary City	Char(20)	addr.city	City beneficiary located in
	Beneficiary State	Char(3)	addr.state	State, if applicable, where beneficiary located in.
	Beneficiary Post Code	Char(10)	addr.post	Post code, if applicable, where beneficiary located in.
	Beneficiary Country	Char(3)	addr.country_id	Country beneficiary located in.
	Currency Code	Char(3)	lc_head.currency_code	The country of origin for the orders on the LC.
	Exchange Rate	Number (20,10)	lc_head.exchange_rate	Exchange_rate to convert LC currency to RMS currency.
	Origin Country ID	Char(3)	lc_head.origin_country_id	Origin country of the orders associated with the LC.
	Presentation Terms	Char(6)	lc_head.presentation_terms	Code for the terms of presentation.
	Presentation Terms Description	Char(40)	desc is retrieved through a decode	Description of the terms of presentation.
	Purchase Type	Char(6)	lc_head.purchase_type	Code for the purchase type.
	Purchase Type Description	Char(40)	desc is retrieved through a decode	Description of the purchase type.
	Advice Method	Char(6)	lc_head.advice_method	Code for the advice method.
	Advice Method	Char(40)	desc is	Description of the advice method

	Description		retrieved through a decode	(eg. Full Wire, Mail, etc).
	Issuance	Char(6)	lc_head.issuance	Code for the issuance.
	Issuance Description	Char(40)	desc is retrieved through a decode	Description of the issuance (eg. Cable, Telex, etc).
	Amount Type	Char(6)	lc_head.amount_type	If 'E'xact, then amount must be exact, if 'A'pproximate then amount can be within variance percent.
	Amount Type Description	Char(40)	desc is retrieved through a decode	Description of amount_type.
	Amount	Number (20,4)	lc_head.amount	The total amt of the Letter of Credit.
	Variance Percent	Number (12,4)	lc_head.variance_pct	Allowed currency variance percent for the LC.
	Specification	Char(6)	lc_head.specification	Code for any condition for the credit, e.g. "maximum", etc.
	Specification Description	Char(40)	desc is retrieved through a decode	Description of condition for the credit, e.g. "maximum", etc.
	Credit Available With	Char(10)	lc_head.credit_avail_with	Code for bank with which credit is available.
	Credit With Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = credit_avail_with and partner_type = 'BK'.
	Credit With Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Credit With Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Credit With Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Credit With City	Char(20)	addr.city	City creditor located in
	Credit With State	Char(3)	addr.state	State, if applicable, where creditor located in.
	Credit With Post Code	Char(10)	addr.post	Post code, if applicable, where creditor located in.
	Credit With Country	Char(3)	addr.country_id	Country creditor located in.
	Drafts At	Char(6)	lc_head.drafts_at	Specifies the terms of the drafts to be drawn under the LC.

	Drafts At Description	Char(40)	desc is retrieved through a decode	Description of the terms of the drafts to be drawn under the LC.
	Drawee	Char(10)	lc_head.paying_bank	Identifies drawee of drafts to be drawn under LC (paying bank)
	Drawee Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = paying_bank and partner_type = 'BK'.
	Drawee Address 1	Char(30)	addr.addr_1	Mandatory line of address.
	Drawee Address 2	Char(30)	addr.addr_2	Non-mandatory line of address (can be null).
	Drawee Address 3	Char(30)	addr.addr_3	Non-mandatory line of address (can be null).
	Drawee City	Char(20)	addr.city	City bank located in
	Drawee State	Char(3)	addr.state	State, if applicable, where bank located in.
	Drawee Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Drawee Country	Char(3)	addr.country_id	Country bank located in.
	Negotiating Bank	Char(10)	lc_head.negotiating_bank	Identifies the negotiating bank.
	Negotiating Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = negotiating_bank and partner_type = 'BK'.
	Negotiating Bank Address 1	Char(30)	addr.addr_1	Mandatory line of address.
	Negotiating Bank Address 2	Char(30)	addr.addr_2	Non-mandatory line of address (can be null).
	Negotiating Bank Address 3	Char(30)	addr.addr_3	Non-mandatory line of address (can be null).
	Negotiating Bank City	Char(20)	addr.city	City bank located in
	Negotiating Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Negotiating Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Negotiating Bank Country	Char(3)	addr.country_id	Country bank located in.
	Confirming Bank	Char(10)	lc_head.confirming_bank	Identifies the confirming bank.
	Confirming Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = confirming_bank and partner_type = 'BK'.

	Confirming Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Confirming Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Confirming Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Confirming Bank City	Char(20)	addr.city	City bank located in
	Confirming Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Confirming Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Confirming Bank Country	Char(3)	addr.country_id	Country bank located in.
	Transferring Bank	Char(10)	lc_head.transferring_bank	Identifies the transferring bank.
	Transferring Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = transferring_bank and partner_type = 'BK'.
	Transferring Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Transferring Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Transferring Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Transferring Bank City	Char(20)	addr.city	City bank located in
	Transferring Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Transferring Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Transferring Bank Country	Char(3)	addr.country_id	Country bank located in.
	Partial Shipment Indicator	Char(1)	lc_head.partial_ship_ind	Indicates whether goods covered by LC can be partially shipped or not.
	Transshipment Indicator	Char(1)	lc_head.transshipment_ind	Indicates whether goods can be transferred to another vessel midway through the voyage.
	Fob Title Pass	Char(6)	lc_head.fob_title_pass	Indicates where the title for goods is passed from the vendor to the purchaser.
	Fob Title Pass Decode	Char(40)	desc is retrieved through a decode	Decode of where the title for goods is passed from the vendor to the purchaser.
	Fob Title Pass Description	Char(45)	lc_head.ob_title_pass_desc	Describes the FOB TITLE PASS – could be

				city name etc.
	Transportation to	Char(5)	lc_head.trans portation_to	Transportation to location
	transportation to description	Char(40)	outloc.outloc _desc	Description of transportation to lo
	With Recourse Indicator	Char(1)	lc_head.with _recourse_in d	Indicates conditional payment on the part of the bank as instructed by the buyer.
	Latest Shipment Date	Char(14)	lc_head.latest _ship_date	Latest ship date for all Pos included in the LC, formatted to 'YYYYMMDDHH24MISS'
	Earliest Shipment Date	Char(14)	lc_head.earli est_ship_date	Earliest ship date for all Pos included in the LC, formatted to 'YYYYMMDDHH24MISS'
	Letter of Credit Negotiation Days	Number(3) replaces x in the string "DOCUMENTS TO BE PRESENTED WITHIN x DAYS AFTER ISSUANCE OF THE SHIPPING DOCUMENTS BUT WITHIN THE VALIDITY OF THIS CREDIT"	lc.head.lc_ne g_days	The number of days to negotiate documents
	Bank's LC reference id	Number(8)	lc_head.bank _lc_id	Bank's LC ref id
File Detail	File Type Record Descriptor	Char(5)	THDCM	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Header Level Comments	Char(2000)	lc_head.com ments	Holds any comments that the user has added to the Letter of Credit.
File Detail	File Type Record Descriptor	Char(5)	TDOCS	Identifies file record type
	File Line Sequence	Number(10)	line number in file	ID of current line being created for output file

	Number			
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Swift Tag	Char(6)	doc.swift_tag	Identifies individual document types that can be associated with an LC.
	Document ID	Number(6)	req_doc.doc_id	Uniquely identifies the individual documents associated with an LC.
	Body Text	Char(2000)	req_doc.doc_text	Documents associated with a given LC. Description of Goods and Services OR Documents Required OR Additional Conditions OR Narrative
File Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Order Number	Number(8)	lc_detail.order_no	PO associated with the LC.
	Item	Char(25)	lc_detail.item	Item on the PO – item is rolled up to the item level of 1, if possible.
	Cost	Number (20,4)	lc_detail.cost	If form_type = 'S'hort then cost is the total cost of the order; if the form_type = 'L'ong then the cost is the unit cost of the item.
	Quantity	Number (12,4)	lc_detail.qty	Total qty of the item for the order on the LC.
	Standard UOM	Char(4)	Rag_style.standard_uom	Standard unit of measure of the quantity of the item for the order on the LC.
	Earliest Ship Date	Char(14)	lc_detail.earliest_ship_date	The earliest date an order on the LC can be shipped, formatted to 'YYYYMMDDHH24MISS'
	Latest Ship Date	Char(14)	lc_detail.latest_ship_date	The latest date an order on the LC can be shipped, formatted to 'YYYYMMDDHH24MISS'
	item description	Char(100)	ITEM_MASTER.desc_up	Item's description
File Detail	File Type Record Descriptor	Char(5)	TMERC	Identifies file record type
	File Line Sequence	Number(10)	line number in file	ID of current line being created for output file

	Number			
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Merchandise Description	Char(2000)	lc_detail.merch_desc	Contains the merchandise description of the field.
File Detail	File Type Record Descriptor	Char(5)	TDTCM	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Detail Level Comments	Char(2000)	lc_detail.comments	Holds any comments that the user has added to the Letter of Credit detail record.
File Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Transaction detail line count	Number(10)	ID of current line being created for output file	Sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Location list batch rebuild [lclrbld]

Design Overview

The Location List Rebuild program performs the rebuilding of all location lists that have static_ind = 'N' and batch_rebuild_ind = 'Y'. It calls package function LOCLIST_BUILD_SQL.REBUILD_LIST to generate LOC_LIST_DETAIL records based on the store and warehouse criteria on the LOC_LIST_CRITERIA table.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Phase II or III. (This is kind of a standalone batch program right now. No other batch program is using the result of location list rebuild at the moment.)

Scheduling Diagram:

Pre-Processing:

Post-Processing:

Threading Scheme: LOC_LIST

Restart Recovery

Logical Unit of Work (recommended Commit check points)
Driving Cursor

The Logical Unit of Work for this program is location list number. Since this unit of work is unique, there is no need to worry about things like reallocating memory or staggered posting.

The following driving cursor is used (with restart/recovery):

```
select l.loc_list
from loc_list_head l,
     v_restart_loc_list rv,
where rv.driver_value = l.loc_list
and l.static_ind = 'N'
and l.batch_rebuild_ind = 'Y'
and rv.driver_name = :os_restart_driver_name
and rv.thread_val = :oi_restart_thread_val
and rv.num_threads = :oi_restart_num_threads
and l.loc_list > NVL(:oi_restart_loc_list, -999)
order by l.loc_list;
```

Program Flow**Structure Chart****Shared Modules**

Listing of all externally referenced functions and Stored procedures and description of usage

LOCLIST_BUILD_SQL.REBUILD_LIST – for rebuilding the LOC_LIST_DETAIL records for a given location list number.

Data Structures

Constructs used to hold and manage data within the program

Function Level Description

All database interactions required and error handling considerations

main() – Follows Retek standards for the main() function. Logs on to the database, calls init(), process(), and final().

int() – Calls restart_init() to start up restart recovery and bring back the bookmark string in case of a restart.

process() – The controller for most of the processing in the program. This function opens and fetches from the driving cursor according to Retek standards, using array processing. For each record fetched from the driving cursor, rebuild_loc_list() will be called to rebuild location list for the given location list number. When the fetched array has been fully processed, restart_commit() should be called to commit the work, and update the restart/recovery variables.

rebuild_loc_list() – Calls the package function LOCLIST_BUILD_SQL.REBUILD_LIST to rebuild LOC_LIST_DETAIL records for a given location list number.

final() – Calls restart_close() to wrap up restart recovery.

I/O Specification

All files layouts input and output

Testing Scenarios

1. Create a few location list, some with static_ind = 'N' and batch_rebuild_ind = 'Y', some with static_ind = 'Y' and/or batch_rebuild_ind = 'N'.
2. Add store and/or warehouse criteria without generating the location list.
3. Run the batch program lclrbld.pc to generate location lists for all location lists with static_ind = 'N' and batch_rebuild_ind = 'Y'.
4. Add some stores and warehouses that will fit the criteria, run lclrbld.pc again. New stores and warehouses that fit the criteria should be added on the location list.
5. Remove some stores and warehouses that fit the criteria, run lclrbld.pc again. A new location list should be generated without the removed stores and warehouses.
6. Test array processing.
7. Test restart recovery.

Technical Issues**Need to create a view V_RESTART_LOC_LIST:**

```

select distinct rc.driver_name driver_name,
       rc.num_threads num_threads,
       l.loc_list driver_value,
       restart_thread_return(l.loc_list, rc.num_threads) thread_val
from restart_control rc,
     loc_list_head l
where rc.driver_name = 'LOC_LIST';

```

Need to add a record to RESTART_CONTROL for program lclrbld:

```

program_name: lclrbld
program_desc: location list batch rebuild
driver_name: LOC_LIST
num_threads: 1
update_allowed: Y
process_flag: T
commit_max_ctr: 100

```

Need to add records to RESTART_PROGRAM_STATUS for program lclrbld. The number of records added should be equal to the num_threads in RESTART_CONTROL table for program lclrbld:

```

restart_name: lclrbld
thread_val: 1
start_time: Null
program_name: lclrbld
program_status: ready for start
restart_flag: Null
finish_time: Null
current_pid: Null
current_operator_id: Null
err_message: Null

```

When testing, use num_threads of at least 2.

Letter of credit amendment download [lcmdnld]

Design Overview

This purpose of this new Letter of Credit amendment download program, which runs nightly, is to process LC amendments based on the amendments flagged during the day by the Select LC Amendments action on the LC Find form (lcfind.fmb) and then written to the staging table LC_DOWNLOAD. This program should be included in the standard download phase of the batch cycle. The amendments associated to the LCs flagged for a given day will be downloaded in full to a standard format Retek flat file. A separate program will later separate the amendments, sorting by Issuing Bank, into separate SWIFT formatted files which will then be sent to the individual banks (S.W.I.F.T. formats are used when transmitting financial information and are a globally recognized sets of standards). After this program has processed all of the LCs amendments on LC_DOWNLOAD, these records will be deleted and their status on LC_AMENDMENTS will be updated to 'D' for Download.

Affected Tables:

<i>TABLE</i>	<i>INDEX</i>	<i>SELECT</i>	<i>INSERT</i>	<i>UPDATE</i>	<i>DELETE</i>
LC_AMENDMENTS	Yes	Yes	No	Yes	No
LC_HEAD	Yes	Yes	No	No	No
LC_DOWNLOAD	No	Yes	No	No	Yes
ADDR	Yes	Yes	No	No	No
PARTNER	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
CODE_DETAIL	Yes	Yes	No	No	No
ITEM_MASTER	Yes	Yes	No	No	No
DOC	No	Yes	No	No	No
REQ_DOC	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: lcmt707 perl script

Threading Scheme: No threading due to low volume.

Restart Recovery

Restart/recovery for this program is set up at the lc_ref_id level.

The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Program Flow



Shared Modules

LANGUAGE_SQL.GET_CODE_DESC – given the code and code type, this function returns the code description.

LC_AMEND_SQL.AMEND_TEXT—given the order_no, item, amended_field, amended_Desc, original_value, new_value, effect, currency_code, indicator (N if it's a sku, Y if it's a style), this function returns the text field.

LC_AMEND_SQL.STYLE_ROLLUP – given the lc_ref_id, order_no, item, amended_field, amend_no, this function returns the new_value, original_value, effect.

Function Level Description

Init():

This function performs preliminary processing and populates global variables. It will call retek_init to start up restart-recovery and bring back the bookmark string in case of a restart. It will also populate vdate, to be used when writing to the output file. Then call write_fhead to open the output file and write the FHEAD information.

Process():

Call the init_arrays function to initialize the arrays.

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from lc_download and lc_head where the lc_download.lc_ref_lc_id value is equal to the lc_head.lc_ref_id value and greater than zero and lc_download.lc_download_type='M'.

This cursor essentially retrieves the bulk of the information that will be written in the header line.

```
EXEC SQL DECLARE c_lc_download CURSOR FOR
      SELECT ROWIDTOCHAR(l.rowid),
```

```

        h.issuing_bank,
        h.lc_ref_id,
        NVL(h.bank_lc_id, ' '),
        h.currency_code,
        NVL(TO_CHAR(h.confirmed_date,
                    'YYYYMMDDHH24MISS'), ' '),
        h.beneficiary
FROM   lc_head h,
       lc_download l
WHERE  h.lc_ref_id = l.lc_ref_id
       AND l.lc_ref_id > nvl(:ps_restart_lc,0)
       AND l.lc_download_type = 'M'
ORDER BY h.issuing_bank, h.lc_ref_id;

```

For each record that is fetched, call `fill_delete_arrays`, which copies the rowid value into an array. Call `write_thead`, which collects the remaining header information and writes it to the output file. Call `get_staple`, which gets the detail information from any staple items. Call `get_style`, which gets the detail information for any item that is an item_parent. Call `write_ttail`, which writes the trailer information to the output file.

Call `update_lc_amendments` to update all the statuses on `lc_amendments` table and call `delete_lc_amendments` to delete all the records on `lc_download` table.

Restart/Recovery and force commit is performed here.

Final():

This function will call `write_ftail` first to write the FTAIL information, then call `retek_close` to wrap up restart-recovery and populate the final error message, if necessary.

Write_fhead():

This function will write the FHEAD line of the output file.

Write_thead():

This function, which takes in the `header_struct`, will write the THEAD line of the output file according to the layout presented later in this document. Most of the variables can be written to the file as is, but some of them will have to be used to fetch additional information from the database. The header structure should then be passed into the `get_all_addresses` function so that all the name and address information can be fetched back for the `issuing_bank` and the `beneficiary`. Also, the function `get_current_amount` should be called to figure out the amount of the Letter of Credit only including already fully processed ('D'ownloaded) amendments.

Write_staple():

This function, which takes in the `detail_struct`, will write the TDETL line of the output file according to the layout presented later in this document when the item is a level one, transaction level, non-pack item. In order to get some of the values, `check_amended_field` will need to be called to get the code description.

Write_style():

This function, which takes in the `detail_struct`, will write the TDETL line of the output file according to the layout presented later in this document when the item is an item parent or item

grandparent. In order to get some of the values, `check_amended_field` will need to be called to get the code description.

Write_ttext():

This function, which takes in the `order_no`, `item`, `amended_field`, `amended_desc`, `original_value`, `new_value`, `effect`, `currency_code` and indicator (which is Y if the item is an item parent and N if the item is a child item), will write a TTEXT record to the output file using the `O_amend_text` variable passed out from the `LC_AMEND_SQL.AMEND_TEXT` function call.

Write_ttail():

This function will write the TTAIL line of the output file.

Write_ftail():

This function will write the FTAIL line of the output file.

Create_format_string():

This function formats the strings for the FHEAD, THEAD, TDETL, TTEXT, TTAIL and FTAIL records in the output file.

Init_arrays():

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on `restart_control`.

Get_current_amount():

This function will take in the `lc_ref_id` and pass out the current amount of the LC. To get the current amount, find the amount from `LC_HEAD` (which is the original amount that the LC was set for) and then, by joining with the `LC_AMENDMENTS` table, add to that all of the `EFFECT` columns for that LC number and whose statuses are in 'D'ownloaded. Note: normally, the net amount is figured out using all of the amendments as long as they have been generated. In this case, because the bank does not have knowledge of these amendments, we want to figure out the current total of the LC using only the amendments they have knowledge of (i.e. ones that have already been downloaded and sent to them).

Get_all_addresses():

This function will call `get_address`, taking in `Issuing_bank`, `Beneficiary`, and will fetch back name and address information.

For Bank and Applicant, `PARTNER.PARTNER_DESC` should be fetched back where the `partner_type` = 'BK' and the `partner_id` = the bank (for the banks or the `credit_avail_with`) or where the `partner_type` = 'AP' and the `partner_id` = applicant (for the applicant). Then all the address information (`add_1`, `add_2`, `add_3`, `city`, `state`, `post`, `country_id`) should be fetched off the `addr` table. Again, fetch this information using the `module` = 'PTNR', `key_value_1` = 'BK' (for the banks or `credit_avail_with`) or 'AP' (for the Applicant), and `key_value_2` = `issuing_bank ID` or the applicant ID.

For the Beneficiary, first the supplier name should be fetched back from the `SUPS` table where `Beneficiary` = `Supplier`. Then, off the `ADDR` table the address information (`add_1`, `add_2`, `add_3`, `city`, `state`, `post`, `country_id`) should be fetched where the `module` = 'SUPP' and `key_value_1` = `beneficiary ID` (`key_value_2` is NULL).

Get_address():

This function will be called by `get_all_addresses` and will return the information of name and address.

Check_amended_field():

This function should take in the `amended_value` and the `new_value` and then perform a series of if statements that deal with all the possible types of amended fields. If the `amended_value` is 'PE' (place of expiry), or 'PRT' (purchase type), then the `new_value` should be passed into `get_desc`, along with the appropriate `code_type`. `Get_desc` will pass back the `code_description` which should be passed out of the function.

Place_of_expiry	LCPE
Purchase_type	LCPT

Get_desc():

This function takes in a `code` and a `code_type` and fetches the description for that code using `LANGUAGE_SQL.GET_CODE_DESC` and passes the returned `O_code_desc` value out of the function.

Update_lc_amendments():

This function will update the status of the amendments to 'D'ownloaded once they have been successfully processed. This is by rowid. A second check was added because some of the amendments are "rolled up" into one – and so not all records were being updated. Now a check is done after the initial update and if it missed any – it will update them.

Delete_lc_download():

This function will delete all records from `LC_DOWNLOAD` that have been fully processed. Done by rowid.

Get_staple():

With the given header information, all of the corresponding detail records for any items will be retrieved with a cursor that joins `lc_amendments` and `item_master` where `lc_amendments.lc_ref_no = lc_ref_no` from header information and `lc_amendments.item = item_master.item` or the `lc_amendments.item` is null and `item_master.item = (select min(item) from item_master)`, and `lc_amendments.status` is 'A' and the `lc_amendments.amend_no` is not null and the item is a level one, transaction level, non-pack item. For each record, call `fill_update_arrays` to put that record in the update array. Call `compare_value` to make sure the effect is valid, and call `write_staple` to write the TDETL line to the output file. Call `write_ttext` to write the TTEXT line to the output file.

Get_style:

With the given header information, all of the corresponding detail records for any styles will be retrieved with a cursor that joins `lc_amendments` and `item_master`. For each record, call `fill_update_arrays` to put that record in the update array. Call `style_rollup` to retrieve more detail information. Call `compare_value` to make sure the effect is valid, and call `write_staple` to write the TDETL line to the output file. Call `write_ttext` to write the TTEXT line to the output file.

Style_rollup:

Calls the `LC_AMEND_SQL.STYLE_ROLLUP` package to retrieve the `new_value`, `original_value` and `effect` for a given `lc_ref_id`, `order_no`, `item`, `amended_field` and `amend_no`.

Fill_update_arrays:

Adds a record to the array. If more space needs to be allocated for the array, it will do that as well.

Fill_delete_arrays:

Adds a record to the array. If more space needs to be allocated for the array, it will do that as well.

I/O Specification

Output file

It should be noted that for every TDETL line, there will be a TTEXT line immediately following it and containing a text summary of the individual amendment to follow.

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	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	File Type Definition	Char(4)	LCAM	Identifies file as 'Letter of Credit Amendment'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Issuing Bank	Char(10)	lc_head.issuing_bank	Used to sort the LCs into individualized bank SWIFT formatted files (using another program) – bank where LC application is headed.
	Issuing Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = issuing_bank and partner_type = 'BK'.
	Issuing Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Issuing Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Issuing Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Issuing Bank City	Char(20)	addr.city	City bank located in

	Issuing Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Issuing Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Issuing Bank Country	Char(3)	addr.country_id	Country bank located in.
	Letter of Credit	Number(8)	lc_detail.lc_ref_id	The LC_REF_ID off the LC_DETAIL table.
	Bank Letter of Credit ID	Char(16)	lc_head.bank_lc_id	The BANK_LC_ID off the LC_HEAD table.
	Currency Code	Char(3)	lc_head.currency_code	The CURRENCY_CODE off the LC_HEAD table.
	Date of Issue/ Transfer of the Credit	Char(14)	lc_head.confirmed_date	The date that the Issuing Bank thinks is the date of issue – when it was officially confirmed, formatted to ‘YYYYMMDDHH24MISS’.
	Current Amount of LC	Number (20,4)		This amount will be calculated in the get_current_amount() function and will be the net amount of the LC calculated only using amendments that have been downloaded. Normally, the net amount is calculated using amendments in the ‘D’ownloaded status.
	Beneficiary	Number(10)	lc.head.beneficiary	Party in favor of which the LC is being issued.
	Beneficiary Name	Char(32)	supsup_name	Beneficiary (supplier) name from the SUPS table.
	Beneficiary Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Beneficiary Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Beneficiary Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Beneficiary City	Char(20)	addr.city	City beneficiary located in
	Beneficiary State	Char(3)	addr.state	State, if applicable, where beneficiary located in.
	Beneficiary Post Code	Char(10)	addr.post	Post code, if applicable, where beneficiary located in.
	Beneficiary Country	Char(3)	addr.country_id	Country beneficiary located in.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record’s position in the file by line number

	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Amendment Number	Number(8)	lc_amendments.amend_no	Holds the amendment number for the amendment.
	Order_no	Number(8)	lc_amendments.order_no	Order_no, if applicable, that is attached to the LC that is being amended.
	Item	Char(25)	lc_amendments.item	Item being amended, either a Style or Staple sku.
	Value Being Amended	Char(6)	lc_amendments.amended_value	LC Field being amended. Can be any of the following code_types: CODE CODE_DESC ----- AI Add Item AO Add PO ARQD Add Required Doc. C Cost ED Expiration Date ESD Earliest Ship Date LSD Latest Ship Date NA Net Amount ND Negotiation Days OC Origin Country OQ Order Quantity PE Place of Expiry PRT Presentation Terms PSF Partial Ship Flag RI Remove Item RO Remove PO RRQD Remove Required Doc. TFF Transferable Flag TSF Transshipment Flag
	Value Being Amended Description	Char(40)	code_detail.code_desc	The Value Being Amended decoded (see the above list). Will possibly be used when printing to the SWIFT file MT 707 for clarity.
	Original Value of Amended Field	Char(45)	lc_amendments.original_value	Current value of field that is being amended.
	New Value of Amended Field	Char(45)	lc_amendments.new_value	New value of the field that is being amended.
	Description of New Value	Char(40)	code_detail.code_desc	The new value decoded (or fetched from a table, as in the origin_country case)– only applicable to the following amended values: place of expiry, title_pass_location, origin_country, presentation terms, purchase type.
	Sign	Char(1)		If the effect is negative it will be

				“-“ if the effect is positive it will be “ “.
	Effect	Number (20,4)	lc.amendments.effect	Effect that amendment will have on LC if amendment to change qty or cost of a PO or amount of LC itself.
	Date of Amendment	Char(14)	Lc_amendments.accept_date	Date on which Issuing Bank (or issuing party, in this case the retailer) considers the credit as being amended, formatted to ‘YYYYMMDDHH24MISS’.
Transaction Text	File Type Record Descriptor	Char(5)	TTEXT	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record’s position in the file by line number
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Amendment Text	Char(2000) <i>{NOTE – size uncertain, within the package, variable allow for char 2000}</i>	text description	A text description of the individual amendment (for each TDETL line of the output file) built by the package LC_AMEND_SQL.AMEND_TEXT.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record’s position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

SWIFT file conversion [lcmt700.perl]

Design Overview

The S.W.I.F.T file is in a different file layout than the standard Retek interface file format. This program will convert the standard Retek flat file into the bank specific S.W.I.F.T. MT 700 output files. One output file will be created for each issuing bank in the interface file. A Perl script program will do the file conversion.

Scheduling Constraints

Pre/Post Logic Description

This program should be run after Letter of Credit application download program (lcadnld.pc).

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and stored procedures and description of usage

Function Level Description

All database interactions required and error handling considerations

main program flow:

- check for the correct number of arguments (should be 2: inputfile outputfile)
- open the input file
- read the first line of the file and verify that it's an FHEAD line – if not, exit.
- Initialize counter variables. These will be set throughout the program and then totaled.
- while a line is read from the input file:
 - if line begins with THEAD:
 - call populate_THEAD()
 - set ctrl to 1
 - if the issuing bank has changed from the previous issuing bank, then:
 - open a new file. Filename should be *outputfile_issuing_bank* (e.g. *output1_NORWEST*)
 - call init_variables()
 - elsif line begins with TDOCS:
 - call populate_TDOCS()
 - elsif line begins with TDETL:
 - call populate_TDETL()
 - elsif line begins with TMERC
 - read in the merch_desc from the input file.
 - elsif line begins with THDCM or TDTCM
 - skip to next line

- elsif line begins with TTAIL:
 - if this is not the first MT700 for this bank:
 - print the ASCII code 3 separator
 - total the counters.
 - call write_MT700()
 - call write_MT701_45B()
 - call write_MT701_46B()
 - call write_MT701_47B()
- elsif line begins with FTAIL
 - if at least one bank has been processed, close the output file
 - stop reading lines from the input file
- else
 - print an error message about an unexpected line type and exit
- now that the while loop is done, close the input file

init_variables()

Assign empty values to global variables that are unique to a single MT700/701 application.

populate_THREAD()

Read substrings from the current line into global variables to be used in the write_* functions. Check the SWIFT file format and the leadnld file format to see if any truncation or manipulation must be done.

populate_TDOCS()

Read the swift tag and body text from the current line. Replace consecutive spaces in the body text with a single space. The body text should become a part of whatever field the swift tag indicates.

populate_TDETL()

Read the order number, item, cost, quantity, earliest ship date, latest ship date, and description from the current line.

Populate()

- if the cost is NULL, set it to '0'
- if the quantity is NULL, set it to '0'
- create a string as shown in field 45B of the MT701 form below. How this string is built depends on the form type, which is read in populate_THREAD().
- push this string into the details array.

write_MT700()

- print the line "MT/PRIORITY:700 02" followed by a newline
- print all fields as indicated in the file format below in the case of fields 71B, 48, 78 and 72, the tag will be concatenated to the front of each string and a '~+' will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function trunc_wrap which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

write_MT701_45B()

- if the MT701 field 45B variable is not null:
 - print the header line "MT/PRIORITY: 701 02"
 - print the field 27 line

- print the field 20 line
- Concatenate 45B to the front of the 45B field string and a '~+' will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function `trunc_wrap` which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

write MT701 46B()

- if the MT701 field 46B variable is not null:
 - print the header line "MT/PRIORITY: 701 02"
 - print the field 27 line
 - print the field 20 line
 - Concatenate 46B to the front of the 45B field string and a '~+' will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function `trunc_wrap` which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

write MT701 47B

- if the MT701 field 47B variable is not null:
 - print the header line "MT/PRIORITY: 701 02"
 - print the field 27 line
 - print the field 20 line
 - Concatenate 47B to the front of the 45B field string and a '~+' will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function `trunc_wrap` which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

trunc_wrap()

arguments: 1) input string
 2) maximum number of characters in a line
 3) maximum number of lines

- declare local variables to hold the total number of lines, the current line, and an array to store all lines
- if the length of the input string is evenly divisible by the max number of chars in a line
 - total number of lines = $\text{int}(\text{length of input string} / \text{max number of chars in a line})$
- else
 - total number of lines = $\text{int}(\text{length of input string} / \text{max number of chars in a line}) + 1$
- for each line (but not more than the max number of lines)
 - create a string from the next (max chars in line) number of characters from the input string
 - if there are less than (max chars in line) remaining in the input string, right pad current line with spaces until length = max chars in line
 - push the current line into the storage array
- return the storage array

I/O Specification

- All files layouts input and output

The SWIFT MT 700 output file should be in the following format:

- Most output fields are contained in their own line (or 3-4 line for addresses).
- Each application consists of four parts, one MT 700 and three MT 701s, which are ordered through the Sequence of Total field: ex. ':27:1/4 MT 700' is the first (MT 700) part of the application
- MT 700 and MT 701s will be mingled in the same file.
- Each record starts with a colon and a SWIFT field identifier, followed by another colon: ex. ':40A:'-
- Each application is separated by a line with only the ASCII 3 symbol (a heart) on it.

Examples of how individual lines of the MT 700 or MT 701 should look:

```
:27:1/4
:40A:IRREVOCABLE
:20:29893098
:23:NOREF
:31C:910906
:31D:911022DALLAS
:51D:NORTHERN TRUST INT'L BANKING CORP.
ONE WORLD TRADE CENTER
SUITE 3941
NY, NY 10048 USA
```

The layout of the S.W.I.F.T MT 700 (Issue of a Documentary Credit) file is as follows:

SWIFT I.D. DATA TYPE CODES (refer to SWIFT User Handbook – Standards General Information – October 1998 release for formatting information):

Notes:

- 1) There is always a new line (nl) after every individual SWIFT ID (and there may be more than one line within an individual field (example 59 – Beneficiary, four lines to hold address information)).
- 2) In some situations, certain fields will be blank. . . these fields should be skipped over. In other words, no blank line or tag should be printed indicating the field is blank. Simply ignore it.

MT700 Mapped to Retek Fields:

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments

27 – Sequence of Total	1n/1n	Number of parts to each application (1 700 and 3 701s are the max so n/4)	'1/4'	Will be 1/4 for LC application body.
40A – Form of Documentary Credit	24x	Defines condition for guarantee of funds and acceptance of LC credit. Can be REVOCABLE or IRREVOCABLE. Then, if applicable, followed by a blank character and the word TRANSFERABLE.	THEAD: - Form of Letter of Credit–I, Char(11) ' - Form of Letter of Credit–II, Char(12)	If the revocable_ind (Form of LC – I) is 'Y' then REVOCABLE, if 'N' then IRREVOCABLE. If the transferable_ind (Form of LC – II) is 'Y', then TRANSFERABLE, otherwise nothing.
20 – Documentary Credit Number	16x	LC number. The one assigned by the Sender (retailer).	THEAD - Letter of Credit ID, Number(8) OR - Bank LC Ref ID, Char(16) – if available	Should the Bank LC Ref ID be used if available? Current position – For pre-issued Bank LC Ref IDs, if the bank LC ref ID is populated for the LC, this should be passed to the output file. If it is not available, in internally generated LC ref ID should be passed.
31C – Date of Issue	6!n	Date on which issuing bank considers the LC as being issued. Blank = date message sent.	NULL	This could be one of three possibilities. The application date or the download date or NULL (which would imply the download date). Current position – use NULL which will be taken as the download date by the receiver.

31D – Date and Place of Expiry	6!n29x	Place and latest date that documents may be presented.	THEAD - Expiration Date, Date - Place of Expiry Description, Char(29)	Put date in correct format (YYMMDD) and concatenate with truncated description (truncate from 40 to 29).
50 – Applicant	4*35x	Party and address on whose behalf the LC is being issued.	THEAD - Applicant Name, Char(35) nl - Applicant Address I, Char(30) nl - Applicant Address II, Char(30) nl - Applicant City, Char(16) ', ' - Applicant State, Char(3) ' ' - Applicant Post Code, Char(10) ' ' - Applicant Country, Char(3)	Truncate applicant name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.
51D – Issuing Bank	Option D- [/1!a][34x] 4*35x	This field specifies the name and address of the financial institution which is acting on behalf of itself, or a customer, to initiate the transaction.	THEAD - Issuing Name, Char(35) nl - Issuing Address I, Char(30) nl - Issuing Address II, Char(30) nl - Issuing City, Char(16) ', ' - Issuing State, Char(3) ' ' - Issuing Post Code, Char(10) ' ' - Issuing Country, Char(3)	Truncate the issuing bank name to 35. New lines after name, address I and address II. Truncate city to 16 to accommodate city, state, postal code and country in line 4.

59 – Beneficiary	[/34x] 4*35x	Specifies the party and address in favor of which the LC is being issued.	THEAD - Beneficiary Name, Char(35) nl - Beneficiary Address I, Char(30) nl - Beneficiary Address II, Char(30) nl - Beneficiary City, Char(15) ', ' - Beneficiary State, Char(3) ' ' - Beneficiary Post Code, Char(10) ' ' - Beneficiary Country, Char(3)	Since an account can not be defined within RTM for the beneficiary, the optional [/34x] Account will not be used. Truncate beneficiary name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.
32B – Currency Code, Amount	3!a15d	Currency Code and amount of LC.	THEAD - Currency Code, Char(3) - Amount, Number(20,4)	Truncate Amount to the rightmost 14 characters. Insert a comma where the implied decimal point would be. The resulting string will have 15 characters including the comma. Current position - given that RTM currency amounts can be up to a (20,4), any LC applications with amounts not able to fit into a 15d field will need to be rejected and not downloaded.
39A – Percentage Credit Amount Tolerance	2n/2n	Tolerance relative to LC credit amount as a percentage plus and/or minus that amount. First field spec. positive tol, 2 nd negative tol.	THEAD If Amount_type = 'A'pproximately then Variance Percent, Char(2) '/' Variance Percent, Char(2) If Amount_type = 'E'xact then - this field is NULL and field 39B is used.	Currently in RTM, can only enter one variance, therefore positive and negative variance percents are always the same. If this field contains a value, 39B cannot, and vice versa.

39B – Maximum Credit Amount	13x	Further qualifies LC credit amount by using one of following terms: UP TO, MAXIMUM, NOT EXCEEDING	THEAD If Amount_type = 'E'xact or - Specification = 'M'aximum then 'MAXIMUM'	RTM currently does not have fields that map directly to this. RTM currently has Amount Type and Specification. Amount Type can be Exact or Approximately. Specification can be 'M'aximum or others like 'Insurance to be Paid Additionally. Current Position – if Specification = Maximum, set this field to MAXIMUM or if Amount Type = 'Exact, set this field to MAXIMUM. If this field contains a value, 39B cannot, and vice versa.

41a – Available With. . . By . . .	Option A – 4!a2!a2!c[3!c] 14x Option D – 4*35x 14x	Identifies bank authorized to pay, accept or negotiate credit. Followed by how credit is available: BY PAYMENT BY ACCEPTANCE BY NEGOTIATION BY DEF PAYMENT BY MIXED PYMT If either last two then fields 42P and 42 M necessary (one or the other).	THEAD - Credit With Name, Char(35) nl - Credit With Address I, Char(30) nl - Credit With Address II, Char(30) nl - Credit With City, Char(16) ', ' - Credit With State, Char(3) ' ' - Credit With Post Code, Char(10) ' ' - Credit With Country, Char(3) nl - Presentation Terms Description, Char(14)	Since RTM has no facilities to maintain BICs, option D will always be used for this field (i.e. 41D) Truncate credit available with name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4. Use the decoded presentation terms value to arrive at one of these 'by' options: BY PAYMENT BY ACCEPTANCE BY NEGOTIATION BY DEFPAYMENT BY MIXED PYMT
42C – Drafts at . . .	3*35x	Specifies the terms of the drafts to be drawn under the LC.	THEAD - Drafts At Description, Char(40)	Since the decode can be up to 40 chars, line wrap by word to the next line if required.

42D – Drawee	Option D – [/1!a]/34 x] 4*35x	Identifies drawee of drafts to be drawn under the LC. Optional act # line must not be present	THEAD - Paying Bank Name, Char(35) nl - Paying Bank Address I, Char(30) nl - Paying Bank Address II, Char(30) nl - Paying Bank City, Char(16) , ‘ - Paying Bank State, Char(3) ‘ ‘ - Paying Bank Post Code, Char(10) ‘ ‘ - Paying Bank Country, Char(3)	Truncate the paying bank name to 35. New lines after name, address I and address II. Truncate city to 16 to accommodate city, state, postal code and country in line 4.
43P – Partial Shipments	1*35x	Specifies whether or not partial shipments permitted.	THEAD If - Partial Shipment Indicator = ‘Y’ then ‘ALLOWED’ else if - Partial Shipment Indicator = ‘N’ then ‘NOT ALLOWED’	
43T – Transshipment	1*35x	Specifies whether or not transshipments permitted.	THEAD If - Transshipment Indicator = ‘Y’ then ‘ALLOWED’ else if - Transshipment Indicator = ‘N’ then ‘NOT ALLOWED’	

44A – Loading on Board /Dispatch/ Taking in Charge at/from . . .	1*65x	Specifies place of dispatch or taking in charge of the goods or loading on board.	THEAD - Fob Title Pass Decode, Char(40) ‘ - Fob Title Pass Description, Char(45)	Current position – Will use FOB title pass decode and description. Since the maximum length of both of these fields combined is 85, truncate both appropriately to fit within 65. Take full length of decode text (not including trailing spaces) and concatenate with truncated description.
44B – For Transportation to . . .	1*65x	Specifies final destination of goods.	THEAD Transport_to, Char(40)	This is a new column in the input file
44C – Latest Date of Shipment	6!n	Specifies the latest date for loading/ dispatch/ taken in charge.	THEAD - Latest Shipment Date, Date(6)	Use date format YYMMDD.
71B – Charges	6*35x	Charge to be borne by the beneficiary. If blank then all charges except negotiation And transportation. Charges to be borne by applicant.	TDOCS If document type = ‘Charges’ then -Text Body	<i>This is a new document type created for the code type ‘DOCT’.</i>
48 – Period for Presentation	4*35x	Number of days for negotiation, payment, and acceptance. Absence = 21 day default where applicable.	THEAD -Narrative - Letter of Credit Negotiation Days, Char(3)	Hardcode this narrative and include the negotiation days: <i>“DOCUMENTS TO BE PRESENTED WITHIN <Negotiation Days> DAYS AFTER DATE OF ISSUANCE OF THE SHIPPING DOCUMENTS BUT WITHIN THE VALIDITY OF THIS CREDIT”.</i>

49 – Confirmation Instructions	7!x	Contains confirmation instructions for the Receiver, as follows: CONFIRM MAY ADD WITHOUT	'WITHOUT'	<i>RTM currently does not have fields that map directly to this. IMC hard-coded 'WITHOUT'</i> Current position – since this is a mandatory field, will default 'WITHOUT'.
78 – Instructions to the Paying/ Accepting/ Negotiating Bank	12*65x	Specifies instructions to the paying, accepting or negotiating bank.	TDOCS If document type = 'Bank Instructions' then - Text Body	New document type being created for code type 'DOCT'.
57a – “Advise Through” Bank	Option A – [/1!a]/[34x] 4!a2!a2!c[3!c] Option B – [/1!a]/[34x] [35x] Option D – [/1!a]/[34x] 4*35x	Name of bank, if different from Receiver (i.e. issuing bank) through which the LC is to be routed/ advised/ confirmed to the Beneficiary.	THEAD - Advising Name, Char(35) nl - Advising Address I, Char(30) nl - Advising With Address II, Char(30) nl - Advising With City, Char(16) , ‘ - Advising With State, Char(3) ‘ ‘ - Advising With Post Code, Char(10) ‘ ‘ - Advising With Country, Char(3)	Since RTM has no facilities to maintain BICs or party identifiers, option D will always be used for this field (i.e. 41D) without [/1!a]/[34x] party identifier. RTM currently has fields to accommodate the advising and confirmation bank. IMC placed both bank in this field if they were different, exceeding the 4*35 length. Current position – will place only the advising bank in this field. Truncate advising bank name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.

72 – Sender to Receiver Information	6*35x	Specifies additional information to the receiver.	TDOCS If document type = 'Sender Instructions' then - Text Body	This is a new document type created for code type 'DOCT'.

MT701 Mapped to Retek Fields:

AS IMC HAS IT, EACH MT 701 ONLY HAS ONE TYPE OF SWIFT TAGGED FIELD IN IT. THE FIRST ONE (2/4) WOULD HAVE 45B (AND 45A?), THE SECOND ONE (3/4) WOULD HAVE 46B, ETC.

ALSO, SHOULD BE NOTED THAT IMC MADE THE STIPULATION IN THEIR DOCUMENTATION THAT THE MT 701s COULD BE AS LONG AS WAS NECESSARY. SEEMS THAT SWIFT WILL ALLOW YOU TO MAKE DECLARATIONS LIKE THAT?

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
27 – Sequence of Total	1n/1n	Number of parts to each application (if first MT701 then it would be 2/4, assuming there were three 701s)	'2/4' for first record, '3/4' for second record, '4/4' for third record	Will be 2/4, 3/4 and 4/4 for three MT 701s.
20 – Documentary Credit Number	16x	LC number. The one assigned by the Sender (retailer)	THEAD - Letter of Credit ID, Number(8) OR - Bank LC Ref ID, Char(16) – if available	Should the Bank LC Ref ID be used if available? Current position – For pre-issued Bank LC Ref IDs, if the bank LC ref ID is populated for the LC, this should be passed to the output file. If it is not available, in internally generated LC ref ID should be passed.

45B – Description of Goods and Services	100*65x	Description of goods and services.	TDOCS If Swift Tag = ‘45B’ then - Body Text (truncate and wrap) If Form Type = ‘S’ (short) then - TDETL description, Char(100) nl - TDETL Order No., Number(8) ’: ‘ - TDETL total cost(qty*cost), Number(32) ’: ‘ - TDETL Earliest Ship Date, Date(8) ’: ‘ - TDETL Latest Ship Date, Date(8) nl - THEAD purchase type decode, Char(40) Elsif Form Type = ‘L’ (long) then - TDETL description, Char(100) nl - TDETL Order No., Number(8) ’: ‘ - TDETL Item, Char(25) ’: ‘ - TDETL Cost, Number(32) ’: ‘ - TDETL Quantity, Number(12,4) - TDETL Earliest Ship Date, Date(8) ’: ‘ - TDETL Latest Ship	Description field must be truncated to 65 characters. It is a varchar2(2000) field on ordlc. The purchase type field should contain the decoded description. See description below for example. Also refer to the spreadsheet below for a more specific layout.

46B – Documents Required	100*65x	Documents required.	TDOCS if - Swift Tag = ‘46B’ then - Doc Text	Current position – Since the MT701 is always used, required documents will always use SWIFT tag 46B. Issue – As well, it appears that one MT701 can be used for each of description of goods and services, documents required and additional conditions. Does this mean that all 3 fields could be used in each MT701 thereby allowing for up to 3*6500 (19500) chars?
47B – Additional Conditions	100*65x	Any further conditions.	TDOCS if document type = ‘Additional Instructions’ then - Text Body	Current position – Since the MT701 is always used, special instructions (additional conditions) will always use SWIFT tag 47B. Issue – As well, it appears that one MT701 can be used for each of description of goods and services, documents required and additional conditions. Does this mean that all 3 fields could be used in each MT701 thereby allowing for up to 3*6500 (19500) chars?

Description of Goods on the MT701

Short form (45B)

MENS LEATHER JACKETS (desc on ordlc – group by this desc)

PO TOTAL COST (USD) EARLIEST/LATEST SHIPDATE

484828 10,000.00 01/11/99 – 15/11/99

SHIPPING TERMS: FOB HONG KONG

Assumptions:

Quantities unnecessary

Changes are tracked at the PO header level for amendments

SHIPPING TERMS: FOB HONG KONG

We will track changes to item quantities, costs, and shipdates for amendments

Short Form																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					

SWIFT file conversion [lcmt707.perl]

Design Overview

The S.W.I.F.T file is in a different file layout than the standard Retek interface file format. To support restart/recovery, we need to convert the standard Retek flat file into the bank specific separate S.W.I.F.T. MT 707 output files. To achieve this, we write a Perl script program that does the file conversion.

Scheduling Constraints

Pre/Post Logic Description

This program should be run after Letter of Credit amendment download program (lcmdnld.pc).

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and stored procedures and description of usage

Function Level Description

All database interactions required and error handling considerations

The SWIFT MT 707 output file should be in the following format:

- Most output fields are contained in their own line (or 3-4 line for addresses).
- Each amendment consists of only one part, the MT 707. There may be several MT 707s at any given time associated to an LC as they are grouped by amendment number at the time of creation. All TDETL records with the same amend_no will be grouped together in one MT 707.
- Each record starts with a colon and a SWIFT field identifier, followed by another colon: ex. ':40A:':-
- Each amendment is separated by a line with only the ASCII 3 symbol (a heart) on it.

Logic Setup:

The input file will be in standard Retek file format. It will potentially have numerous TDETL lines per each THEAD line. There may be numerous TDETL records for one amendment. MT 707 will write one record for each amendment – so if there are multiple TDETL records they need to be combined. There is one TTEXT for each TDETL.

There are three values that need to be calculated. 32B, 33B, 34B. 32B is the total increment or the sum of the positive effect values for each amendment. 33B is the total decrement or the sum of all the negative effect values for each amendment. 32B and 33B are separate totals for each amendment. 34B is the total difference – so it is the sum of the total increment and total

decrement. 34B is not just for one amendment though, it is for all amendments of a THEAD record – so this total will run through each TDETL in a THEAD.

For example: if the input file contains:

```
THEAD....
TDETL amendment 1, effect +1000
TTEXT
TDETL amendment 1, effect +500
TTEXT
TDETL amendment 2, effect -2500
TTEXT
TDETL amendment 3, effect +4000
TTEXT
TDETL amendment 3, effect -1000
TTEXT
TDETL amendment 3, effect +500
TTEXT
TDETL amendment 4, effect -1000
TTEXT
TDETL amendment 4 , effect -2500
TTEXT
TTAIL
```

32B for amendment 1 = 1500

33B for amendment 1 = 0

34B for amendemnt 1 = 1500

32B for amendment 2 = 0

33B for amendment 2 = 2500

34B for amendemnt 2 = -1000

32B for amendment 3 = 4500

33B for amendment 3 = 1000

34B for amendemnt 3 = 4500

32B for amendment 4 = 0

33B for amendment 4 = 3500

34B for amendemnt 4 = 1000

Examples of how individual lines of the MT 707 should look:

APPLICANT:

OPERATOR:

OPERATION DATE:

OPERATION TIME:

TEST KEY:

BATCH TOTAL:

SEGMENT TOTAL:

MT/PRIORITY:707 02

:27:1/1

:20:10001981

```

:21:1981
:52D:Bank One
100 Bank One Way
Columbus      ,OH  41984      US
:31C:990204
:30:990204
:26E:1
:59:David Fashion Creations P/L Pack
Wholesale Division
109 Ackland St.
St. Kilda     ,VA  30280-1234 US
:32B:USD500,0
:33B:USD0,0
:34B:USD500,0
:79:Letter of Credit: has been changed from 25 to 30
for Style 10049369, resulting in an effect of 500
(USD).

```

I/O Specification

All files layouts input and output

The layout of the S.W.I.F.T MT 707 (Amendment to a Documentary Credit) file is as follows:

SWIFT I.D. DATA TYPE CODES (refer to SWIFT User Handbook – Standards General Information – October 1998 release for formatting information):

Notes:

- 1 The field lengths and types in the “Retek Standard Download Format. . “ of the MT 707 are important as sometimes they are different from the information that is being placed in them and the fields may have to be truncated, rounded etc.
- 2 There is always a new line (nl) after every individual SWIFT ID (and there may be more than one line within an individual field (example 59 – Beneficiary, four lines to hold address information).
- 3 In some situations, certain fields will be blank. . . these fields should be skipped over. In other words, no blank line or tag should be printed indicating the field is blank. Simply ignore it.

MT707 Mapped to Retek Fields:

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
20 – Documentary Credit Number	16x	LC number. The one assigned by the Sender (retailer).	THEAD - Letter of Credit, Number(8)	

21 – Receiver's Reference	16x	LC number assigned by the Receiver (issuing bank in this case).	THEAD - Bank Letter of Credit ID, Char(16)	Issue – Currently in RTM, field defined as 15 long. Logged a SIR to increase its length to 16 to accommodate SWIFT standards.
23 – Issuing Bank's Reference	16x	When the MT707 is sent by a bank other than the Issuing Bank (i.e. it is sent by an Advising Bank), this field specifies the LC number of the Issuing Bank.		IMC did not use. Current position – assumption is that receiver is always the issuing bank, therefore this field does not need to be included in the output file. This is not a mandatory field.
52D Issuing Bank	Option D – [/1!a]/[34x] 4*35x	Identifies the Issuing Bank, when different from the Sender of the message.	THEAD - Issuing Bank Name, Char(35) nl - Issuing Bank Address I, Char(30) nl - Issuing Bank Address II, Char(30) nl - Issuing Bank City, Char(16) ', ' - Issuing Bank State, Char(3) ' ' - Issuing Bank Post Code, Char(10) ' ' - Issuing Bank Country, Char(3)	Since an account can not be defined within RTM for the issuing bank, the optional [/34x] Account will not be used. Truncate issuing bank name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.
31c – Date of Issue /Transfer of Credit	6!n	Date of the original issue of the LC (i.e. date on which issuing bank considers the LC as being issued).	THEAD - Confirmed Date, Date(6)	This could be one of two dates. The application date or the confirmed date. IMC used the confirmed date. Current position – use the confirmed date. Format should be YYMMDD.

30 – Date of Amendment	6!n	Date on which the Issuing Bank (or retailer) considers the credit as being amended. If blank, the day it was sent.	THEAD - Date of Amendment, Date(6)	Use date format YYMMDD.
26e – Number of Amendment	2n	Specifies the number of the amendment.	TDETL - Amendment Number, Number(2)	Issue – RTM currently allows amendment numbers up to 8 long. Will have to reject records with numbers greater than 99.
59 – Beneficiary (before this amendment – if this amendment is changing it)	[/34x] 4*35x	Specifies the party and address in favor of which the LC is being issued.	THEAD - Beneficiary Name, Char(32) nl - Beneficiary Address 1, Char(30) nl - Beneficiary Address 2, Char(30) nl - Beneficiary City, Char(16) , ‘ - Beneficiary State, Char(3) ‘ ‘ - Beneficiary Post Code, Char(10) ‘ ‘ - Beneficiary Country, Char(3)	Since an account can not be defined within RTM for the beneficiary, the optional [/34x] Account will not be used. Truncate beneficiary name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.
31E – New Date of Expiry	6!n	New/revised expiry date for presentation under the LC.	TDETL if - Value Being Amended = ‘ED’ then - New Value of Amended Field, Date(6)	The new value field is char(45) so it will need to be truncated and converted to a date of the format YYMMDD.

32B – Increase of Documentary Credit Amount	3!a15d	Currency code and amount of an increase in the LC amount.	TDETL Check the Effect field from the Retek flat file if - Value Being Amended = 'NA', 'C', 'OQ', 'AI', 'AO', 'RI', 'RO'. and if Effect is > 0 then put - Currency Code, Char(3) - total of Effects, Number(15) in SWIFT field 32B.	The integer part of Amount must contain at least one digit. A decimal comma ',' is mandatory and is included in the maximum length. Current position - given that RTM currency amounts can be up to a (20,4), any LC amendment amounts not able to fit into a 15d field will need to be rejected and not downloaded.
33B – Decrease of Documentary Credit Amount	3!a15d	Currency code and amount of a decrease in the LC amount.	and if Effect is < 0 then put - Currency Code (THEAD), Char(3) - total of Effects, Number(15) in SWIFT field 33B. NOTE: It is the total effect of all the different parts of an amendment summed up that determines whether 32B or 33B is used.	The integer part of Amount must contain at least one digit. A decimal comma ',' is mandatory and is included in the maximum length. Current position - given that RTM currency amounts can be up to a (20,4), any LC amendment amounts not able to fit into a 15d field will need to be rejected and not downloaded. If negative, put the total EFFECT amount (from LC_AMENDMENTS) here.

34B – New Documentary Credit Amount After Amendment	3!a15d	Currency code and new amount disregarding any drawings.	THEAD -Currency_code, Char(3) See second paragraph under Logic Setup in the Function Level Description section to populate the amount piece, char(15)	This field can be figured out using the LC_AMENDMENTS table: Use the Current Amount of LC field from THEAD and add to it (it will be negative if a decrease is involved) the EFFECT fields. Do this until the amend_no changes. Put the total here. The integer part of Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length. Current position - given that RTM currency amounts can be up to a (20,4), any LC amendment amounts not able to fit into a 15d field will need to be rejected and not downloaded.
39A – Percentage Credit Amount Tolerance	2n/2n	When the tolerance relative to LC credit amount as a percentage plus and/or minus that amount is being amended. This field specifies the new tolerance.		Currently in RTM, once cannot amend the tolerance percent. IMC did not use. Current position – do not include in output file. It is not a mandatory field.
39B – Maximum Credit Amount	13x	Specifies the amended qualification of the documentary credit amount.		Currently in RTM, once cannot amend the maximum credit amount. IMC did not use. Current position – do not include in output file. It is not a mandatory field.

39C – Additional Amounts Covered	4*35x	Specifies amendments to any additional amounts covered.		Currently in RTM, once cannot amend additional amounts covered. IMC did not use. Current position – do not include in output file. It is not a mandatory field.
44A – Loading on Board /Dispatch / Taking in Charge at/from ...	1*65x	Specifies amendments to the places from and to which the documents are to show loading on board/ dispatch/ taking in charge.		Currently in RTM, once cannot amend the FOB title pass decode and/or description. Current position – do not include in output file. It is not a mandatory field.
44B – For Transportation to...	1*65x	Specifies amendments to the final destination of goods.	THEAD - outloc_desc, Char(40)	This is a new column in the input file.
44C– Latest Date of Shipment	6!n	Specifies amendments to the latest ship date.	TDETL if - Value Being Amended = 'LSD' then - New Value of Amended Field, Date(6)	The new value field is char(45) so it will need to be truncated and converted to a date of the format YYMMDD.
44D– Shipment Period	6*65x	Specifies amendments to the shipment period.		Currently in RTM, once cannot amend the shipment period. Current position – do not include in output file. It is not a mandatory field.

79 – Narrative	35*50x	Specifies amendments to the documentary credit for which there is no other specific field.	TDETL All text descriptions for the amendment number. Place new line after every text description until space filled (i.e. 1750 chars).	The text fields built by LC_AMEND_SQL.A MEND_TEXT for each amended field value should be written here. If a required document gets deleted (RRQD), or added (ARQD), the old and new text will have to be shown. If required documents are added or deleted, show what the original required documents were and what they are now. I don't think the actual text can be edited once the LC has been confirmed. If a document gets added hardcode text that says something to the affect of: "the following document(s) have been added to the L/C..." If a document gets removed from an LC, the text should read: "The following document(s) have been removed from the L/C..." Codes for value_being_amended field: ARQD – Add Required Doc. RRQD – Remove Required Doc.

SWIFT file conversion [lcmt730.perl]

Design Overview

The S.W.I.F.T file is in a different file layout than the standard Retek interface file format. To support restart/recovery and to allow the user to run the reject file after the rejected records have been fixed, we need to convert the S.W.I.F.T. MT 730 input file to the standard Retek file format. To achieve this, we write a Perl script program that does the file conversion.

This script is being modified to NOT include bank charges information. It will only be used to convert confirmation information. Any reformatting that applies to charges or drawdowns will be removed.

Scheduling Constraints

Pre/Post Logic Description

This script will need to be run before lcupld.pc in phase II.

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Program Flow

Structure Chart

Shared Modules

Listing of all externally referenced functions and stored procedures and description of usage

Function Level Description

All database interactions required and error handling considerations

The newly created upload file should contain an FHEAD line (of file type definition 'LCUP'), as The OUTPUT file must also be changed. Here is its format. The main changes are fields that have been removed; the bank lc ref id has also been lengthened from 15 to 16.

Original format:

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	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	LCUP	Identifies file as 'Letter of Credit Upload'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'

File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check.
	Sender's Reference	Char(16)	lc_head.bank_lc_id	The LC number that the bank assigns to a Letter of Credit. SWIFT allows for up to 16 characters. . . we might have to truncate in the conversion script? Does it matter? Maybe make this field Char(16)?
	Receiver's Reference	Number(8)	lc_activity.lc_ref_id	The LC number that Retek assigned to the Letter of Credit. Might have to to_number it in the conversion file?
	Upload Type	Char(1)	lc_activity.trans_code	Determines what kind of message this individual FDETL is: L = Confirmation of LC B = Bank Charge/Fee D = Drawdown
	Date of Message Being Acknowledged	Char(14)	lc_activity.activity_date	If the upload type is 'L' then this date will match the date MT 700 date of issue (which we have not resolved between being the vdate or the lc_head.application_date) If the upload type is B or D no idea what to do? What if (for a charge) no message was sent . . . what would this date be? <u>Perhaps it does not matter which date is what. . just put the date in the activity_date field on LC_ACTIVITY.</u> formatted to 'YYYYMMDDHH24MISS'
	Currency of Charge	Char(3)	currency code	Should match lc_head.currency_code for that LC.
UNECES SARY???	Method of Payment	Char(1)	method of payment	Not and RMS field. . . if D, then then Amount of Charge will be debited from the account (line of

				credit). . . if B, then ? (a bill sent or something?) Can be B or D. May be unnecessary as the upload type is basically the same thing.
	Amount of Charge (or should this be Amount of Drawdown?)	Number (20,4)	lc_activity.amount	If method of payment = 'D', the amount that will be debited from the credit line (the LC) either for bank charges(? – are charges debited from the account or not?) or drawdowns.
	Amount of Bill (or should this be Amount of Charge?)	Number (20,4)	???	If method of payment = 'B' then do we have some sort of field like this. . . that is processed somewhere else besides LC_ACTIVITY.
	Account With Bank	Char(10)	lc_head.issuing_bank	Assumption that this will always be the issuing bank? Should match lc_head.issuing_bank?
	Comments	Char(2000)	lc_activity.comments	Need to truncate? This field will probably be a concatenation of the following SWIFT fields: 71B – Charges, 72 – Sender information.
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
	Total number lines	Number(10)	Specified by external system	Total number lines in file

New format:

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	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	LCUP	Identifies file as 'Letter of Credit Upload'
	File Create Date	Char (14)	vdate	date file was written by external system

				'YYYYMMDDHH24MISS' format
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	Sender's Reference	Char(16) (was 15)	lc_head.bank_id_id	The LC number that the bank assigns to a Letter of Credit.
	Receiver's Reference	Number(8)	lc_activity.lc_ref_id	The LC number that Retek assigned to the Letter of Credit.
	Date of Message Being Acknowledged	Date (char 8)	lc_activity.activity_date	If the upload type is 'L' then this date will match the date MT 700 date of issue (which we have not resolved between being the vdate or the lc_head.application_date) 'YYYYMMDD' format
	Comments	Char(2000)	lc_activity.comments	Need to truncate? This field will probably be a concatenation of the following SWIFT fields: 71B – Charges, 72 – Sender information.
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
	Total number lines	Number(10)	Specified by external system	Total number lines in file

The layout of the original S.W.I.F.T MT 730 (Acknowledgement) file is as follows (this is the INPUT file):

SWIFT I.D. DATA TYPE CODES (refer to SWIFT User Handbook – Standards General Information – October 1998 release for formatting information):

SWIFT I.D. & Description	Data Type	Description	How the MT 730 fields should be put into the Retek standard file format and what the size SHOULD be for RMS to deal with:	Comments

20 – Sender's Reference	16x	LC number. The one assigned by the Sender (issuing bank).	FDETL - Sender's reference , Char(16)	This field maps to RTM's Bank LC Ref ID.
21 – Receiver's Reference	16x	LC number assigned by the Receiver (retailer)	FDETL - Receiver's reference , Number(8) (NOREF used if unknown)	This field maps to RTM's LC Ref ID. If this field has 'NOREF', the record must be rejected since this field is used to indicate the LC within RTM to which this record applies.
25 – Account Identification	35x	Identifies the number of the account, which has been used for the settlement of charges, on the books of the Sender.		RTM currently does not have fields that map directly to this. Current position – will be included in the input file, however, will be ignored during the upload process.
30 – Date of Message Being Acknowledged	6!n	When the message is acknowledging an MT 700, this field specifies the date of issue. In all other cases, this field specifies the date on which the message being acknowledged was sent.	FDETL - Date of Message Being Acknowledged , Date	This field maps to the LC activity date. As well, if this in confirming an LC application, it will be mapped to the LC's confirmation date. Year interpretation: If YY>79 then YYMMDD = 19YYMMDD Else YYMMDD = 20YYMMDD.
32a – Amount of Charges	Option B – 3!a15d Option D – 6!n3!a15d	Contains the currency code and total amount of charges claimed by the sender of the message. When charges have been debited, D is used (:32D) and when reimbursement for charges is needed, B is used (:32B).	FDETL - Upload_type = 'C' on confirmation	Current position – Since the 730 will only be used for confirmations, this field will not contain any values. The upload type should be set equal to 'C' on confirmation.

57a – Account With Bank	Option A – [/1!a]/[34 x] 4!a2!a2!c[3!c] Option D – [/1!a]/[34 x] 4*35x	This field specifies the bank to which the amount of charges is to be remitted in favor of the Sender.	FDETL - Account With Bank , Char(10)	Current position – will be added to the input file however will be ignored in the upload process. Since RTM has no facilities to maintain BICs or party identifiers, option D will always be used for this field (i.e. 57D) without [/1!a]/[34x] party identifier.
71B – Charges	6*35x	Specification of the charges claimed.	FDETL - Comments , Char(2000)	This field maps to RTM's activity comments field. Sender to Receiver information (72) will be concatenated to this.
72 – Sender to Receiver Information	6*35x	Text explanation if wanted.	FDETL - Comments , Char(2000)	This field maps to RTM's activity comments field. Charges (71B) will be concatenated to this.

SWIFT file conversion [lcmt798.perl]

Design Overview

This new perl script will take an input file consisting of SWIFT tags and reformat it into a standard Retek field format. The input file will contain Letter of Credit (L/C) activity data for charges and drawdowns.

Stored Procedures / Shared Modules (Maintainability)

Input Specifications

Swift Tag	Description	Required ?	Datatype	Retek Field
20 – Transaction Reference Number	The Sender's unambiguous identification of the transaction. Its detailed form and content are at the discretion of the Sender.	Yes	16x – Transaction Reference Number	Bank L/C ID Lc_head.bank_lc_id Varchar2(16)
12 – Type of Financial Instrument	This field classifies the financial instrument by a description or proprietary code.	Yes	Option A- :4!c/[8c]/30x :4!c – Qualifier / - Delimiter [8c] – Issuer Code / - Delimiter 30x - Type	This field will contain a constant identifier – '798'.
77E – Proprietary Message	This field contains the proprietary message in a format agreed to by the Sender and the Receiver.	Yes	Option E- 73x [n*78x]	This field will contain the information below (fields 21, 23, 32C, 32D, 71A, 33A, 72). Carriage return, Line feed, Colon 'CrLf:' will be used to separate fields included in this 77E. For example: :77E:'CrLf' :21:10004321:CrLf' :32C:990121USD1045 etc...

				There may be multiple 77E's in one file.
21 – Related Reference	This field specifies, in an unambiguous way, a message or transaction identifier which is normally included as part of the information supplied with the message or transaction itself, and can subsequently be used to distinguish the message or transaction identified from other messages or transactions.	No	16x	P/O Number Lc_activity.order_no Number(8)
23 – Further identification	This field specifies the type of transaction being confirmed, as well as the settlement method used.	No	16x	Invoice Number Lc_activity.invoice_no Varchar2(15)
32C – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option C-6! n3! a15d 6! n – Date 3! a – Currency 15d – Amount	Charges Credited (this is interpreted as a positive amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ',' is mandatory and is included in the maximum length. Lc_activity.amount Number(20,4) Lc_activity.currency_code Varchar2(3) Lc_activity.activity_date Date
32D – Date and	This field specifies the currency	No	Option D-	Charges Debited

Amount	code and amount in a transaction and a corresponding date.		6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	(this is interpreted as a negative amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length. Lc_activity.amount Number(20,4) Lc_activity.currency_code Varchar2(3) Lc_activity.activity_date Date
33A – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option A-6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	Date, currency, amount of drawing (this is interpreted as a positive amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length. Lc_activity.amo

				unt Number(20,4) Lc_activity.curr ency_code Varchar2(3) Lc_activity.activ ity_date Date
33C – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option A- 6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	Date, currency, amount of drawing (this is interpreted as a negative amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length. Lc_activity.amo unt Number(20,4) Lc_activity.curr ency_code Varchar2(3) Lc_activity.activ ity_date Date
72 – Sender to Receiver Information	This field specifies instructions or additional information for the Receiver, Intermediary, Account With Institution or Beneficiary Institution.	No	6*35x	Comments Lc_activity.com ment Varchar2(2000)
18A – Number of Repetitive Parts	This field specifies the number of times the repetitive part(s)/sequence(s), directly before or after this field appears in the message.	No	Option A- 5n – Number of Repetitive Parts.	Number of 77E’s contained within the file.

Output Specifications

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	line number in file	ID of current line being created for output file.
	File Type Definition	Char(4)	LCCH	Identifies file as 'Letter of Credit Changes'
	File Create Date	Char(14)	Create date	current date, formatted to 'YYYYMMDDHH24MISS'
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file.
	Bank Letter of Credit Reference ID	Char2(16)	SWIFT tag 20	Bank L/C ID
	Order Number	Number(8)	SWIFT tag 21	Contains the order number that is attached to the letter of credit.
	Invoice Number	Char2(15)	SWIFT tag 23	Identifies the Issuing Bank's invoice number to which the drawdown refers. This field does not correspond to a Retek invoice number.
	Transaction Number	Char2(10)	Null	Identifies the amendment number or actual transaction number assigned by the bank.
	Transaction Code	Char2(6)	If the transaction is a Bank Charge – 'B' If the transaction is a Drawdown – 'D'	Identifies the type of transaction that occurred. The type is determined by what detail fields are received for the record. If the record contains a 33A this field will get a 'D'. If the record contains either a 32C or 32D this field will get a 'B'.
	Amount Sign	Char2(1)	SWIFT 33A, 33C SWIFT 32C, 32D	If the record contains a 33A field leave a blank space in this field. If the record contains a 33C field this field should contain a '-'. If the record contains a 32C field leave a blank space in this field. If the record contains a 32D field this field should contain a '-'.

	Amount	Number(20)	SWIFT 33A, 33C SWIFT 32C, 32D	Holds the amount of the activity. This field will have 4 implied decimal places. If SWIFT 32C or 32D (Bank Charge) contains a value, use the amount from this field. If SWIFT 33A or 33C (Drawdown) contains a value, use the amount from this field.
	Currency Code	Char2(3)	SWIFT 33A, SWIFT 32C, 32D	Contains the activity's currency code. If SWIFT 32C or 32D (Bank Charge) contains a value, use the currency from this field. If SWIFT 33A (Drawdown) contains a value, use the currency from this field.
	Activity Date	Char2(8)	SWIFT 33A, SWIFT 32C, 32D	Holds the date that the activity took place. Formatted to 'YYYYMMDD' If SWIFT 32C or 32D (Bank Charge) contains a value, use the date from this field. If SWIFT 33A (Drawdown) contains a value, use the date from this field.
	Comments	Char2(2000)	SWIFT tag 72	Holds any comments for the activity.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)		This will contain the number of FDETL lines processed.

Function Level Description

Bank Charges and Drawdowns will be received from the bank in a SWIFT lcmt798 file. This perl script will take this lcmt798 file and convert it to a standard Retek file. The file produced by this script will be uploaded into the Retek system by a new upload program (lcup798.pc).

Refer to the file layouts above for formatting and field mapping. A file containing a Bank Charge will be identifiable because it will contain a 32C or 32D field, Drawdown files will not. A field containing a Drawdown will be identifiable because it will contain a 33A or 33C field, Bank Charge files will not.

There can be multiple input files run with this program at one time. The input file will contain at least one 77E. Within the 77E will be the details of the record. A single file may contain multiple 77E's. The 18A field will contain the number of 77E records within the file. Use the value in this field to ensure the correct number of 77E records were received.

The perl script will do the following:

Check number of arguments for program name, outputfile, input file (and possibly more input files)

Create handle for output file

Write FHEAD

Open the input file and read in one line at a time call function process_one_file

Process_one_file

Read in and disregard the header information until the start of the detail information

Call function init_variables

While loop (read in one line at a time)

If the line contains the string PRIORITY:798

Call write_FDETL

Increment line counter

Else if the line contains the string :03:

Go to the next line

Else if the line contains the string :20:

Read in up to 16 characters starting with the 4th (after the :20:) into a variable

Pad to 16 with blank spaces

Else if the line contains the string :12:

Go to the next line

Else if the line contains the string :77E:

Check to see if :21: is contained on this line

If it is, read in up to 12 characters starting with the 4th (after the :21:) into a variable

Pad to 8 with blank spaces

Else if the line contains the string :21:

If it is, read in up to 12 characters starting with the 4th (after the :21:) into a variable

Pad to 8 with blank spaces

Else if the line contains the string :23:

If it is, read in up to 15 characters starting with the 4th (after the :23:) into a variable

Pad to 15 with blank spaces

Else if the line contains the string :32C: or :32D: or :33A: or :33C:

Read the first 5 characters into a variable (code)

Read the next 6 characters into a variable (date)

Read the next 3 characters into a variable (currency code)

Read the next up to 15 characters into a variable (amount)

The amount contains at least one digit and a comma. Replace the comma with a period

Multiply the amount by 10000

Add .5 to the amount and then chop off everything after the decimal point (round the number)

Zeropad the amount to 20 places

Change the dates to have 4 digit years instead of 2 digit years

If the code is :32C:, transaction_code = B, amount sign is a blank space

If the code is :32D:, transaction_code = B, amount sign is '-'

If the code is :33A:, transaction_code = D, amount sign is a blank space

If the code is :33C:, transaction_code = D, amount sign is '-'
 Else if the line contains the string :72:
 Read in all the comments line and concatenate them together into one variable
 Else if the line contains the string :18A:
 Read in up to 5 characters starting with the 5th (after the :18A:) into a variable
 Pad to 5 with blank spaces
End loop

Function zeropad
 Pads comments to 2000 characters

Function inti_variables
 Initializes variables

Function write_fDETL
 Writes variables to output file.

Scheduling Considerations

This program should be run before lcup798.pc in phase II.

Locking Strategy

None

Restart / Recovery

None

Performance

None

Security Considerations

None

Design Assumptions

Bank can pass SWIFT format expected

Letter of credit update [lcup798]

Purpose

This program will read in an input file (in standard Retek format, modified from the SWIFT 798 format by the lcmt798 Perl script) and insert the letter of credit charges and drawings contained in the file into the lc_activity table.

Objectives

Functional Area

Letter of Credit—Charges and drawings

Derived from the Process / Function Overview (PFO) and the Technical Design, the specific Functional Area of which this batch module is apart. Examples: HTS Maintenance, Receiving

Module Affected

lcup798.pc –new batch program

The program name or name of specific module being modified / designed (using the Retek naming standards). Examples: htupld.pc, pos_recpt.pc.

The program will read data from the input file, validate it, and insert it into the lc_activity table. If a record fails validation, it will be written to a reject file (that can be reprocessed by lcup798 after errors have been corrected).

Program Flow

This is a detailed diagram (structure chart type or function level text) that will define all functions performed in the module. This section will require the designer to specify the program flow and activities performed in each code segment without detailing SQL.

Thoroughly analyze the module with the sole objective of achieving the most efficient implementation. Uncover parallelism inherent in the application. Identify functional modules executed concurrently and the dependencies between these modules. In addition, identify standard computational modules, scope, performance, scheduling constraints, common functions, maintainability, and overall module integrity.

This should provide an overall picture of the LUW performed by the batch module, derived from the Technical Design.

Function Level Description

init

Initialize restart/recovery; open input file and reject file.

Be sure to check if this is a restart or not. If it's a fresh start, check to make sure that the file type (from the FHEAD line) is 'LCCH'; fatal error if it isn't

process

In a loop, read a line from the input file (use the retek_get_record library routine); validate (call validate_record), write to an array for future insert (or write to reject file, if validation failed). Do an array insert of the validated records by calling post_records whenever the commit point is

reached. Quit when the FTAIL line is read (validate the FTAIL line to check number of FDETL lines)

validate_record

Check that all dates are valid dates and all numbers are valid numbers. Call validate_lc_ref, validate_order, and validate_currency to validate the bank lc ref id, order number, and currency code against those in the database. Note that it is OK for the order number and/or the invoice number to be blank, but the bank lc ref id, amount, transaction code, activity date, and currency code must be given.

****can transaction number be blank? currently in debate—we might not ever get one**

validate_lc_ref:

Check that a given bank lc_id exists on the lc_head table and get the corresponding lc_ref_id for insert into lc_activity.

Only check against lc_head records with a status of 'C' onfirmed.

validate_order:

Check that a given order number exists on the lc_detail table for a given lc_ref_id. If it does not exist on lc_detail, check the lc_amendments table (it is acceptable if an order exists on either of these tables).

validate_currency

Check that given currency code exists on the currencies table. Call get_exchange_rate to get the exchange rate for this currency.

get_exchange_rate:

Call the CURRENCY_SQL.GET_RATE function to get the exchange rate for a given currency code and activity date. Use 'L' as the exchange type (this will pick up the letter of credit exchange rate if it exists and default to consolidation or operational exchange rate otherwise). Pass in the activity date as the effective date.

post_records

Array insert records into the lc_activity table. All columns on the table should be inserted, but note that some may be NULL.

(LC_REF_ID—must always be given

ORDER_NO—may be null

INVOICE_NO—may be null

TRANS_NO—may be null

TRANS_CODE—must be given

AMOUNT—must be given

ACTIVITY_DATE—must be given

CURRENCY_CODE—must be given

EXCHANGE_RATE—must be given

COMMENTS—may be null)

(No duplicate checking.).

final

Restart/recovery close; close input and reject files.

Stored Procedures / Shared Modules (Maintainability)

Batch modules should attempt to use stored procedures / shared modules for common functional processing units and development optimization. Any common functionality between batch and forms should be performed using the same procedure modules if applicable. Since validation and integrity checking is not performed in the same fashion in batch as in on-line, functions that are used for these purposes should not be used in batch modules. Common functionality that exists in a functional group of batch modules should be written in a common library routine. This will allow maintenance to be performed in one location, preventing unnecessary multiple program maintenance.

This section should highlight the shared / common modules used and identify exceptions. A modular design that reuses code successfully will need to make use of stored procedures, packages or common library functions. Library functions that perform specific functional sections that can be reused across modules need to be listed here and a description of their use should be given. Basic library functions for error handling, restart recovery and database interaction do not need to be defined since they are assumed to be in every module. (ie. Logon, Commit, Rollback, Write_Error etc.) Example: get_transfer_qty - called to obtain the quantity in transit of a given sku/location record to determine the total number of items affected by the markdown being processed.

Shared Module	Module Description
CURRENCY_SQL.GET_RATE	Given a currency code and date, get the exchange rate

Input Specifications

Input file:

Input Files

If input is received via a file, explicitly describe the layout (derived from the Technical Design). The client, or Retek partner, may use this document to understand for interface development work.

For example:

Record Name names the line-level record that is being described. A new record name will be used for every unique line layout in an interface file.

Field Type will define the data type of the field. Interface will always be read or written as character fields, however the data contained may need to be validated as numeric IDs or decimal numbers. The type of validation should be noted here. The following are valid field types and their validation requirements:

Default Value field will define the characters or type of data that will be expected when the field is read or written.

Description field will provide more information regarding the data in the field. Include as much information about the field that is pertinent, e.g. specify whether the unit_retail came from the ordering tables or from the item_zone_price tables.

The layout should be in table format as seen below:

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	line type	char(5)	'FHEAD'	marks beginning of file
	line number	number(10)	0000000001	unique line identifier
	file type ID	char(4)	'LCCH'	identifies this as an LC

Record Name	Field Name	Field Type	Default Value	Description
				798 file-Letter of Credit Charges
	file create date	char(14)		File create date 'YYYYMMDDHH24MISS' format
FDETL	line type	char(5)	'FDETL'	marks beginning of a transaction
	line number	number(10)		unique line identifier
	bank letter of credit reference ID	char(16)	SWIFT tag 20	Bank's LC ref ID
	Order number	Number(8)	SWIFT tag 21	order number attached to LC May be blank.
	invoice number	char(15)	SWIFT tag 23	Note: this is NOT a RMS invoice number, just a reference invoice number from the issuing bank (to which the drawdown refers) May be blank
	transaction number	char(10)		amendment number or transaction number assigned by bank—can be null
	transaction code	char(6) (why not char(1)?	B or D	'B'ank charge or 'D'rawdown
	amount	number(21)	SWIFT tag 33A, 71A	(This is a 20-digit number with a leading – sign or blank and 4 implied decimal places.) Amount of charge or drawdown.
	currency code	char(3)	SWIFT 33A, 71A	Currency that the amount is in.
	activity date	char(8)	SWIFT 33A, 32C, 32D	activity date, 'YYYYMMDD' format
	comments	char(2000)	SWIFT tag 72	Any comments associated with activity. Can be null.
FTAIL	line type	char(5)	'FTAIL'	marks end of file
	line number	number(10)		unique line identifier
	file record counter	number(10)		number of FDETL lines in file

Output Specifications

A reject file with the exact same format as the input file will be created, containing any rejected records from the input file. If no records are rejected, this file will consist only of an FHEAD and an FTAIL line.

Output Files

If output is received via a file, explicitly describe the layout (derived from the Technical Design). The client, or Retek partner, may use this document to understand for interface development work.

For example:

Record Name names the line-level record that is being described. A new record name will be used for every unique line layout in an interface file.

Field Name will name the field that is being defined.

Field Type will define the data type of the field. Interface will always be read or written as character fields, however the data contained may need to be validated as numeric IDs or decimal numbers. The type of validation should be noted here. The following are valid field types and their validation requirements:

Default Value field will define the characters or type of data that will be expected when the field is read or written.

Description field will provide more information regarding the data in the field. Include as much information about the field that is pertinent, e.g. specify whether the unit_retail came from the ordering tables or from the item_zone_price tables.

The layout should be in table format as seen below:

Database Integrity

This information derives from the Database Considerations within the Process / Functional Overview (PFO), the Conversation Flow and Database Objects of the Technical Design.

Tables used:

	Select	Insert	Update	Delete
lc_head	x			
lc_detail	x			
lc_activity			x	
lc_amendments	x			
currencies	x			
currency_rates	x			
system_options	x			

Operations that affect other entities in the system must be validated to ensure that integrity constraints have not been violated. If a record cannot exist in the system without a related parent record existing first, it is essential that the application enforce this constraint. Similarly, if a record cannot be deleted due to the existence of child records in the system the application should prevent the user from performing a delete operation.

Scheduling Considerations

This program should be run after the lcmt798 script

Run in Phase II (along with lcupld)

Many batch modules also require some initial processing or clean-up processing be performed. This functionality cannot be performed in the code itself due to multi-threading constraints. Although identified within the Technical Design, the processing that is required should be defined (cut and pasted from the Technical Design) in this section to jog the memories of Development, Quality Control and other Project Team of the interaction. Example: table truncation of temporary tables and updating system variables.

Possible threading schemes should be designed for at this time. Calculate program threading options, justify one and estimate an optimal number of threads. A threading method must be chosen and it is important to note that the threading method chosen cannot be changed dynamically.

For example, WINSAL may thread by department or store. By choosing the store option and assuming 100 stores processed with a thread value of 5, an estimate of 20 stores per thread is derived.

Locking Strategy

In conjunction with the Performance and the Scheduling Considerations section, this section should describe the locking (and release) strategy required beyond the preset Retek standards. It should describe how the module accesses data and the 'hold' or 'lock' it has on a database and / or its records, during processing. It should also describe the 'lock' release.

Restart / Recovery

The Logical Unit of Work (LUW) is drawn from the Functional Level Description (above). Specifically, what is the elementary transaction performed. The LUW defines the commit processing and must not compromise database integrity. A given transaction may consist of several database modifications, but any one of these transaction sections is invalid unless all are processed.

For example, consider Receiving. Order-received quantities, shipment quantities, perpetual inventory and post financial transactions must be updated. If any one portion is not process, the transactions are invalid, therefore the LUW is defined as processing the receipt of a given SKU / Location quantity for a given order.

The driving query must be defined based on the LUW to ensure that the system can recover to a particular LUW break point and not allow partial transactions posted to a database. The driving query is the heart of the batch module. Only one driving query per module is allowed. Information required from the driving tables, should be selected, since this is the only query operation that will be performed on these tables. Restart recovery logic must be added to the driving query based on the defined LUW.

This program will be restartable but not threadable.

Performance

In conjunction with the Scheduling Considerations and Locking Strategy sections, the optimization considerations of a batch module must adhere to Retek standards. This section should call out special performance considerations that may exceed current documented Retek practices. Such considerations should be the basis for update to Retek standards. Each database operation should be optimized based on quantity and quality of the database transactions. Batch modules are executed on the database or dedicated batch server and thus there are no additional performance gains to forcing database interaction logic onto the server.

Security Considerations

Highlight any batch security design exceptions / options.

Design Assumptions

Design assumptions are presumed design factors, inferred from current information, expected to hold true over the life of the project. Design assumptions must be documented in order to justify and validate derived design considerations with the Business Requirements (documented within the BRD and PFO).

Letter of credit upload [lcupld]

Design Overview

The purpose of this new Letter of Credit upload program, which will run nightly, is to upload confirmations, drawdowns, and bank fees and charge information returned from an Issuing Bank(s) regarding Letter of Credits. This program should run during the standard upload phase of the batch cycle. The information will be read in using the standard Retek flat file format. A translation script will have previously converted the information to a standard flat file from multiple files formatted in MT 730, which is one of several S.W.I.F.T. formats used by banks to transfer financial information regarding LCs. S.W.I.F.T. formats are used when transmitting financial information and are a globally recognized set of standards. After this program has processed a confirmation/charge/drawdown, the appropriate tables will be updated; a confirmation will update the LC to confirmed status and drawdowns and bank charges will write the appropriate records to the LC_ACTIVITY table.

The lcupld program needs to be modified to insert total amount and currency information into the lc_activity table.

Previously, this program was used to upload confirmations, drawdowns, bank fees, and charge information. The drawdowns and charges will now be handled by lcmt798 (Perl script) and lcup798.pc (batch program); lcupld.pc will only be used for LC confirmations.

Affected Tables:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
LC HEAD	Yes	Yes	No	Yes	No
LC_ACTIVITY	Yes	No	Yes	No	No

Scheduling Constraints

Processing Cycle: Standard Upload Phase (II)

This program must be run after the lcmt730 Perl script, as it uses the output file from that script. It will be run in Phase 2 along with other upload programs

Scheduling Diagram: N/A

Pre-Processing: N/A

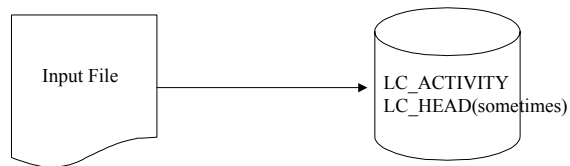
Post-Processing: N/A

Threading Scheme: No threading due to low volume.

Restart Recovery

Restart/recovery for this program is set up at the individual FDETL record. Although there may be more than one FDETL record for a given LC, they will each be processed as a separate entity. The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Program Flow



Shared Modules

CURRENCY_SQL.CONVERT

currency_sql.convert will no longer be used.

Function Level Description

Init():

This function performs preliminary processing and populates global variables. It will call `retek_init()` to start up restart-recovery and bring back the bookmark string in case of a restart. It will also populate `vdate`, to be used when update the `lc_head.confirmed_date`. Create and open a reject file for write. Open input file for read. Call `get_record` to read the FHEAD into the `header_struct`.

Process():

This function drives the processing of the program. Call the function `retek_get_record` to read the FDETL line into the `detail_struct`. If the record type returned is FTAIL, exit the loop. If the record type returned is other than FDETL or FTAIL then return a fatal error (wrong record type). If the record type is FDETL then call the function `validate_variables()`, passing in the FDETL variables, to validate and then insert the variables parsed above into the correct fields on the LC_ACTIVITY table.

If the `upload_type = 'L'` (Confirmation of an LC), call `update_lc_head()`. If `validate_variables()` return a non_fatal error, call `retek_write_rej()` to write reject records.

After reading and processing `commit_max_ctr` number of Letter of Credit acknowledgement records, call `retek_force_commit()` to perform intermittent commit for restart/recovery.

Final():

This function will call `retek_close()` to wrap up restart/recovery, populate the final error message, if necessary, and close the input and reject files.

Init_array():

This function initializes the size of the array used for holding the input records at the size of the restart max counter on restart_control.

Validate_variables():

This function basically consists of a number of function calls to validate the variables that will later, if they pass validation, be inserted into LC_ACTIVITY.

Call check_upload_type(), passing in the upload_type variable.

Call check_lc_refs(), passing in the upload_type, the bank_lc_id, the lc_ref_id, and the issuing_bank variables. Fetch back lc_head.rowid if the record is valid.

Call check_date(), passing in the activity_date.

Call check_amount(), to ensure it is all numeric, passing in the amount.

Call check_currency(), passing in the lc_ref_id, the currency_code and the amount.

Check_upload_type():

This function takes in the upload_type and ensures that it is 'B', 'D', or 'L'; reject record if it is not. Note: these three codes are being used temporarily until it is confirmed that these are the only three types of MT 730 that will be coming in.

Check_lc_refs():

This function takes in the upload_type, bank_lc_id, the lc_ref_id and the issuing_bank. If the upload_type is 'B' or 'D', then check if both the bank_lc_id and the lc_ref_id are valid and exist on the same record on lc_head for a 'C' onfirmed Letter of Credit. If the upload_type is 'L', then check if only the lc_ref_id exists on lc_head (as the bank_lc_id is being sent with this confirmation and will not be present on the table until after this program runs), the function will also check lc_head.issuing_bank with the passing in issuing_bank. Reject the record if the check fails, return lc_head.rowid if the check passes.

Check_date():

This function check that the date is a valid date (that the date exists, 063199 is an example of a date that would not be valid as it does not exist) by calling valid_date(), a library function. Note: Not sure if we insert the date sent in the MT 730 into lc_activity or use the vdate instead?

Check_amount():

This function takes in the amount and calls the library function valid_all_numeric() to ensure that the amount is all numeric. If not reject the record.

Check_currency():

This function takes in the currency and the amount and checks if it exists on the CURRENCY table. If it exists on the currency table, check if it matches the LC_HEAD.CURRENCY_CODE for that lc_ref_id. If the currency does not match the currency on lc_head then use CURRENCY_SQL.CONVERT to convert the amount to the LC_HEAD currency. Pass in the amount, the currency, the currency off of lc_head as the I_currency_out, the I_cost_retail_ind as 'C', and then NULL for the last four input variables. Reject the record if the package fails. Note: The I_exchange_type will perhaps be entered as 'LC', although this logic is not in place right now.

Update_lc_head():

This function will update the confirmed_date of the Letter of Credits to vdate, the bank_lc_id to the input value if the upload_type = 'L'. This is by rowid.

Insert_records():

This function will insert all the valid records to LC_ACTIVITY through activity_struct array.

In the prototype section, include variable names in the functions.

Throughout the program, remove unused variables (that came from fields on the input file that have been removed).

Since we will only ever be getting confirmations, we will remove the upload_type, check_amount, and check_currency from the file and the program. The update to the lc_head table should always be done. The check_upload_type function should be removed. The code in check_lc_refs that does stuff if the upload type is 'B' or 'D' can be removed.

For each confirmation received, we need to insert a record into the lc_activity table that contains the total amount on the letter of credit (from lc_head), the currency code (from lc_head) and the exchange rate (from lc_head). The trans_code should be inserted as 'L' and the activity date should come from the SWIFT file.

Since the amount will come from the lc_head table, it will never need to be converted and the call to CURRENCY_SQL.CONVERT can be removed.

I/O Specification

Input file:

Original Version:

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	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	LCUP	Identifies file as 'Letter of Credit Upload'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check.
	Sender's Reference	Char(16)	lc_head.bank_lc_id	The LC number that the bank assigns to a Letter of Credit. SWIFT allows for up to 16

				characters. . . we might have to truncate in the conversion script? Does it matter? Maybe make this field Char(16)?
	Receiver's Reference	Number(8)	lc_activity.lc_ref_id	The LC number that Retek assigned to the Letter of Credit. Might have to to_number it in the conversion file?
	Upload Type	Char(1)	lc_activity.trans_code	Determines what kind of message this individual FDETL is: L = Confirmation of LC B = Bank Charge/Fee D = Drawdown
	Date of Message Being Acknowledged	Char(14)	lc_activity.activity_date	If the upload type is 'L' then this date will match the date MT 700 date of issue (which we have not resolved between being the vdate or the lc_head.application_date) If the upload type is B or D no idea what to do? What if (for a charge) no message was sent . . . what would this date be? <u>Perhaps it does not matter which date is what. . just put the date in the activity_date field on LC_ACTIVITY.</u> formatted to 'YYYYMMDDHH24MISS'
	Currency of Charge	Char(3)	currency code	Should match lc_head.currency_code for that LC.
UNECESSARY????	Method of Payment	Char(1)	method of payment	Not and RMS field. . . if D, then then Amount of Charge will be debited from the account (line of credit). . . if B, then ? (a bill sent or something?) Can be B or D. May be unnecessary as the upload type is basically the same thing.
	Amount of Charge (or should this be Amount of Drawdown?)	Number (20,4)	lc_activity.amount	If method of payment = 'D', the amount that will be debited from the credit line (the LC) either for bank charges(? – are charges debited from the account or not?)

				or drawdowns.
	Amount of Bill (or should this be Amount of Charge?)	Number (20,4)	???	If method of payment = 'B' then do we have some sort of field like this. . . that is processed somewhere else besides LC ACTIVITY.
	Account With Bank	Char(10)	lc_head.issui ng_bank	Assumption that this will always be the issuing bank? Should match lc_head.issuing_bank?
	Comments	Char(2000)	lc_activity.co mments	Need to truncate? This field will probably be a concatenation of the following SWIFT fields: 71B – Charges, 72 – Sender information.
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
	Total number lines	Number(10)	Specified by external system	Total number lines in file

Changed Version:

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	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	LCUP	Identifies file as 'Letter of Credit Upload'
	File Create Date	Char (14)	vdate	date file was written by external system 'YYYYMMDDHH24MISS' format
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	Sender's Reference	Char(16) (was 15)	lc_head.bank ld_id	The LC number that the bank assigns to a Letter of Credit.
	Receiver's Reference	Number(8)	lc_activity.lc ref_id	The LC number that Retek assigned to the Letter of Credit.

	Date of Message Being Acknowledged	Date (char 8)	lc_activity.activity_date	If the upload type is 'L' then this date will match the date MT 700 date of issue (which we have not resolved between being the vdate or the lc_head.application_date) 'YYYYMMDD' format
	Comments	Char(2000)	lc_activity.comments	Need to truncate? This field will probably be a concatenation of the following SWIFT fields: 71B – Charges, 72 – Sender information.
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
	Total number lines	Number(10)	Specified by external system	Total number lines in file

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the letter of credit upload 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.

Stock upload conversion [lifstkup]

Design Overview

This program converts the Nautilus inventory balance upload file into the Retek standard flat file for stkupld.pc to process.

This program verifies that the inventory data is for the requested cycle count and warehouse before proceeding. Other data needed for the RMS flat file will be obtained from RMS tables and inserted in the RMS flat file.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should run before stkupld.pc and after the Nautilus inv_bal_upload.sh program.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

Init()

Obtain the cycle_count from tables stake_head and stake_location by matching the warehouse location and the stocktake_date with combination of the Nautilus input file, location(DC) and the transaction date/time (date portion only).

If no match can be made, the program will terminate.

Call restart init

Write FHEAD record.

Process()

Loop through and read each line from the Nautilus file.

Format the data for the output file.

Call write_fdetl() function.

Write_fdetl()

Call check_catch_weight() function

Write a transaction record (FDETL) to the Retek flat file for each line found in the Nautilus flat file.

Check_catch_weight()

Called within write_fdetl to check if item is check weight Copies total weight value if it is a check weight item, otherwise copies total units.

Final ()

Write a file trailer record (FTAIL).

Close files

Call restart close

Write status to log file

I/O Specification

Both input and output file should be accepted as a runtime parameter at the command line.

Input format:

DC_DEST_ID	11 - Number(10) + 1 for trailing space.	Unique identifier for the DC
TRANSACTION_DATE	13 - Date(12) + 1 for trailing space.	Date of the run
ITEM_ID	26 - Varchar2(25) + 1 for trailing space	Uniquely identifies the item.
AVAILABLE_QTY	10 - Number(8) + 1 for leading sign and + 1 for trailing space.	Units available for distribution
DISTRIBUTED_QTY	9 Number(8) + 1 for trailing space.	Units distributed include: Units distributed but not yet picked, units picked but not yet manifested, units manifested but not yet shipped.
RECEIVED_QTY	9 Number(8) + 1 for trailing space.	Units received but not put away.
TOTAL_QTY	9 Number(8) + 1 for trailing space.	Sum of all units that physically exist: container status of: I, D, M, R, T, X.
AVAILABLE_WEIGHT	15 - Number(12,4) + 1 for leading sign + 1 for decimal + 1 for trailing space.	Weight available for distribution of catch weight items.
RECEIVED_WEIGHT	14 - Number(12,4) + 1 for decimal + 1 for trailing space.	Weight received but not put away for catch weight items.
DISTRIBUTED_WEIGHT	14 - Number(12,4) + 1 for decimal + 1 for trailing space.	Weight distributed includes: weight distributed but not yet picked, weight picked but not yet manifested, weight manifested but not yet shipped (value only catch weight items)
TOTAL_WEIGHT	14 - Number(12,4) + 1 for decimal + 1 for trailing space.	Sum of all weight that physically exist: container status of: I, D, M, R, T, X. For catch weight items.

Output Format:

Record Name	Field Name	Field Type	Description
File Header	file type record descriptor	Char(5)	hardcode 'FHEAD'
	file line identifier	Number(10)	Id of current line being processed., hardcode '000000001'
	file type	Char(4)	hardcode 'STKU'
	file create date	Date(14) YYYYMMDDHHMIS S	date written by convert program
	stocktake_date	Date(14) YYYYMMDDHHMIS S	stake_head.stocktake_date
	cycle count	Number(8)	stake_head.cycle_count
	loc_type	Char(1)	hardcode 'W'
	location	Number(10)	stake_location.wh
Transaction record	file type record descriptor	Char(5)	hardcode 'FDETL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	item type	Char(3)	hardcode 'ITM'
	item value	Number(25)	item id
	inventory quantity	Number(12,4)	total units or total weight
	location description	Char(30)	NULL
File trailer	file type record descriptor	Char(5)	hardcode 'FTAIL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	file record count	Number(10)	Number of detail records.

On order download [onorddnld]

Functional Area

RMS to RPP interface

Module Affected

onorddnld.pc

Design Overview

This program sends on order cost, retail, and qty at the item/location/week level to Retek Predictive Planning (RPP). The values are used by RPP to generate open to buy numbers that are interfaced back into RMS.

This is the second part of a two part process to calculate and export the on order numbers. On order extract (onordext) first hits the ordering tables and breaks out the on order numbers. It inserts these values into ON_ORDER_TEMP. These values are at the item/location/week/order_no level. Onorddnld.pc then picks up the records on ON_ORDER_TEMP, groups them by item/location/week and writes them to an output file to be exported to RPP.

Stored Procedures / Shared Modules (Maintainability)

Input/Output Specifications

Program Flow

Function Level Description

Init()

Initialize restart recovery and file processing.
Setup the output file.

Process()

Define driving cursor here:

This program should select ITEM, OTB_EOW_DATE, STORE, WH, LOC_TYPE, ON_ORDER_RETAIL, ON_ORDER_COST, and ON_ORDER_QTY. ON_ORDER_RETAIL, ON_ORDER_COST, and ON_ORDER_QTY should be summed up across ITEM/OTB_EOW_DATE/STORE/WH groups. The cursor should bring back 'S' if the store is populate and 'W' if the wh field is populated for the LOC_TYPE. This program should be threaded by store/wh, use the V_RESTART_STORE_WH.

Call size_array() and define_buffer() .

Use array processing to fetch the driving cursor

For each record brought back by the driving cursor, call write_to_file().

Call retek_force_commit() once commit_max_counter records have been processed.

Call `free_array()` after the processing is finished.

Define_buffer()

Set up a string with the `fprintf` format that will be used when writing records to the output file.

Size_array()

Size the driving cursor fetch array to the commit max counter.

Free_array()

Give back the memory allocated by `size_array()`.

Write_to_file()

Print out the item, location, `loc_type`, `otb_eow_date`, `on_order_retail`, `on_order_cost`, and `on_order_qty` to the output file.

Final()

Clean up restart recovery and file processing.

Input Specifications

'Table-To-Table'

This program sources `ON_ORDER_TEMP`. All records on `ON_ORDER_TEMP` are selected, the cost and retail are summed up to the item/location/week level.

Output Specifications

'Table-To-Table'

Output Files

Record Name	Field Name	Field Type	Default Value	Description
	Item	CHAR(25)		RMS Item Identifier (left justified)
	Location	NUMBER (20)		Store or Warehouse Identifier (left justified)
	Loc Type	CHAR(1)		Indicates if the location is a store or a warehouse: S – if the location is a store, W – If the location is a warehouse
	EOW Date	DATE(8)		The OTB end of week date
	On Order Retail	NUMBER(25)		Total extended retail for the item/location/EOW date. Right justified, decimal point is included in output file

	On Order Cost	NUMBER(25)		Total extended cost for the item/location/EOW date. Right justified, decimal point is included in output file
	On Order Qty	NUMBER(17)		Total qty for the item/location/EOW date. Right Justified, decimal point is included in output file.

Scheduling Considerations

Should run after onordext.pc in phase 4.

Locking Strategy

This program has no DML statements -- no lock strategy is required.

Restart/Recovery

This program has a unique logical unit of work (LUW) of item/location/otb_eow_date.

Performance Considerations**Security Considerations****Unit Test Considerations****Design Assumptions**

This program doesn't need to split its output by domain.

On order extract [onordext]

Design Overview

This program calculates the value in cost and retail of items that are on order for the department/class/subclass/location level. This program is the first step in the stock ledger download process to RPP. It calculates the on order cost and retail for all approved orders that have not before dates less than or equal to the planning horizon date. Once the program has calculated the costs and retails, they are inserted into the ON_ORDER_TEMP table. This table is used in the second step of the stock ledger download process -- the Stock Ledger Extract program (stlgnld.pc).

IMPORTANT NOTE:

The MAX_BUFFER_NUM #define should be set to a number that will allow safe buffering of component pack items, allocations, and orders at the component sku location level. If it is possible for any of these variable to exceed the value assigned to MAX_BUFFER_NUM, its value must be increased.

Stored Procedures / Shared Modules (Maintainability)

Convert_to_primary – Pro*c library (utils.h / utils.pc).

Program Flow

Function Level Description

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

init():

Initialize restart recovery by calling retek_init(). Opens the input file and reads the planning horizon end date. Gets the primary currency from the system_options table.

process():

There are two process loops, one for transaction level items and one for pack items.

For each record brought back by the item driving cursor:

Convert the cost from the order currency into the primary currency by calling the convert_to_primary function.

Convert the retail from the location currency into the primary currency by calling the convert_to_primary function.

Call the handle_alloc function.

After all the record from the item cursor have been processed. The process flag should be flipped to 'P'ack. The handle_alloc function is called to clean up the last allocation.

For each record brought back by the pack driving cursor:

Convert the pack cost from the order currency into the primary currency by calling the convert_to_primary function.

Convert the pack retail from the location currency into the primary currency by calling the convert_to_primary function. convert the pack component item cost from the supplier currency into the primary currency by calling the convert_to_primary function.

Convert the pack component item retail from the zone/zone_group currency into the primary currency by calling the `convert_to_primary` function.
Call the `handle_pack` function.

Handle_insert()

This function adds records to arrays that will be inserted into `ON_ORDER_TEMP`. First it populated a record into the regular array (non-allocation). Then if the current record is associated with an allocation, a record is added to the allocation array.

Add_insert_record()

This function copies records from a driving cursor record into an insert array. It has two modes, the first if for non-allocation records. The cost and retail are set to the converted cost and retail from the driving cursor times the item qty from `ordloc`. The second mode handles allocation records, it sets the cost and retail to the converted cost and retail from the driving cursor times the `alloc qty` from `alloc_detail`.

Handle_alloc()

This function sums up the quantities allocated to on `alloc_header` for each allocation. Once every record associated with an allocation have been summed, it is possible to determine how much of the qty ordered to the allocation warehouse should be assigned to it. If all of the items ordered to it have been allocated, the warehouse is not assigned any of the cost or retail. If items are not fully allocated, the warehouse is assigned the items that were not allocated. Once the qty to give to the warehouse it determined, the `handle_insert` is function is called for each allocation record.

Handle_pack()

This function prorates the amount and cost ordered through packs to their component items. Every component item ordered through a pack has their total cost and retail summed up. Once all the component items have been summed, the cost and retail of the pack is prorated. The formula cost is: (the pack's cost / the summed component cost) * the individual component item's cost. The same formula is used for retail.

After the records are prorated they are sent to `buffer_alloc()`.

Buffer_alloc()

This function is used to reorder the records sent to it by `handle_pack()` before they are sent to `handle_alloc()`. The `handle_pack()` function requires that the records be ordered by `order_no`, `pack_no`, `location`, `alloc location`. `Handle_alloc()` requires the records to be ordered by `order_no`, `pack_no`, `location`, `component item`. This function build groups of `order_no/pack_no/location` and reorders them by component item. **`sort_for_allocation()`**, **`get_max_indx()`**, **`swap_driv_array()`**, and **`copy_info()`** are utility functions that help sort the driving cursor records.

Once the buffered records have been ordered, they are sent to the `handle_alloc()` function.

Copy_fetch()

Utility function used by `handle_alloc` and `handle_pack`. It totals up the allocated qty for the `handle_alloc` function. It totals up the components cost and retail (times the pack item qty) for `handle_pack`.

Insert_on_order_temp()

This function performs an array insert into the `on_order_temp` table.

Size_driv_cur_struct()

Allocates memory for the structure used to fetch the driving cursors and used when summing up allocations and pack component items.

Size_insert_struct()

Allocates memory used to perform bulk inserts into the on_order_temp table.

Input Specifications**Command Line Parameters:**

Onordext will calculate the on order cost and retail for a given planning horizon. The planning horizon date will be a command line parameter contained in an input file. The file will also contain information used by stlgdnld.pc.

The file contains 1 line

Field Position: 1 – weekly or historic indicator

2-9 – planning horizon start date

10-17 – planning horizon end date

Onordext userid/passwd input-file

Driving Cursor:

There are two driving cursors in this program. The first deals with items, the second deals with packs.

```

/*
  The first driving cursor handles on-order amounts for items.
  The first part deals with non-allocation ordeers or allocation orders
  that are not pre-marked
*/
EXEC SQL DECLARE c_driver CURSOR FOR
  SELECT  oh.order_no,
          -999 alloc_no,
          ol.item,
          ol.qty_ordered - nvl(ol.qty_received,0),
          ol.unit_cost,
          ol.unit_retail,
          ol.location,
          ol.loc_type,
          -999 alloc_loc,
          'N' alloc_loc_type,
          0 alloc_qty,
          im.dept,
          im.class,
          im.subclass,
          oh.exchange_rate,
          oh.currency_code,
          oh.oth_eow_date,
          1 comp_item_qty
  FROM    ordhead oh,
          ordloc ol,
          item_master im
  WHERE   oh.status   = 'A'
          AND ol.qty_ordered > nvl(ol.qty_received,0)
          AND oh.not_before_date <= TO_DATE(:ps_on_order_date, 'YYYYMMDD')
          AND oh.order_no = ol.order_no
          AND ol.item     = im.item
          AND im.pack_ind = 'N'
          AND (oh.pre_mark_ind = 'N' OR

```

```

        (oh.pre_mark_ind = 'Y'
        and not exists (select alloc_no
                        from alloc_header ah
                        where ah.order_no = oh.order_no
                        and ol.item      = ah.item
                        and ol.location = ah.wh)))
    AND oh.order_no > NVL(:ps_restart_order, -999)
    AND MOD(oh.order_no, :pi_restart_num_threads) + 1 = :pi_restart_thread_val
UNION ALL
SELECT oh.order_no,
       NVL(ah.alloc_no,-999),
       ol.item,
       ol.qty_ordered - nvl(ol.qty_received,0),
       ol.unit_cost,
       ol.unit_retail,
       ol.location,
       ol.loc_type,
       ad.to_loc,
       ad.to_loc_type,
       nvl(ad.qty_allocated, 0) - nvl(ad.qty_transferred,0),
       im.dept,
       im.class,
       im.subclass,
       oh.exchange_rate,
       oh.currency_code,
       oh.otb_eow_date,
       1 comp_item_qty
FROM   ordhead oh,
       ordloc ol,
       item_master im,
       alloc_header ah,
       alloc_detail ad
WHERE  oh.status   = 'A'
      AND ol.qty_ordered > nvl(ol.qty_received,0)
      AND oh.pre_mark_ind = 'Y'
      AND oh.not_before_date <= TO_DATE(:ps_on_order_date, 'YYYYMMDD')
      AND oh.order_no = ol.order_no
      AND ol.item      = im.item
      AND im.pack_ind = 'N'
      AND ol.order_no = ah.order_no
      AND ol.location = ah.wh
      AND ol.item      = ah.item
      AND ah.alloc_no = ad.alloc_no
      AND oh.order_no > NVL(:ps_restart_order, -999)
      AND MOD(oh.order_no, :pi_restart_num_threads) + 1 = :pi_restart_thread_val
ORDER BY 1,2,7,9;

```

```

/*
The second driving cursor handles on-order amounts for packs.
- The first part deals with packs being ordered directly to a store.
- The second part deals with packs being directly ordered to a wh or
  allocated from a wh with the pre-mark indicator set to 'N'.
- The third part deals with packs being allocated from a wh with the
  pre-mark indicator set to 'Y'.

```

```

*/
EXEC SQL DECLARE c_pack_driver CURSOR FOR
SELECT oh.order_no,
       -999 alloc_no,
       ol.item pack_no,
       ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
       ol.unit_cost pack_cost,
       ol.unit_retail pack_retail,
       vpq.item,
       vpq.qty item_qty,
       iscl.unit_cost comp_cost,
       izp.unit_retail comp_retail,
       ol.location,
       ol.loc_type,
       -999 alloc_loc,
       'N' alloc_loc_type,
       0 alloc_qty,

```

```

        im.dept,
        im.class,
        im.subclass,
        iscl.supplier,
        izp.zone_id,
        izp.zone_group_id,
        oh.exchange_rate,
        oh.currency_code,
        oh.otb_eow_date,
        vpq.qty comp_item_qty
FROM   ordhead oh,
       ordsku os,
       ordloc ol,
       item master im,
       v_packsku_qty vpq,
       item_supp_country_loc iscl,
       price_zone_group_store pzgs,
       item_zone_price izp
WHERE  oh.status          = 'A'
      AND ol.qty_ordered > nvl(ol.qty_received,0)
      AND oh.not_before_date <= TO_DATE(:ps_on_order_date, 'YYYYMMDD')
      AND oh.pre_mark_ind  = 'N'
      AND oh.order_no      = os.order_no
      AND oh.order_no      = ol.order_no
      AND ol.loc_type      = 'S'
      AND os.item          = ol.item
      AND ol.item          = vpq.pack_no
      AND im.item          = vpq.item
      AND iscl.item        = im.item
      AND oh.supplier      = iscl.supplier
      AND os.origin_country_id = iscl.origin_country_id
      AND pzgs.store       = ol.location
      AND pzgs.zone_id     = izp.zone_id
      AND pzgs.zone_group_id = izp.zone_group_id
      AND izp.item         = vpq.item
      AND oh.order_no > NVL(:ps_restart_order, -999)
      AND MOD(oh.order_no, :pi_restart_num_threads) + 1 = :pi_restart_thread_val
UNION ALL
SELECT oh.order_no,
       -999 alloc_no,
       ol.item pack_no,
       ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
       ol.unit_cost pack_cost,
       ol.unit_retail pack_retail,
       vpq.item,
       vpq.qty item_qty,
       iscl.unit_cost comp_cost,
       izp.unit_retail comp_retail,
       ol.location,
       ol.loc_type,
       -999 alloc_loc,
       'N' alloc_loc_type,
       0 alloc_qty,
       im.dept,
       im.class,
       im.subclass,
       iscl.supplier,
       izp.zone_id,
       izp.zone_group_id,
       oh.exchange_rate,
       oh.currency_code,
       oh.otb_eow_date,
       vpq.qty comp_item_qty
FROM   ordhead oh,
       ordsku os,
       ordloc ol,
       item master im,
       v_packsku_qty vpq,
       item_supp_country_loc iscl,
       item_zone_price izp
WHERE  oh.status          = 'A'

```

```

AND ol.qty_ordered > nvl(ol.qty_received,0)
AND oh.not_before_date <= TO_DATE(:ps_on_order_date, 'YYYYMMDD')
AND oh.order_no      = os.order_no
AND oh.order_no      = ol.order_no
AND os.item          = ol.item
AND ol.loc_type      = 'W'
AND ol.item          = vpq.pack_no
AND im.item          = vpq.item
AND iscl.item        = im.item
AND oh.supplier      = iscl.supplier
AND os.origin_country_id = iscl.origin_country_id
AND izp.item         = vpq.item
AND izp.base_retail_ind = 'Y'
AND (oh.pre_mark_ind = 'N' OR
     (oh.pre_mark_ind = 'Y'
      and not exists (select alloc_no
                       from alloc_header ah
                       where ah.order_no = oh.order_no
                             and ol.item   = ah.item
                             and ol.location = ah.wh)))
AND oh.order_no > NVL(:ps_restart_order, -999)
AND MOD(oh.order_no, :pi_restart_num_threads) + 1 = :pi_restart_thread_val
UNION ALL
SELECT oh.order_no,
       NVL(ah.alloc_no,-999),
       ol.item pack_no,
       ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
       ol.unit_cost pack_cost,
       ol.unit_retail pack_retail,
       vpq.item,
       vpq.qty item_qty,
       iscl.unit_cost comp_cost,
       izp.unit_retail comp_retail,
       ol.location,
       ol.loc_type,
       ad.to_loc,
       ad.to_loc_type,
       nvl(ad.qty_allocated, 0) - nvl(ad.qty_transferred,0),
       im.dept,
       im.class,
       im.subclass,
       iscl.supplier,
       izp.zone_id,
       izp.zone_group_id,
       oh.exchange_rate,
       oh.currency_code,
       oh.otb_eow_date,
       vpq.qty comp_item_qty
FROM   ordhead oh,
       ordsku os,
       ordloc ol,
       item_master im,
       v_packsku_qty vpq,
       item_supp_country_loc iscl,
       item_zone_price izp,
       alloc_header ah,
       alloc_detail ad
WHERE  oh.status      = 'A'
AND    ol.qty_ordered > nvl(ol.qty_received,0)
AND    oh.pre_mark_ind = 'Y'
AND    oh.not_before_date <= TO_DATE(:ps_on_order_date, 'YYYYMMDD')
AND    oh.order_no      = os.order_no
AND    oh.order_no      = ol.order_no
AND    os.item          = ol.item
AND    ol.loc_type      = 'W'
AND    ol.item          = vpq.pack_no
AND    im.item          = vpq.item
AND    iscl.item        = im.item
AND    oh.supplier      = iscl.supplier
AND    os.origin_country_id = iscl.origin_country_id
AND    izp.item         = vpq.item

```

```

AND izp.base_retail_ind = 'Y'
AND ol.order_no         = ah.order_no
AND ol.location         = ah.wh
AND ol.item             = ah.item
AND ah.alloc_no         = ad.alloc_no
AND oh.order_no > NVL(:ps_restart_order, -999)
AND MOD(oh.order_no, :pi_restart_num_threads) + 1 = :pi_restart_thread_val
ORDER BY 1,2,3,11,13;

```

Output Specifications

ON_ORDER_TEMP will be populated by this program.

ITEM	NOT NULL VARCHAR2(25)
DEPT	NOT NULL NUMBER(4)
CLASS	NOT NULL NUMBER(4)
SUBCLASS	NOT NULL NUMBER(4)
OTB_EOW_DATE	NOT NULL DATE
STORE	NOT NULL NUMBER(10)
WH	NOT NULL NUMBER(10)
ON_ORDER_RETAIL	NOT NULL NUMBER(20,4)
ON_ORDER_COST	NOT NULL NUMBER(20,4)
ON_ORDER_UNITS	NOT NULL NUMBER(20,4)

Scheduling Considerations

This program can be run weekly in Phase 4.

Prepost onordext pre must run before this program.

This program should be run before onoroddnld.pc.

Locking Strategy

Restart/Recovery

Logical unit of work (LUW) is a unique order number. This is a non-unique LUW.

It is also split into two sections item and pack. First all items on orders are processed. When they are done a pack 'flag' is turned on and the restart order is reset. Then all the packs on order are processed. So all orders are considered twice, once for items and once for packs.

Order deal discount [orddsent.pc/h]

Design Overview

This new batch program will fetch orders that need to be recalculated for cost from DEAL_CALC_QUEUE. Using the dealordlib shared library, it will update the unit cost on the argument order along with potentially populating ORDLOC_DISCOUNT, and ORDHEAD_DISCOUNT. This program is a batch-end wrapper for the library. For an in-depth description of what happens inside the library, please refer to dealordlib Master Batch Library Design document.

Stored Procedures / Shared Modules (Maintainability)

Dealordlib.pc/.h

Program Flow

```
main()
```

Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init(): Standard Retek init function. Calls retex_init(), and the library's lib_init().

Process(): Drives the rest of the program:

- Call size_order_array(), size_deal_array(), size_disct_bld_array() and size_disc_item_array() from the dealordlib library.
- Array-fetch the driving cursor.
- Call check_lock_and_validate() from the dealordlib library.
- For each fetched record:
 - Call reset_order() from the dealordlib library.
 - Call get_deals() from the dealordlib library.
 - Call get_deal_ct() from the dealordlib library to see how many deals were found. If none, move to next order.
 - Call make_ordloc_build() from the dealordlib library.
 - Call buyer_packs_detected() from the dealordlib library to see if make_ordloc_build() found any buyer packs on the ORDLOC table. If number returned is higher than 0, write a warning.
 - If the order is approved, call get_otb_cost() to get the pre-deals OTB cost.
 - Call find_items_for_deal() to create a combination data between deals and items for this order (deal + item deal applies to).
 - Call calculate_cost() to calculate new costs.
 - If the order is approved and there were transaction level deals found for the order, call process_txn_discount() to apply those deals. Also call get_otb_cost() to get the post-deals OTB cost. Then call update_otb() to make sure the change in costs is communicated to OTB. Also call insert_rev_orders() from the library to make sure rev_orders table is properly inserted if costs changed on an approved order.

- Call free_item_cost_array() from the library.
- Call delete_ordloc_disc_bld() to clean the ORDLOC_DISCOUNT_BUILD staging table.
- Call cleanup_discount_tables() to delete unnecessary data.
- Call delete_dcq() to delete the DEAL_CALC_QUEUE record that was just processed.
- Call retek_force_commit() to commit work.

Final():

Performs restart/recovery close logic. Calls retek_close().

Input Specifications**'Table-To-Table'**

DEAL_CALC_QUEUE			
DEAL_CALC_QUEUE			
DEAL_CALC_QUEUE			
ORDHEAD			
ORDHEAD			
ORDHEAD			
ORDHEAD			
ORDHEAD			
ORDHEAD			
ORDHEAD			
SUPS			
SUPS			

Output Specifications**'Table-To-Table'**

See dealordlib Master Batch Library Design.

Scheduling Considerations

This program should run after ditinsrt before dealcls or dealprg in the deals batch cycle.

Locking Strategy

See dealordlib Master Batch Library Design.

Restart/Recovery

This program has inherent restartability, since records are deleted from deal_calc_queue as they are processed.

Recommended maximum commit counter: --- (low—will be doing lots and lots of stuff for each order...)

Performance Considerations**Security Considerations**

Order purge [ordprg]

Design Overview

The purpose of this module is to remove old orders from the system.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'N' and if invoice matching is not installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). Orders are deleted only if shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'Y' and if invoice matching is not installed, then all details associated with the order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status), and as long as all shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

This program will also create a PO header flat file to interface with the Nautilus system. When orders are deleted, a record with the action type = 'D'leted will be written to an output file. Nautilus will then process this file and delete the PO from the warehouse's database to maintain consistency between the host and warehouse environment.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALC_COMP_LOC	No	No	No	No	Yes
ALC_HEAD	No	Yes	No	No	Yes
ALLOC_CHRG	No	No	No	No	Yes
ALLOC_DETAIL	No	No	No	No	Yes
ALLOC_HEADER	Yes	No	No	No	Yes
APPT_HEAD	No	Yes	No	No	Yes
APPT_DETAIL	No	Yes	No	No	Yes
CE_LIC_VISA	No	No	No	No	Yes
CE_CHARGES	No	No	No	No	Yes
CE_ORD_ITEM	No	Yes	No	No	Yes
CE_SHIPMENT	No	No	No	No	Yes
CE_HEAD	No	No	No	No	Yes
DOC_CLOSE_QUEUE	No	No	No	No	Yes
INVC_DETAIL	Yes	No	No	No	Yes
INVC_HEAD	Yes	Yes	No	No	Yes
INVC_MATCH_WKSH T	Yes	No	No	No	Yes

INVC_MATCH_QUEUE	Yes	No	No	No	Yes
INVC_MERCH_VAT	Yes	No	No	No	Yes
INVC_NON_MERCH	Yes	No	No	No	Yes
INVC_XREF	Yes	No	No	No	Yes
OBLIGATION	No	No	No	No	Yes
OBLIGATION_COMP	No	No	No	No	Yes
OBLIGATION_COMP_LOC	No	No	No	No	Yes
ORDCUST	No	No	No	No	Yes
ORDHEAD_DISCOUNT	No	No	No	No	Yes
ORDHEAD	Yes	Yes	No	No	Yes
ORDLOC	Yes	No	No	No	Yes
ORDSKU_INVC_COST	No	No	No	No	Yes
ITEM_MASTER	Yes	No	No	No	Yes
LC_ORDAPPLY	No	No	No	No	Yes
DEAL_HEAD	No	No	No	No	Yes
DEAL_DETAIL	No	No	No	No	Yes
DEAL_ITEMLOC	No	No	No	No	Yes
DEAL_THRESHOLD	No	No	No	No	Yes
DEAL_QUEUE	No	No	No	No	Yes
DEAL_CALC_QUEUE	No	No	No	No	Yes
ORD_INV_MGMT	No	No	No	No	Yes
REPL_RESULTS	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
QC	Yes	No	No	No	Yes
RTV_DETAIL	No	No	No	No	Yes
SHIPMENT	Yes	Yes	No	No	Yes
SHPSKU	Yes	No	No	No	Yes
SYSTEM_OPTIONS	No	Yes	No	No	No
TRANS_CLAIMS	No	No	No	No	Yes
TRANS_DELIVERY	No	No	No	No	Yes
TRANS_LIC_VISA	No	No	No	No	Yes
TRANS_PACKING	No	No	No	No	Yes
TRANSPORTATION	No	Yes	No	No	Yes
TSFDETAIL	Yes	No	No	No	Yes
TSFHEAD	Yes	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

INV_SQL.DELETE_INV

Program Flow

Function Level Description

Delete from the appropriate ordering tables and any tables that may have referential integrity constraints for the fetched order number. Fetch order numbers appropriately based on whether or not invoice matching is installed.

Init()

Select the following fields values:

- invc_match_ind, import_ind, repl_order_history_days, edi_rev_days, rws_ind from the system_options table
- order_history_months from the unit_options table
- vdate from period table

Init_logistics()

Initialize the format of the output file.

Process()

Call del_rev to delete order revision

Open the particular driving cursor based on the indicator inv_match_ind

For each order:

Fetch the particular driving cursor based on inv_match_ind

If ordlc_lc_ind = 'Y' then skip and fetch next record;

If import_ind = 'Y' and alc_head.status = 'PR' then

Call delete_landed_costs;

End if;

Delete from related tables associated with orders being purged by.

Calling del_appts, delete_invc_data, delete_deals and delete_repl_orders

Delete from ordhead table.

Delete_invc_data()

Call INVC_SQL.DELETE_INVC function to delete the invoice data for the specific orders being purged.

Del_rev()

Delete records from the tables ordloc_rev, ordsku_rev, ordhead_rev and alloc_rev associated with the orders which have been closed for more days than specified in edi_rev_days(in table UNIT_OPTIONS). But deleting occurs only:

- when a letter of credit is not present(ordlc_lc_ind='N').
- Import indicator equals 'N'. Or
- import indicator equals 'Y', and the landed costs are completed (alc_head.status = 'PR'). In this case, purge these landed costs before deleting the above tables.

Also, before deleting from these tables, purge all related transportation and custom entries.

Purge_transport()

Delete from the table transportation for specific orders being purged as well as child records from tables missing_doc, trans_packing, trans_delivery, trans_claims and trans_lic_visa(based on transportation_id).

Purge_customs_entry()

Delete from customs entry for specific orders being purged.

Delete_landed_costs()

Delete landed costs and obligations as well as their child records for specific orders being purged. Involved tables include: alc_head, alc_comp_loc, obligation, obligation_comp, obligation_comp_loc.

Write_logistic_details()

Write the deleted order to the output file with action type = 'D'eleted.

Delete_deals()

Delete all PO-specific deals assigned to the order being purged. PO-specific deals are identified by the existence of a value in deal_head.order_no.

Del_repl_orders ()

Delete records from the table ord_inv_mgmt and repl_results associated with the order being purged.

Final()

Close the output file.

Del_appts()

Deletes records from appt_detail, first saving distinct appt/loc combination into a local array that is dynamically sized based on the number of records to be deleted from appt_head. Then array deletes records based on the array from appt_head. Also deletes from doc_close_queue. Calls size_appt_array() to size the appt_head delete array.

Size_appt_array()

Sizes the array used to hold appt_head appt/loc info between deletes from appt_detail and appt_head.

Input Specifications**Driving cursor:**

When Retek's Invoice Matching product is not in use:

```
SELECT oh.order_no,
       lc.lc_ind
  FROM ordhead oh,
       ordlc lc
 WHERE lc.order_no(+) = oh.order_no
       AND ((0 < (NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                                     oh.close_date),0) - :pi_hist_months))
            OR (oh.status = 'W'
                AND oh.orig_ind = 0
                AND oh.contract_no is NULL
                AND (TO_DATE(:ps_vdate,'YYYYMMDD') - oh.written_date) >=
:pi_repl_order_history_days));
```

When Retek's Invoice Matching product in use:

```
SELECT distinct oh.order_no,
               lc.lc_ind
  FROM ordhead oh,
       shipment sh,
       shipsku ss,
       invc_head ih,
       ordlc lc
 WHERE oh.order_no = sh.order_no
       AND lc.order_no(+) = oh.order_no
       AND sh.shipment = ss.shipment
       AND ss.match_invc_id = ih.invc_id(+)
       AND (0 < (NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                                     oh.close_date),0) - :pi_hist_months))
       AND ((0 < (NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
                                     sh.invc_match_date),0) - :pi_hist_months))
            OR (sh.invc_match_date IS NULL))
       AND (ss.match_invc_id is null OR ih.status = 'P')
```

```

        AND sh.invc_match_status = 'C'
        AND 'C' = (SELECT decode(max(ship.invc_match_status),
min(ship.invc_match_status), 'C', 'X')
                    FROM shipment ship
                    where ship.order_no = oh.order_no
                    group by ship.order_no)
UNION
SELECT oh.order_no,
       lc.lc_ind
FROM   ordhead oh,
       ordlc lc
WHERE  lc.order_no(+) = oh.order_no
      AND oh.status = 'W'
      AND oh.orig_ind = 0
      AND oh.contract_no is NULL
      AND (TO_DATE(:ps_vdate, 'YYYYMMDD') - oh.written_date) >=
:pi_repl_order_history_days
UNION
SELECT oh.order_no,
       lc.lc_ind
FROM   ordhead oh,
       ordlc lc
WHERE  lc.order_no(+) = oh.order_no
      AND (0 < (NVL(MONTHS_BETWEEN(TO_DATE(:ps_vdate, 'YYYYMMDD'),
oh.close_date), 0) - :pi_hist_months))
      AND oh.status = 'C'
      AND NOT EXISTS (SELECT 'x'
                      FROM shipment sh
                      WHERE sh.order_no = oh.order_no);

```

Output Specifications

Output file: Format will be as in table given below. Each order is a separate transaction; multiple orders can be given in each file. The skeleton of file is:

FHEAD (file identification – only one line per file) REQUIRED

FDETL detail lines – one line for every record on ordhead. NOT REQUIRED.

FTAIL (total number lines) One line per file. REQUIRED.

Output File

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	CHAR(5)	FHEAD	Value that identifies the record type.
	Line Identifier	NUMBER(10)		identifies file line number
	File Type Definition	CHAR(4)	POHD	identifies file type
	File Create Date	CHAR(14)		YYYYMMDDHH24MISS format
File Detail	File Type Record Descriptor	CHAR(5)	"FDETL"	Value that identifies the record type.
	Line Number	NUMBER(10)		identifies file line number
	Action type	CHAR(1)	D	identifies the action type 'D'etele
	Location	CHAR(4)		ordloc.location number

	File create date	CHAR(12)	vdate	
	Order number	CHAR(8)		
	Vendor	CHAR(7)		
	Preassigned Flag	CHAR(1)		
	Dev not before	CHAR(8)		
	Dev not after	CHAR(8)		
	Shipping terms	CHAR(3)		
	Buyer code	CHAR(12)		
File Trailer	File Type Record Descriptor	CHAR(5)	FTAIL	Value that identifies the record type.
	Line number	NUMBER(10)		identifies file line number
	File Record Counter	NUMBER(6)		number of records/transactions processed in current file

Scheduling Considerations

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Purchase order information written to order history tables [ordrev]

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.

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.

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Restartability 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.order_no,
       oh.pre_mark_ind,
       ro.rowid,
       s.edi_po_chg,
       oh.status
FROM   rev_orders ro,
       ordhead oh,
       sups s
WHERE  ro.order_no = oh.order_no
      AND oh.supplier = s.supplier
      AND (oh.status = 'A' OR oh.status = 'C')
      AND MOD(ro.order_no, TO_NUMBER(:ps_restart_num_threads)) + 1 =
:ps_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 an item/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.

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.

The insert_header() function will be called from here to insert header level information, the insert_sku() function will insert order item information, the insert_item() will insert order item/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.

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_item()

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

Insert_loc()

The current information on the order item/location table is inserted into the order item/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.

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

There are no input or output files for the program.

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.

Order update [ordupd]

Design Overview

This program will be used to automatically change all retail costs on purchase orders when a retail price change is implemented for an item on the order with the status of 'Worksheet', 'Submit' and 'Approve'.

Open To Buy is updated to give a more accurate picture of the retail value of open orders if the order is 'Approved' and if the department calculate the OTB as Retail.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPS	No	Yes	No	No	No
ORDHEAD	No	Yes	No	No	No
ORDLOC	No	Yes	No	Yes	No
OTB	No	Yes	No	Yes	No
PERIOD	No	Yes	No	No	No
PRICE_HIST	No	Yes	No	No	No
V_PACKSKU_QTY	No	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

CURRENCY_SQL.CONVERT_BY_LOCATION - use this package for all conversions.

Program Flow

Function Level Description

Input Specifications

Driving Cursor:

```
EXEC SQL DECLARE c_affected_orders CURSOR FOR
EXEC SQL DECLARE c_affected_orders CURSOR FOR
  SELECT distinct 'S',
    oh.order_no,
    oh.currency_code,
    '0' pack_no,
    TO_CHAR(oh.otb_eow_date, 'YYYYMMDD') eow_date,
    oh.order_type,
    oh.status,
    NVL(ph.unit_retail,0) - NVL(ol.unit_retail, 0) ordloc_retail,
    NVL(ol.qty_ordered, 0) - NVL(ol.qty_received, 0) qty_outstanding,
    ph.item,
    ol.loc_type,
    ol.location,
    0 pack_qty,
    NVL(ph.unit_retail, 0) price_hist_unit_retail
  FROM price_hist ph,
       ordloc ol,
       ordhead oh
 WHERE oh.order_no      = ol.order_no
    AND ph.item         = ol.item
    AND ph.loc          = ol.location
    AND ph.tran_type    in (4, 8, 11)
    AND ph.action_date = TO_DATE(:ps_tomorrow, 'YYYYMMDD')
    AND oh.status       in ('W', 'S', 'A')
```

```

UNION ALL
SELECT distinct 'P',
               oh.order_no,
               oh.currency_code,
               vpq.pack_no,
               TO_CHAR(oh.otb_eow_date, 'YYYYMMDD') eow_date,
               oh.order_type,
               oh.status,
               NVL(ol.unit_retail, 0) ordloc_retail,
               vpq.qty * (NVL(ol.qty_ordered, 0) - NVL(ol.qty_received, 0))
               qty_outstanding,
               ph.item,
               ol.loc_type,
               ol.location,
               vpq.qty pack_qty,
               NVL(ph.unit_retail, 0) price_hist_unit_retail
FROM   ordloc      ol,
       ordhead     oh,
       v_packsku_qty vpq,
       price_hist   ph
WHERE  ol.item      = vpq.pack_no
      AND ph.item    = vpq.item
      AND oh.order_no = ol.order_no
      AND oh.status   in ('W', 'S', 'A')
      AND ph.loc      = ol.location
      AND ph.tran_type in (4, 8, 11)
      AND ph.action_date = TO_DATE(:ps_tomorrow, 'YYYYMMDD');

```

Output Specifications

Scheduling Considerations

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program should be run after pccext.pc and pccext.pc to ensure that all price changes have been handled by batch processing.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: This module does not contain restart/recovery.

Outstanding order export download file [otbdlord]

Design Overview

This batch program will sum outstanding orders from past periods for each subclass and export the data to a flat file for use by an outside planning system.

Approved orders of each order type are summed by subclass across all periods with an end-of-week date before today. The outstanding order amount is then calculated by subtracting all receipts for each order type by subclass from the approved order amount. This figure is written to the output file for each order type by subclass.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
OTB	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program must be run after salweek for the week just ended. This program and otbdlsal can run anytime after salweek, but saldly cannot run between otbdnld, otbdlsal and otbdlord

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Since array fetches are used in this program, only one thread will be used.

Restart Recovery

The logical unit of work for the otbdlord module is department/class/subclass with a recommended commit counter setting of 10,000. Each time the record counter equals the maximum recommended commit number, an application image array record will be written to the restart_start_array for restart/recovery if a fatal error occurs.

```

SELECT dept,
       class,
       subclass,
       ROUND(SUM(NVL(n_approved_amt,0)) -
             SUM(NVL(n_receipts_amt,0)),4),
       ROUND(SUM(NVL(b_approved_amt,0)) -
             SUM(NVL(b_receipts_amt,0)),4),
       ROUND(SUM(NVL(a_approved_amt,0)) -
             SUM(NVL(a_receipts_amt,0)),4)
FROM otb
WHERE   eow_date < TO_DATE(file_create_date,'YYYYMMDD')
AND (dept > NVL(:ora_restart_dept, -999) or
    (dept = :ora_restart_dept and

```

```

        (class > :ora_restart_class or
        (class = :ora_restart_class and
        subclass > :ora_restart_subclass)
        )
    )
)
GROUP BY dept,
        class,
        subclass
ORDER BY dept,
        class,
        subclass;

```

Program Flow

declare the start string array to include department, class and subclass
 declare output file variables

```

main()
{
    accept the output file name as an argument from the command line
    init()
    {
        initialize restart/recovery
        assign restart variables
        copy start string variables from the restart_start_array
            fetch today's date and, if not restarting, select
            otb_file_rec_sequence.NEXTVAL
        open the temp output file
        if not restarting, write a header record to the temp file
    }

    declare the driving cursor
    table_process()
    {
        set up out_buffer to receive fetched data
        open the driving cursor
        LOOP
            fetch
            if no rows are found, set the end flag
            break if there are no rows left to process
            process_records()
                LOOP through the records to be processed
                    increment the file line sequence number
                    print each record fetched to the temp file
                end LOOP
            break if the end flag is set
            set the application image array and call restart_commit() to save it
            call restart_file_write() to write successfully processed records from the temp file to the
            final output file
        end LOOP
    }
}

```

```

        close the driving cursor
        write a trailer record to the temp file
    }

    final()
    {
        close restart/recovery logic
        if no errors were encountered in the last round of processing, write any remaining records in
            the temp file to the final output file
        close the export file
        remove the temp file
    }

```

Shared Modules

N/A

Function Level Description

Restart/recovery for table-based processing should be implemented for this program.

main():

- Accept three runtime parameters at the command line: program name, user name/password, and output file name.

init():

- Initialize restart/recovery.
- Open the temp file for output.
- Fetch today's date into file_write_date.
- If not restarting, fetch otb_file_rec_sequence.NEXTVAL into transaction_set_control_number and write the file header record to the temp file.
- Set up an array to receive fetches in table_process() - memory should be dynamically allocated so that the cursor fetches arrays of size g_l_restart_max_counter.

table_process():

- Set up out_buffer to receive fetched data:


```

strcpy(out_buffer, "%5s"); /* file type record descriptor */
strcat(out_buffer, "%-10d"); /* file line sequence number */
strcat(out_buffer, "%-14d"); /* transaction set control number */
strcat(out_buffer, "%04d"); /* dept */
strcat(out_buffer, "%04d"); /* class */
....
strcat(out_buffer, "%020.0f"); /* a_outs_amt */
....

```

Each strcpy/strcat combination should set up space in out_buffer for a single fetched variable. It should copy the appropriate data type (d for decimal integers or s for strings), and all numbers should be left-padded with zeroes except counters, which are left-justified and right-padded with spaces. The length of the space should correspond with the variable's length on the OTB table (e.g. cancel_amt is an oracle variable of (20,4)--it needs a space 20 wide and should be left-padded with zeroes (%020.0f)). The name of the variable that will be copied into that space is included as a comment to ensure maintainability of this code.

- Open the driving cursor C_OUTS_ORD.

- ```
final():
```
- Close restart/recovery logic.
  - If no errors were encountered in the final round of processing, write any last records from the temp file to the final output file.
  - Close the output file and delete the temp file.

- Loop through a subset of records and write each of them to the temp file using `out_buffer` as a format mask.
- The file `line` `sequence` `number` should be incremented each time a record is written.

Output File:  
The output file name should be accepted as a runtime parameter at the command line.

A sample detail record would look like the following:

A sequence will need to be added to the database to increment the Transaction Set Control Number. This sequence should be large enough that there is very little danger of running out of numbers. It will give a unique identifying number to every file exported from Retek to an outside planning system.

| Record Name | Field Name                        | Field Type | Default Value | Description                 |
|-------------|-----------------------------------|------------|---------------|-----------------------------|
| File Header | File Type<br>Record<br>Descriptor | Char(5)    | FHEAD         | Identifies file record type |



|              |                                |               |                                   |                                                                     |
|--------------|--------------------------------|---------------|-----------------------------------|---------------------------------------------------------------------|
|              | File Line Sequence Number      | Number(10 )   | line number in file               | Keeps track of the record's position in the file by line number     |
|              | File Type Definition           | Char(4)       | OTBE                              | Identifies file as 'OTB Export'                                     |
|              | File Create Date               | Date          | vdate                             | Date file was written by batch program                              |
| File Detail  | File Type Record Descriptor    | Char(5)       | FDETL                             | Identifies file record type                                         |
|              | File Line Sequence Number      | Number(10 )   | line number in file               | Keeps track of the record's position in the file by line number     |
|              | Transaction Set Control Number | Number(14 )   | sequence number                   | Used to force unique file check                                     |
|              | Department                     | Number(4)     | dept identifier                   | The id number of a department                                       |
|              | Class                          | Number(4)     | class identifier                  | The id number of a class within the department given                |
|              | Subclass                       | Number(4)     | subclass identifier               | The id number of a subclass within the class given                  |
|              | N Outstanding Amt              | Number(20 ,4) | non-basic outstanding order amt   | The amt. of outstanding non-basic orders for past periods           |
|              | B Outstanding Amt              | Number(20 ,4) | buyer-basic outstanding order amt | The amt. of outstanding buyer-basic orders for past periods         |
|              | A Outstanding Amt              | Number(20 ,4) | auto-basic outstanding order amt  | The amt. of outstanding auto-basic orders for past periods          |
| File Trailer | File Type Record Descriptor    | Char(5)       | FTAIL                             | Identifies file record type                                         |
|              | File Line Sequence Number      | Number(10 )   | line number in file               | Keeps track of the record's position in the file by line number     |
|              | Control Number File Line Count | Number(10 )   | total detail lines                | sum of all transaction lines, not including file header and trailer |

#### Error File

Standard Retek batch error handling 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**

Level Customization:

CLASS LEVEL (extrapolate for department level):

- Remove the subclass field from the export file specification.
- Remove subclass from the logical unit of work, the start string array, and the driving cursor.
- Remove space created for subclass from out\_buffer.
- Remove logic involving subclass from the fprintf statement printing to the output file.

MONTH LEVEL (454 & JULIAN):

- Change all variable names like 'eow\_date' to 'eom\_date'.

## Stock ledger data download [otbdlisal]

### Design Overview

This module will sum stock ledger data from the DAILY\_DATA table and opening stock information from the WEEK\_DATA table across the current week, grouping by department, class, subclass, location and date, and export the data to a flat file for use by an outside planning system.

| TABLE      | INDEX | SELECT | INSERT | UPDATE | DELETE |
|------------|-------|--------|--------|--------|--------|
| DAILY_DATA | Yes   | Yes    | No     | No     | No     |
| WEEK_DATA  | Yes   | Yes    | No     | No     | No     |
| PERIOD     | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: PHASE 4 (weekly/monthly)

Scheduling Diagram: This program must be run after salweek for the week just ended. This program and otbdnld can run anytime after salweek, but saldly cannot run between otbdnld, otbdlisal and otbdlord.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Since array fetches are used in this program, only one thread will be used.

### Restart Recovery

The logical unit of work for the otbdlisal module is department, class, subclass, store, warehouse, half, month and week with a recommended commit counter setting of 10,000. Each time the record counter equals the maximum recommended commit number, an application image array record will be written to the restart\_start\_array for restart/recovery if a fatal error occurs.

```
SELECT w.dept,
 w.class,
 w.subclass,
 w.store,
 w.wh,
 w.half_no,
 w.month_no,
 w.week_no,
 NVL(w.opn_stk_retail,0),
 NVL(w.opn_stk_cost,0),
 SUM(NVL(d.stock_adj_retail,0)),
 SUM(NVL(d.stock_adj_cost,0)),
 SUM(NVL(d.purch_retail,0)),
 SUM(NVL(d.purch_cost,0)),
```

```

SUM(NVL(d.rtv_retail,0)),
SUM(NVL(d.rtv_cost,0)),
SUM(NVL(d.freight_cost,0)),
SUM(NVL(d.net_sales_retail,0)),
SUM(NVL(d.net_sales_cost,0)),
SUM(NVL(d.returns_retail,0)),
SUM(NVL(d.returns_cost,0)),
SUM(NVL(d.prom_markdown_retail,0)),
SUM(NVL(d.markdown_can_retail,0)),
SUM(NVL(d.empl_disc_retail,0)),
SUM(NVL(d.workroom_amt,0)),
SUM(NVL(d.cash_disc_amt,0)),
SUM(NVL(d.sales_units,0)),
SUM(NVL(d.markup_retail,0)),
SUM(NVL(d.markup_can_retail,0)),
SUM(NVL(d.clear_markdown_retail,0)),
SUM(NVL(d.perm_markdown_retail,0))
FROM week_data w, daily_data d
WHERE d.dept(+) = w.dept
AND d.class (+) = w.class
AND d.subclass (+) = w.subclass
AND d.store (+) = w.store
AND d.wh (+) = w.wh
AND d.currency_ind(+) = w.currency_ind
AND d.currency_ind(+) = 'L'
AND w.eow_date = L_eow_date
AND d.data_date (+) >= L_eow_date - 6
AND (w.dept > NVL(:ora_restart_dept, -999) or
(w.dept = :ora_restart_dept and
(w.class > :ora_restart_class or
(w.class = :ora_restart_class and
(w.subclass > :ora_restart_subclass or
(w.subclass = :ora_restart_subclass and
(w.store > :ora_restart_store or
(w.store = :ora_restart_store and
w.wh > :ora_restart_wh))))))))))
GROUP BY w.dept,
w.class,
w.subclass,
w.store,
w.wh,
w.half_no,
w.month_no,
w.week_no,
w.opn_stk_retail,
w.opn_stk_cost
ORDER BY w.dept,
w.class,
w.subclass,
w.store,
w.wh;

```

**Program Flow**

declare the start string array to include department, class, subclass, store, warehouse  
 declare output file variables

```

main()
{
 accept the output file name as an argument from the command line

 init()
 {
 initialize restart/recovery
 assign restart variables
 copy start string variables from the restart_start_array
 fetch today's date and, if not restarting, select otb_file_rec_sequence.NEXTVAL
 open the temp output file
 if not restarting, write a header record to the temp file
 }

 declare the driving cursor
 table_process()
 {
 call CAL_TO_454_LDOW to get the last day of the current week
 set up out_buffer to receive fetched data
 open the driving cursor
 LOOP
 fetch
 if no rows are found, set the end flag
 break if there are no rows left to process
 process_records()
 LOOP through the records to be processed
 increment file_line_sequence_number
 print each record fetched to the temp file
 end LOOP
 break if the end flag is set
 set the application image array and call restart_commit() to save it
 call restart_file_write() to write successfully processed records from the temp file to
 the final output file
 end LOOP
 close the driving cursor
 write a trailer record to the temp file
 }

 final()
 {
 close restart/recovery logic
 if no errors were encountered in the last round of processing, write any remaining records in
 the temp file to the final output file
 close the export file
 remove the temp file
 }
 }

```

**Shared Modules**

CAL\_TO\_454\_LDOW to get the end-of-week date for the current week.

**Function Level Description**

Restart/recovery for table-based processing should be implemented for this program. Use the restart\_skeleton program as a framework.

main():

- Accept three runtime parameters at the command line: program name, user name/password, and output file name.

init():

- Initialize restart/recovery.
- Open the temp file for output.
- Fetch today's date into file\_write\_date.
- If not restarting, fetch otb\_file\_rec\_sequence.NEXTVAL into transaction\_set\_control\_number and write a file header record to the temp file.
- Set up an array to receive fetches in table\_process() - memory should be dynamically allocated so that the cursor fetches arrays of size g\_l\_restart\_max\_counter.

table\_process():

- Pass file\_write\_date into CAL\_TO\_454\_LDOW to get the last day of the current week.
- Set up out\_buffer to receive fetched data:  

```
strcpy(out_buffer, "%5s"); /* file type record descriptor */
strcat(out_buffer, "%-10d"); /* file line sequence number */
strcat(out_buffer, "%-14d"); /* transaction set control number */
strcat(out_buffer, "%04d"); /* dept */
strcat(out_buffer, "%04d"); /* class */
....
strcat(out_buffer, "%020.0f"); /* opn_stk_retail */
....
```

Each strcpy/strcat combination should set up space in out\_buffer for a single fetched variable. It should copy the appropriate datatype (d for decimal integers or s for strings), and all numbers should be left-padded with zeroes except counters, which are left-justified and right-padded with spaces. The length of the space should correspond with the variable's length on the OTB table (e.g. cancel\_amt is an oracle variable of (20,4)--it needs a space 20 wide and should be left-padded with zeroes (%020d)). The name of the variable that will be copied into that space is included as a comment to ensure maintainability of this code.

- Open the driving cursor C\_STK.
- Use a while loop to fetch rows from the table into the arrays.
  - If no rows are fetched from the table, set a flag to indicate that processing should end after this array.
  - Set file\_type\_record\_descriptor to 'FDETL'.
  - Break if there are no more rows to process.
  - Call process\_records() to process the array.
  - Break if the end flag has been set.
  - Copy the last row's start string variables into ora\_restart\_cur\_string, delimited by semicolons. This will serve as the application image array. Call the restart\_commit() function to save it.

- Call the `restart_file_write()` function to write successfully processed records to the temp file.
- End while loop.
- Close `C_STK`.
- Write a trailer record to the temp file with `control_number_file_line_count = file_line_sequence_number - 2` and `file_type_record_descriptor = 'FTAIL'`.

final():

- Close restart/recovery logic.
- If no errors were encountered in the final round of processing, write any remaining records from the temp file to the final output file
- Close the output file and delete the temp file.

```
process_records():
```

- Loop through a subset of records and write each of them to the temp file using `out_buffer` as a format mask.
- The file `line` `sequence` `number` should be incremented each time a record is written.

## I/O Specification

Output File:

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

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

A sample detail record would look like the following:

[illegible]

A sequence will need to be added to the database to increment the Transaction Set Control Number. This sequence should be large enough that there is very little danger of running out of numbers. It will give a unique identifying number to every file exported from Retek to an outside planning system.

| 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   | Number(10) | line number in file | Keeps track of the record's position in the file by line number |
|             | File Type Definition        | Char(4)    | STKE                | Identifies file as 'Stk Export'                                 |
|             | File Create Date            | Date       | vdate               | Date file was written by batch program                          |
| File Detail | File Type Record            | Char(5)    | FDETL               | Identifies file record type                                     |

|  | Descriptor                     |              |                      |                                                                                            |
|--|--------------------------------|--------------|----------------------|--------------------------------------------------------------------------------------------|
|  | File Line Sequence Number      | Number(10)   | line number in file  | Keeps track of the record's position in the file by line number                            |
|  | Transaction Set Control Number | Number(14)   | sequence number      | Used to force unique file check                                                            |
|  | Department                     | Number(4)    | dept identifier      | The id number of a department                                                              |
|  | Class                          | Number(4)    | class identifier     | The id number of a class within the department given                                       |
|  | Subclass                       | Number(4)    | subclass identifier  | The id number of a subclass within the class given                                         |
|  | Store                          | Number(10)   | store identifier     | The store from which stock ledger data was collected (-1 if this location is a WH)         |
|  | Wh                             | Number(10)   | warehouse identifier | The wh from which stock ledger data was collected (-1 if this location is a store)         |
|  | Half No.                       | Number(5)    | half no.             | The half number for this stock ledger data                                                 |
|  | Month No.                      | Number(2)    | month no.            | The month number in the half for this stock ledger data                                    |
|  | Week No.                       | Number(2)    | week no.             | The week number in the month for this stock ledger data                                    |
|  | Open Stock Retail              | Number(20,4) | opn stk retail       | The retail opening stock from the week_data table for this stock ledger period             |
|  | Open Stock Cost                | Number(20,4) | opn stk cost         | The cost opening stock from the week_data table for this stock ledger period               |
|  | Stock Adjustments Retail       | Number(20,4) | stock adj retail     | The retail stock adjustments summed from the daily_data table for this stock ledger period |
|  | Stock Adjustments Cost         | Number(20,4) | stock adj cost       | The cost stock adjustments summed from the daily_data table for this stock ledger period   |
|  | Purchases Retail               | Number(20,4) | purch retail         | The retail purchases summed from the                                                       |



|  |                               |              |                      |                                                                                                  |
|--|-------------------------------|--------------|----------------------|--------------------------------------------------------------------------------------------------|
|  |                               |              |                      | daily_data table for this stock ledger period                                                    |
|  | Purchases Cost                | Number(20,4) | purch cost           | The cost purchases summed from the daily_data table for this stock ledger period                 |
|  | RTV Retail                    | Number(20,4) | rtv retail           | The retail return to vendor amount summed from the daily_data table for this stock ledger period |
|  | RTV Cost                      | Number(20,4) | rtv cost             | The cost return to vendor amount summed from the daily_data table for this stock ledger period   |
|  | Freight Cost                  | Number(20,4) | freight cost         | The freight cost summed from the daily_data table for this stock ledger period                   |
|  | Net Sales Retail              | Number(20,4) | net sales retail     | The retail net sales summed from the daily_data table for this stock ledger period               |
|  | Net Sales Cost                | Number(20,4) | net sales cost       | The cost net sales summed from the daily_data table for this stock ledger period                 |
|  | Returns Retail                | Number(20,4) | returns retail       | The retail returns amount summed from the daily_data table for this stock ledger period          |
|  | Returns Cost                  | Number(20,4) | returns cost         | The cost returns amount summed from the daily_data table for this stock ledger period            |
|  | Promotional Markdowns Retail  | Number(20,4) | prom markdown retail | The retail promotional markdowns summed from the daily_data table for this stock ledger period   |
|  | Markdown Cancellations Retail | Number(20,4) | markdown can retail  | The retail markdown cancellations summed from the daily_data table for this stock ledger period  |
|  | Employee Discount Retail      | Number(20,4) | empl disc retail     | The retail employee discounts amount summed from the daily_data table for this                   |

|              |                                |              |                       |                                                                                               |
|--------------|--------------------------------|--------------|-----------------------|-----------------------------------------------------------------------------------------------|
|              |                                |              |                       | stock ledger period                                                                           |
|              | Workroom Amount                | Number(20,4) | workroom amt          | The workroom amount summed from the daily_data table for this stock ledger period             |
|              | Cash Discount Amount           | Number(20,4) | cash disc amt         | The cash discounts amount summed from the daily_data table for this stock ledger period       |
|              | Sales Units                    | Number(12,4) | sales units           | The sales units summed from the daily_data table for this stock ledger period                 |
|              | Markups Retail                 | Number(20,4) | markup retail         | The retail markups summed from the daily_data table for this stock ledger period              |
|              | Markup Cancellations Retail    | Number(20,4) | markup can retail     | The retail markup cancellations summed from the daily_data table for this stock ledger period |
|              | Clearance Markdowns Retail     | Number(20,4) | clear markdown retail | The retail clearance markdowns summed from the daily_data table for this stock ledger period  |
|              | Permanent Markdowns Retail     | Number(20,4) | perm markdown retail  | The retail permanent markdowns summed from the daily_data table for this stock ledger period  |
| File Trailer | File Type Record Descriptor    | Char(5)      | FTAIL                 | Identifies file record type                                                                   |
|              | File Line Sequence Number      | Number(10)   | line number in file   | Keeps track of the record's position in the file by line number                               |
|              | Control Number File Line Count | Number(10)   | total detail lines    | Sum of all transaction lines, not including file header and trailer                           |

#### Error File

Standard Retek batch error handling 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**

Level Customization:

CLASS LEVEL (extrapolate for department level):

- Remove the subclass field from the export file specification.
- Remove subclass from the logical unit of work, the start string array, and the driving cursor.
- Remove space created for subclass from out\_buffer.
- Remove logic involving subclass from the fprintf statement printing to the output file.

MONTH LEVEL (454 & JULIAN):

- Change all variable names like “eow\_date” to “eom\_date”.
- Call CAL\_TO\_454\_LDOM (for 454 calendar) or CAL\_TO\_CAL\_LDOM (for Julian calendar) instead of CAL\_TO\_454\_LDOW to determine the last day of the current month (L\_eom\_date).
- Call CAL\_TO\_454\_FDOM (for 454 calendar) or CAL\_TO\_CAL\_FDOM (for Julian calendar) to determine the first day of the current month (L\_bom\_date).
- Change the cursor line “and d.data\_date >= L\_eow\_date - 6” to “and w.eow\_date >= L\_bom\_date”.
- Remove the week\_no field from the export file specification.
- Remove space created for week\_no from out\_buffer.
- Remove logic involving week\_no from the fprintf statement printing to the output file.
- Change the cursor to select opening stock figures from month\_data and summed rows from week\_data .

## Open to buy download [otbdnld]

### Design Overview

This batch program will extract current and future Open to Buy data from the OTB table and export it to a flat file for use by an outside planning system.

All Open to Buy data except budget figures is exported directly to the file from the table with no intermediate manipulation.

Tables Affected:

| TABLE  | INDEX | SELECT | INSERT | UPDATE | DELETE |
|--------|-------|--------|--------|--------|--------|
| OTB    | Yes   | Yes    | No     | No     | No     |
| PERIOD | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: PHASE 4 (weekly/monthly)

Scheduling Diagram: This program must be run after salweek for the week just ended. This program and otbdlsal can run anytime after salweek, but saldly cannot run between otbdnld and otbdlsal

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Since array fetches are used in this program, only one thread will be used.

### Restart Recovery

The logical unit of work for the otbdnld module is department, class, subclass and end-of-week date with a recommended commit counter setting of 10,000. Each time the record counter equals the maximum recommended commit number, an application image array record will be written to the restart\_start\_array for restart/recovery if a fatal error occurs.

```
SELECT dept,
 class,
 subclass,
 TO_CHAR(eow_date,'YYYYMMDD'),
 week_no,
 month_no,
 half_no,
 NVL(cancel_amt,0),
 NVL(n_approved_amt,0),
 NVL(n_receipts_amt,0),
 NVL(b_approved_amt,0),
 NVL(b_receipts_amt,0),
```

```

 NVL(a_approved_amt,0),
 NVL(a_receipts_amt,0)
 FROM otb
 WHERE eow_date >= TO_DATE(file_create_date,'YYYYMMDD')
 AND (dept > NVL(:ora_restart_dept, -999) or
 (dept = :ora_restart_dept and
 (class > :ora_restart_class or
 (class = :ora_restart_class and
 (subclass > :ora_restart_subclass or
 (subclass = :ora_restart_subclass and
 eow_date > :ora_restart_eow_date)
)
)
)
)
)
 ORDER BY dept,
 class,
 subclass,
 eow_date;

```

### Program Flow

declare the start string array to include department, class, subclass and end-of-week date  
 declare output file variables

```

main()
{
 accept the output file name as an argument from the command line

 init()
 {
 initialize restart/recovery
 copy start string variables from the restart_start_array
 fetch today's date and, if not restarting, select
 otb_file_rec_sequence.NEXTVAL
 open the temp output file
 if not restarting, write a header record to the temp file
 }

 declare the driving cursor
 table_process()
 {
 set up out_buffer to receive fetched data
 open the driving cursor
 LOOP
 fetch
 if no rows are found, set the end flag
 break if there are no rows left to process
 process_records()

```

```

 LOOP through the records to be processed
 increment the file line sequence number
 print each record fetched to the temp file
 end LOOP
 break if the end flag has been set
 set the application image array and call restart_commit() to save it
 call restart_file_write() to write successfully processed records from the temp file
 to the final output file
 end LOOP
 close the driving cursor
 write a trailer record to the temp file
}

final()
{
 close restart/recovery logic
 if no errors were encountered in the last round of processing, write any
 remaining records in the temp file to the final output file
 close the export file
 remove the temp file
}

```

### Shared Modules

N/A

### Function Level Description

Restart/recovery for table-based processing should be implemented for this program.

- main():

- Accept three runtime parameters at the command line: program name, user name/password, and output file name.

init():

- Initialize restart/recovery.
- Open the temp file for output.
- Fetch today's date into file\_write\_date.
- If not restarting, fetch otb\_file\_rec\_sequence.NEXTVAL into transaction\_set\_control\_number and write a file header record to the temp file.
- Set up an array to receive fetches in the table\_process() function - memory should be dynamically allocated so that the cursor fetches arrays of size g\_l\_restart\_max\_counter.

table\_process():

- Set up out\_buffer to receive fetched data:
 

```

strcpy(out_buffer, "%5s"); /* file type record descriptor */
strcat(out_buffer, "%-10d"); /* file line sequence number */
strcat(out_buffer, "%-14d"); /* transaction set control number */
strcat(out_buffer, "%04d"); /* dept */
strcat(out_buffer, "%04d"); /* class */
....
strcat(out_buffer, "%020.0f"); /* cancel_amt */

```

Each strecpy/strcat combination should set up space in out\_buffer for a single fetched variable. It should copy the appropriate data type (d for decimal integers or s for strings), and all numbers should be left-padded with zeroes except counters, which are left-justified and right-padded with spaces. The length of the space should correspond with the variable's length on the OTB table (e.g. cancel\_amt is an oracle variable of (20,4)--it needs a space 20 wide and should be left-padded with zeroes (%020.0f)). The name of the variable that will be copied into that space is included as a comment to ensure maintainability of this code.

- Open the driving cursor C\_OTB.
- Use a while(1) loop to fetch rows from the table into the arrays.
  - If all rows on the table have been read, set a flag to indicate that processing should end after this array.
  - Set file\_type\_record\_descriptor to 'FDETL'.
  - Break if there are no more rows to process.
  - Call process\_records() to process the array.
  - Break if the end flag has been set.
  - Copy the last row's start string variables into ora\_restart\_cur\_string, delimited by semicolons. This will serve as the application image array. Call restart\_commit() to save it.
  - Call restart\_file\_write() to write successfully processed records to the temp file.
- End while(1) loop.
- Close C\_OTB.
- Write a trailer record to the temp file with control\_number\_file\_line\_count = file\_line\_sequence\_number - 2 and file\_type\_record\_descriptor = 'FTAIL'.

final():

- Close restart/recovery logic.
- If no errors were encountered in the final round of processing, write any remaining records from the temp file to the final output file.
- Close the output file and delete the temp file.

```
process records():
```

- Loop through a subset of records and write each of them to the temp file using out\_buffer as a format mask.
- The file\_line\_sequence\_number should be incremented each time a record is written.

## I/O Specification

Output File:

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

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

A sample detail record would look like the following:

[illegible]

A sequence will need to be added to the database to increment the Transaction Set Control Number. This sequence should be large enough that there is very little danger of running out of numbers. It will give a unique identifying number to every file exported from Retek to an outside planning system.

| 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      | Number(10)   | line number in file    | Keeps track of the record's position in the file by line number                |
|             | File Type Definition           | Char(4)      | OTBE                   | Identifies file as 'OTB Export'                                                |
|             | File Create Date               | Date         | today's date           | Date file was written by batch program                                         |
| File Detail | File Type Record Descriptor    | Char(5)      | FDETL                  | Identifies file record type                                                    |
|             | File Line Sequence Number      | Number(10)   | line number in file    | Keeps track of the record's position in the file by line number                |
|             | Transaction Set Control Number | Number(14)   | sequence number        | Used to force unique file check                                                |
|             | Department                     | Number(4)    | dept identifier        | The id number of a department                                                  |
|             | Class                          | Number(4)    | class identifier       | The id number of a class within the department given                           |
|             | Subclass                       | Number(4)    | subclass identifier    | The id number of a subclass within the class given                             |
|             | Eow Date                       | Date         | end of week date       | The end of week date for the budgeted period                                   |
|             | Week No.                       | Number(2)    | week no.               | The week number in the month for the budgeted period                           |
|             | Month No.                      | Number(2)    | month no.              | The month number in the half for the budgeted period                           |
|             | Half No.                       | Number(5)    | half no.               | The half number for the budgeted period                                        |
|             | Cancel Amt.                    | Number(20,4) | cancel amt.            | The total amt. cancelled from orders of all order type for the budgeted period |
|             | N Approved Amt                 | Number(20,4) | non-basic approved amt | The amt. of approved non-basic orders for the                                  |



|              |                                |              |                          |                                                                                   |
|--------------|--------------------------------|--------------|--------------------------|-----------------------------------------------------------------------------------|
|              |                                |              |                          | budgeted period                                                                   |
|              | N Receipts Amt                 | Number(20,4) | non-basic receipts amt   | The amt. of non-basic orders due in the budgeted period that have been received   |
|              | B Approved Amt                 | Number(20,4) | buyer-basic approved amt | The amt. of approved buyer-basic orders for the budgeted period                   |
|              | B Receipts Amt                 | Number(20,4) | buyer-basic receipts amt | The amt. of buyer-basic orders due in the budgeted period that have been received |
|              | A Approved Amt                 | Number(20,4) | auto-basic approved amt  | The amt. of approved auto-basic orders for the budgeted period                    |
|              | A Receipts Amt                 | Number(20,4) | auto-basic receipts amt  | The amt. of auto-basic orders due in the budgeted period that have been received  |
| File Trailer | File Type Record Descriptor    | Char(5)      | FTAIL                    | Identifies file record type                                                       |
|              | File Line Sequence Number      | Number(10)   | line number in file      | Keeps track of the record's position in the file by line number                   |
|              | Control Number File Line Count | Number(10)   | total detail lines       | sum of all transaction lines, not including file header and trailer               |

#### Error File:

Standard Retek batch error handling 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

##### Level Customization:

##### CLASS LEVEL (extrapolate for department level):

- Remove the subclass field from the export file specification.
- Remove subclass from the logical unit of work, the start string array, and the driving cursor.
- Remove space created for subclass from out\_buffer.
- Remove logic involving subclass from the fprintf statement printing to the output file.

##### MONTH LEVEL (454 & JULIAN):

- Change all variable names like 'eow\_date' to 'eom\_date'.
- Remove the week\_no field from export file specification.
- Remove space created for week\_no from out\_buffer.
- Remove logic involving week\_no from the fprintf statement printing to the output file.

## Open to buy purge [otbprg]

### Design Overview

This batch program runs at the end of the half to delete rows from the OTB table that are at least one half old. The current and previous half's purchase budget data is retained. Open to Buy history can be retained longer with a modification to the function that calculates the purge date.

Table Affected:

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-------|-------|--------|--------|--------|--------|
| OTB   | No    | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

### Restart Recovery

There is no restart/recovery in this module.

The logical unit of work is 10,000 records of data less than a particular date.

There is no driving cursor in this module.

### Program Flow

Aside from the main function and its standard sub-functions, the body of this program is all in one function called process() which does the following:

- Retrieves the purge date.
- Deletes records with an end-of-week date prior to the purge date.
- Commits every 10,000 records.

### Shared Modules

OTB\_SQL.GET\_PURGE\_DATE is called to determine which records should be deleted.

Deleted records should have a week ending date that is earlier than the purge date.

SQL\_LIB.BATCH\_MSG is needed to format error messaging from the package so that the batch module can use the information.

### Function Level Description

process():

- Call the function OTB\_SQL.GET\_PURGE\_DATE to retrieve the date that will be used to determine which records should be purged.
- Set up a while loop to delete from the OTB table in groups of 10,000 records where the end-of-week date is less than or equal to the OTB purge date. After each set of 10,000 records is deleted, perform a commit. If less than 10,000 records were processed by the previous delete statement, exit the loop.
- Error handling should be written according to the new standards for batch programs calling a batch-enabled package.

**I/O Specification**

N/A

**Technical Issues**

N/A

## Accept forward limit percentages upload [otbupfwd]

### Design Overview

The purpose of this batch module is to accept forward limit percentages by period from an outside planning system. This data is then inserted into or used to update the forward limit percentage held on the OTB\_FWD\_LIMITS table.

There is no processing done to these records as the data is transferred directly from the input file to the table. If there is not a row on the table to update, a new row is inserted with the department, class, subclass, period ahead and forward limit percentage as taken from the input file.

Tables affected:

| TABLE         | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---------------|-------|--------|--------|--------|--------|
| OTB_FWD_LIMIT | No    | No     | Yes    | Yes    | No     |

### Scheduling Constraints

Processing Cycle: PHASE AD-HOC

Scheduling Diagram: This program is independent of all other batch processing and can be run at any time in the batch cycle.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

The logical unit of work for the otbupfwd module is the discrete forward limit transaction. Each record is uniquely identified by department, class, subclass and period ahead. Processing of each row is independent and thus if an erroneous record is found during processing, only that record needs to be corrected and reprocessed.

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE\_ERROR library function.

The recommended commit counter setting is 10,000. A savepoint will be issued after each record is successfully processed. If a non-fatal error occurs, a rollback to the last savepoint will be issued so that the rejected record is not posted to the database. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

**Program Flow**

```

main()
{
 accept the names of the input and reject files as arguments from the
 command line
 declare structures for the file header, detail and trailer records
 declare an application image array that includes g_l_rec_cnt and g_l_rej_cnt

 init()
 {
 open the input and reject files
 initialize restart/recovery
 call get_record()
 if it fails, return FATAL ERROR
 }

 file_process()
 {
 LOOP
 read the number of records indicated by g_l_restart_max_ctr into memory
 if the end of file character is reached, decrement the record counter by 1 and set the
 end of file flag to TRUE
 LOOP
 read a record
 copy items from structs into varchar arrays and nullpad them
 validate_record()
 check that department, class, subclass and period ahead are not null by
 calling field_has_value()
 check that department, class and subclass are numbers by calling
 valid_all_numeric() and call SUBCLASS_VALIDATE_SQL.EXIST to
 see that they exist on the subclass table
 check that 0.00 <= fwd_limit_pct <= 100.00
 check that 0 < period_ahead < 100
 if the record fails validation, return NON-FATAL ERROR (the library
 validation functions will return 1 on NON-FATAL ERROR)
 if NON-FATAL ERROR is returned during validation
 call write_to_rej_file() to write the invalid record to the reject file
 increment g_l_rej_cnt
 move on to the next record
 process_record()
 update the appropriate row on OTB_FWD_LIMITS with fwd_limit_pct from
 the input file record
 if no row is found to update, insert a new one with the input file record
 values
 if the update/insert fails, return FATAL ERROR
 end LOOP
 increment the record counter by the number of records processed in the loop
 set the application image array
 call restart_file_commit()
 call restart_file_write()
 }
}

```

```

 exit if the end of file flag has been set to TRUE
 end LOOP
 call process_dtl_ftail () to process the file trailer record
}

final()
{
 close restart/recovery logic
 close the reject file
}
}

```

### Shared Modules

SUBCLASS\_VALIDATE\_SQL.EXIST: This function will validate that a department/class/subclass combination exists on the subclass table.

### Function Level Description

main():

- Accept four runtime parameters at the command line: program name, user name/password, input file name, and reject file name. Copy the input reject file names into strings for later use.

init():

- Open the input and reject files.
- Initialize restart/recovery.
- Call the get\_record() function to read the file header record.

file\_process():

- Set up a buffer in memory to hold the number of input file rows indicated by g\_l\_restart\_max\_counter.
- Use a while loop to read g\_l\_restart\_max\_counter number of rows into the buffer.
  - If the end of file character is reached, decrement the record counter by one (to avoid processing file trailer record) and set the end-of-file flag to TRUE.
  - Set up a FOR loop to read each record from the buffer into the detail record structure and process it.
    - Copy each record item into a local variable for processing.
    - Call the validate\_record() function to validate record items.
    - If validate\_record() returns a value of 1 (an error was encountered) then handle NON-FATAL ERROR.
    - Call the process\_record() function.
    - If record processing fails, handle FATAL ERROR.
  - End FOR loop.
  - Increment the record counter by the number of records processed in the loop.
  - Set the application image array for restart/recovery and call restart\_file\_commit() to save it.
  - Call the restart\_file\_write() function to write rejected records to the rejected record file.
  - If the end-of-file has been reached, exit the while loop.
- End while loop.
- Call the process\_dtl\_ftail() function to process the file trailer record.

validate\_record():

- Check that the period ahead is a number between 1 and 99 (inclusive)
- Ensure that the department, class, subclass and period ahead fields all contain values by calling field\_has\_value() for each. If one of the fields is NULL, field\_has\_value() will return 1 and the record should be written to the rejected record file.
- Ensure that the department, class and subclass fields are numeric by calling valid\_all\_numeric() for each and, if no error is returned, call SUBCLASS\_VALIDATE\_SQL.EXIST to validate that the record's department/class/subclass combination exists on the subclass table.
- Check that forward limit percent is a number between 0.00 and 100.00 (inclusive).

process\_record():

- Update the OTB\_FWD\_LIMITS table (by department, class, subclass and period ahead) with the forward limit percent extracted from the input file.
- If no row is found to update, insert a new one with the input record's values.
- If the insert fails, raise FATAL ERROR.

final():

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Rollback to the previous physical commit point.
- Exit the program.

on NON-FATAL ERROR:

- Rollback to the previous savepoint.
- Call the function write\_to\_rej\_file() to write rejected records to the rejected record file.
- Increment g\_l\_rej\_cnt.

## I/O Specification

Input File:

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

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

A sample detail record would look like the following:

FD ETL12345678901234567890123412341234123499123456789012

| Record Name | Field Name                        | Field Type | Default Value | Description                 |
|-------------|-----------------------------------|------------|---------------|-----------------------------|
| File Header | File Type<br>Record<br>Descriptor | Char(5)    | FHEAD         | Identifies file record type |
|             | File Line                         | Number(10) | Line number   | Keeps track of the          |

|              |                                |                 |                              |                                                                     |
|--------------|--------------------------------|-----------------|------------------------------|---------------------------------------------------------------------|
|              | Sequence Number                | )               | in file                      | record's position in the file by line number                        |
|              | File Type Definition           | Char(4)         | FWDI                         | Identifies file as 'FWD Import'                                     |
|              | File Create Date               | Date            | create date                  | Date file was written by external system                            |
| File Detail  | File Type Record Descriptor    | Char(5)         | FDETL                        | Identifies file record type                                         |
|              | File Line Sequence Number      | Number(10)<br>) | Line number in file          | Keeps track of the record's position in the file by line number     |
|              | Transaction Set Control Number | Number(14)<br>) | specified by external system | Used to force unique transaction check                              |
|              | Department                     | Number(4)       | dept identifier              | The id number of a department                                       |
|              | Class                          | Number(4)       | class identifier             | The id number of a class within the department given                |
|              | Subclass                       | Number(4)       | subclass identifier          | The id number of a subclass within the class given                  |
|              | Period Ahead                   | Number(2)       | period identifier            | The period ahead (1-99) for the forward limit planning              |
|              | Forward Limit Percent          | Number(12)<br>) | planning percentage          | Percentage of period's budget available to buyer at current time    |
| File Trailer | File Type Record Descriptor    | Char(5)         | FTAIL                        | Identifies file record type                                         |
|              | File Line Sequence Number      | Number(10)<br>) | Line number in file          | Keeps track of the record's position in the file by line number     |
|              | Control Number File Line Count | Number(10)<br>) | total detail lines           | sum of all transaction lines, not including file header and trailer |

#### Reject File

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

#### Error File

Standard Retek batch error handling 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**

Level Customization:

CLASS LEVEL (extrapolate for department level):

- Remove the subclass field from the input file specification.
- Remove subclass from the file detail structure.
- Remove local variable designated to hold subclass from the declarations above file\_process().
- In file\_process(), remove logic copying subclass from memory to a local variable.
- In validate\_record(), remove logic validating that subclass is numeric and not blank.
- In validate\_record(), call CLASS\_VALIDATE\_SQL.EXISTS instead of SUBCLASS\_VALIDATE\_SQL.EXISTS.
- In process\_record(), remove subclass from all update or insert logic.

454 MONTH LEVEL

- No modifications necessary.

JULIAN MONTH LEVEL

- No modifications necessary.

## New budget data and budget adjustments [otbupld]

### Design Overview

The purpose of this batch module is to accept new budget data and budget adjustments from an outside planning system. This data is then inserted into or used to update the budget figures on the OTB table.

Detail processing is very limited. If there is not a row on the table to update, a new row is inserted with the department, class, subclass, end-of-week date and budget amount for the specified order type as taken from the input file. Order types tracked by Retek are Non-Basic (N/B), Buyer Reorder of Basic (BRB) and Automatic Reorder of Basic (ARB). This module doesn't use Direct Store Delivery (DSD) orders since they don't have records on the OTB table. The date procedures CAL\_TO\_454 and CAL\_TO\_454\_HALF are used to calculate the half, month and week numbers of the record's end-of-week date, which are also held on the OTB table. If a budget amount is being adjusted, the old budget amount for the specified department, class, subclass, end-of-week date and order type is overwritten with the budget amount extracted from the input file.

Tables Affected:

| TABLE | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-------|-------|--------|--------|--------|--------|
| OTB   | No    | No     | Yes    | No     | No     |

### Scheduling Constraints

Processing Cycle: PHASE AD-HOC

Scheduling Diagram: This program is independent of all other batch processing and can be run at any time in the batch cycle.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

The logical unit of work for the otbupld module is the discrete budget transaction. Each row is uniquely identified by order type, department, class, subclass and end-of-week date. Processing of each row is independent and thus if an erroneous record is found during processing, only that record needs to be corrected and reprocessed.

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE\_ERROR library function.

The recommended commit counter setting is 10,000. A savepoint will be issued after each record is successfully processed. If a non-fatal error occurs, a rollback to the last savepoint will be issued so that the rejected record is not posted to the database. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

### Program Flow

```

main()
{
 accept the names of the input and reject files as arguments from the command line
 declare structures for the file header, detail and trailer records
 declare an application image array that includes g_l_rec_cnt and g_l_rej_cnt

 init()
 {
 open the input and reject files
 initialize restart/recovery
 call the library function get_record()
 if this fails, return FATAL ERROR
 }

 file_process()
 {
 LOOP
 read number of records indicated by g_l_restart_max_ctr into memory
 if the end of file character is reached then decrement the counter by 1 and set the end
 of file flag to TRUE
 LOOP
 read a record
 copy items from structs into varchar arrays and nullpad them
 validate_record()
 check that order type, department, class, subclass and end-of-week date are
 not NULL by calling field_has_value()
 check that order type is 'A', 'B', or 'N'
 check that the end-of-week date is a valid date by calling valid_date() and a
 valid end-of-week date by calling CAL_TO_454_LDOW
 check that department, class and subclass are numbers by calling
 valid_all_numeric() and call SUBCLASS_VALIDATE_SQL.EXIST to
 check that they exist on the subclass table
 check that the budget amount is a number by calling
 valid_all_numeric_signed()
 if record fails validation, return NON-FATAL ERROR
 if NON-FATAL ERROR is returned during validation
 call write_to_rej_file() to write the invalid record to the reject file
 increment g_l_rej_cnt
 move on to the next record
 process_record()
 call OTB_SQL.OTB_INSERTS
 if the function returns FALSE then return FATAL ERROR
 end LOOP
 increment the record counter by the number of records processed in the loop
 }
}

```

```

 set the application image array
 call the restart file commit process
 call restart_file_write()
 if the end of file flag = TRUE then exit
 end LOOP
 call process_dtl_ftail () to process the file trailer record
}

final()
{
 close restart/recovery logic
 close the reject file
}
}

```

### Shared Modules

SUBCLASS\_VALIDATE\_SQL.EXIST: This function will validate that a department/class/subclass combination exists on the subclass table.

CAL\_TO\_454\_LDOW: This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

OTB\_SQL.OTB\_INSERTS: This function performs the inserts into and updates of the OTB table. If there is no record on the OTB table for the specified department, class, subclass and end-of-week date, this package calculates the half, month and week numbers for the given end-of-week date using CAL\_TO\_454\_HALF and CAL\_TO\_454. It then inserts a new row with the data from the input file and the calculated date figures with all other rows assigned a value of zero. If there is a row already on the table, the specified order type's budget amount is overwritten with the number extracted from the input file.

### Function Level Description

main():

- Accept four runtime parameters at the command line: program name, user name/password, input file name, and reject file name. Copy the input and reject file names into strings for later use.

init():

- Open the input and reject files.
- Initialize restart/recovery.
- Call the get\_record() function to read the input file header record

file\_process():

- Set up a buffer in memory to hold the number of input file rows indicated by g\_l\_restart\_max\_counter.
- Use a while loop to read g\_l\_restart\_max\_counter number of rows into the buffer.
  - If the end of file character is reached, decrement the record counter by one (to avoid processing file trailer record) and set the end-of-file flag to TRUE.
  - Set up a FOR loop to read each record from the buffer into the detail record structure and process it.
    - Copy each record field into a local variable for processing.

- Call the `validate_record()` function to validate record fields.
- If `validate_record()` returns a value of 1 (an error was encountered) then handle NON-FATAL ERROR.
- Call the `process_record()` function
- If record processing fails, handle FATAL ERROR.
- End FOR loop.
- Increment the record counter by the number of records processed in the loop.
- Set the application image array for restart/recovery and call the `restart_file_commit()` function to save it.
- Call the `restart_file_write()` function to write the rejected records to the rejected record file.
- If the end-of-file flag has been set to TRUE, exit the while loop.
- End while loop.
- Call the `process_dtl_ftail()` function to process the file trailer record

`validate_record()`:

- Ensure that the order type, department, class, subclass, and end-of-week date fields all contain values by calling the function `field_has_value()` for each. If one of the fields is NULL, `field_has_value()` will return 1 and the record should be written to the reject file.
- Check that the order type field contains 'A', 'B' or 'N'.
- Check that the input record's end-of-week date is a valid date in the correct format by calling the `valid_date()` function. If not, `valid_date()` will return 1 and the record should be written to the reject file. If `valid_date()` returns 0, check that the end-of-week date is a valid end-of-week date using `CAL_TO_454_LDOW`.
- Ensure that the department, class and subclass are all numeric by calling `valid_all_numeric()` for each and, if no error is returned from that function, call `SUBCLASS_VALIDATE_SQL.EXIST` to validate that the record's department/class/subclass combination exists on the subclass table.
- Check that the budget amount is numeric by calling `valid_all_numeric_signed()`.

`process_record()`:

- Call `OTB_SQL.OTB_INSERTS` to insert or update the OTB table.

`final()`:

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Rollback to previous physical commit point.
- Exit the program.

on NON-FATAL ERROR:

- Rollback to the previous savepoint.
- Call the function `write_to_rej_file()` to write the rejected record to the rejected record file.
- Increment `g_l_rej_cnt`.

## I/O Specification

Input File:

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

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDD' in order to validate it using `valid_date()`.

A sample detail record would look like the following:

---

FDETL123456789012345678901234A123412341234199501151234567890123456789

| 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      | Number(10) | Line number in file          | Keeps track of the record's position in the file by line number                                                                   |
|              | File Type Definition           | Char(4)    | OTBI                         | Identifies file as 'OTB Import'                                                                                                   |
|              | File Create Date               | Date       | create date                  | Date file was written by external system                                                                                          |
| File Detail  | File Type Record Descriptor    | Char(5)    | FDETL                        | Identifies file record type                                                                                                       |
|              | File Line Sequence Number      | Number(10) | Line number in file          | Keeps track of the record's position in the file by line number                                                                   |
|              | Transaction Set Control Number | Number(14) | specified by external system | Used to force unique transaction check                                                                                            |
|              | Order Type                     | Char(1)    | A, B, N                      | Order type budgeted for: specified as A for auto-basic replenishment, B for buyer-basic replenishment, and N for non-basic orders |
|              | Department                     | Number(4)  | dept identifier              | The id number of a department                                                                                                     |
|              | Class                          | Number(4)  | class identifier             | The id number of a class within the department given                                                                              |
|              | Subclass                       | Number(4)  | subclass identifier          | The id number of a subclass within the class given                                                                                |
|              | Eow Date                       | Date       | end of week date             | The end of week date for the budgeted week (formatted as 'YYYYMMDD')                                                              |
|              | Budget Amount                  | Number(20) | budget amount                | Budgeted amount for the specified order type/week                                                                                 |
| File Trailer | File Type Record               | Char(5)    | FTAIL                        | Identifies file record type                                                                                                       |

|  | Descriptor                     |            |                     |                                                                     |
|--|--------------------------------|------------|---------------------|---------------------------------------------------------------------|
|  | File Line Sequence Number      | Number(10) | Line number in file | Keeps track of the record's position in the file by line number     |
|  | Control Number File Line Count | Number(10) | total detail lines  | sum of all transaction lines, not including file header and trailer |

**Reject File:**

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

**Error File:**

Standard Retek batch error handling 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****Level Customization:****CLASS LEVEL (extrapolate for department level):**

- Remove the subclass field from the input file specification.
- Remove subclass from the file detail structure.
- Remove the local variable designated to hold subclass from the declarations above file\_process().
- In file\_process(), remove the logic copying subclass from memory to a local variable.
- In validate\_record(), remove all logic validating that subclass is numeric.
- In validate\_record(), call CLASS\_VALIDATE\_SQL.EXISTS instead of SUBCLASS\_VALIDATE\_SQL.EXISTS.
- In process\_record(), remove subclass from the call to OTB\_SQL.OTB\_INSERTS.

**454 MONTH LEVEL:**

- Change all variable names like 'eow\_date' to eom\_date.
- Call CAL\_TO\_454\_LDOM to check that the input record's end-of-month date is a valid 454 calendar end-of-month date.

**JULIAN MONTH LEVEL:**

- Change all variable names like 'eow\_date' to eom\_date.
- Call CAL\_TO\_CAL\_LDOM to check that the input record's end-of-month date is a valid Julian calendar end-of-month date.

## Clearance pricing POS download [pccdnld]

### Design Overview

The purpose of the Clearance Download (pccdnld) module is to send clearance markdown retail prices to the point of sales system. Clearance markdowns that are to take effect within the predetermined number of days are written out to the POS\_MODS table, which will be used as an interface point with the point of sale. Also, the clearance detail records are updated with the current date as a downloaded date. The number of days prior to taking effect that clearance markdowns are downloaded is maintained as a system option.

| Table                  | Index | Select | Insert | Update | Delete |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 | No    | Yes    | No     | No     | No     |
| UNIT_OPTIONS           | No    | Yes    | No     | No     | No     |
| CLEAR_SUSP_DETAIL      | No    | Yes    | No     | Yes    | No     |
| CLEAR_SUSP_HEAD        | No    | Yes    | No     | No     | No     |
| PRICE_ZONE_GROUP_STORE | No    | Yes    | No     | No     | No     |
| ITEM_LOC               | No    | Yes    | No     | No     | No     |
| ITEM_MASTER            | No    | Yes    | No     | No     | No     |
| ITEM_ZONE_PRICE        | No    | Yes    | No     | No     | No     |
| POS_MODS               | No    | No     | Yes    | No     | No     |
| RESTART_CONTROL        | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Clearance

### Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

Driving Cursor

```
EXEC SQL DECLARE c_clear CURSOR FOR
 SELECT csd.item,
 csd.clearance,
 TO_CHAR(csd.active_date, 'DDMMYYYY'),
 csd.zone_group_id,
 csd.zone_id,
 csd.unit_retail,
 pzgs.store,
```



```

 ROWIDTOCHAR(csd.rowed),
 csd.selling_uom,

FROM clear_susp_detail csd,
 clear_susp_head csh,
 price_zone_group_store pzgs,
 item_master im,
 restart_control rc
WHERE :prc_ext_days >= (csd.active_date - TO_DATE(:vdate, 'DDMMYYYY'))
 AND csd.active_date >= TO_DATE(:vdate + 1, 'DDMMYYYY')
 AND csh.status = 'A'
 AND csd.downloaded_date is NULL
 AND csd.clearance = csh.clearance
 AND pzgs.zone_group_id = csd.zone_group_id
 AND pzgs.zone_id = csd.zone_id
 AND csd.item = im.item
 AND rc.driver_name = :ora_restart_driver_name
 AND rc.num_threads = :ora_restart_num_threads
 AND rc.program_name = 'pccdnld'
 AND restart_thread_return(csh.clearance, rc.num_threads) = :ora_restart_thread_val
 AND im.status = 'A'
 AND im.item_level = im.tran_level
 AND im.pack_ind = 'N'
ORDER BY csd.clearance,
 csd.item,
 csd.zone_group_id,
 csd.zone_group_id;

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Retek clearance pricing update [pccext]

### Design Overview

The function of this program is to extract the new clearance event retail pricing from the clearance suspense tables and update the retail information in Retek. Clearance events that are to take place on the following business day are selected and the appropriate tables are updated with the new pricing information and with a new clearance indicator value. Transaction-level stock ledger, price history, supplier history and clearance sell through records are also written. If an item/location is on replenishment, using the RMS replenishment functionality, then when the item/location goes on clearance, it is taken off of replenishment.

| Table                  | Index | Select | Insert | Update | Delete |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 | No    | Yes    | No     | No     | No     |
| CLEAR_SUSP_DETAIL      | No    | Yes    | No     | No     | No     |
| CLEAR_SUSP_HEAD        | No    | Yes    | No     | Yes    | No     |
| ITEM_LOC               | No    | Yes    | No     | Yes    | No     |
| ITEM_LOC_SOH           | No    | Yes    | No     | No     | No     |
| ITEM_MASTER            | No    | Yes    | No     | No     | No     |
| ITEM_SUPPLIER          | No    | Yes    | No     | No     | No     |
| CLEAR_SELL_THRGH       | No    | No     | Yes    | No     | No     |
| TRAN_DATA              | No    | No     | Yes    | No     | No     |
| SUP_DATA               | No    | No     | Yes    | No     | No     |
| PRICE_HIST             | No    | No     | Yes    | No     | No     |
| PRICE_ZONE             | No    | No     | Yes    | No     | No     |
| PRICE_ZONE_GROUP_STORE | Yes   | Yes    | No     | No     | No     |
| REPL_DAY               | No    | No     | No     | No     | Yes    |
| REPL_ITEM_LOC          | No    | No     | No     | No     | Yes    |
| V_RESTART_CLEARANCE    | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS         | No    | Yes    | No     | No     | No     |

Indexes: CLEAR\_SUSP\_HEAD(reason)

ITEM\_MASTER(standard\_uom)

ITEM\_SUPPLIER(supplier)

PRICE\_ZONE\_GROUP\_STORE(zone\_group\_id, zone\_id),  
(store)

### Scheduling Constraints

Processing Cycle: Phase 3

Scheduling Diagram: N/A

Pre-Processing: pctranex should complete before the pccext module begins.

Post-Processing: pccrext should not start until pccext has completed.

Threading Scheme: Clearance

### Restart Recovery

```
EXEC SQL DECLARE c_clear CURSOR FOR
 SELECT csh.clearance,
 csh.reason,
 csh.status,
 im.item,
 csd.unit_retail,
 csd.selling_uom,
 im.standard_uom,
 TO_CHAR(csd.active_date, 'DDMMYYYY'),
 pzgs.store,
 pz.currency_code,
 im.pack_ind,
 ';' || TO_CHAR(csh.clearance) ||
 ';' || TO_CHAR(csd.item) ||
 ';' || TO_CHAR(pzgs.store)
 FROM v_restart_clearance rv,
 price_zone_group_store pzgs,
 price_zone pz,
 clear_susp_detail csd,
 clear_susp_head csh,
 item_master im
 WHERE csh.status = 'A'
 AND csh.clearance = csd.clearance
 AND csd.active_date = to_date(:ps_tomorrow, 'DDMMYYYY')
 AND csd.zone_group_id = pzgs.zone_group_id
 AND csd.zone_id = pzgs.zone_id
 AND csd.zone_group_id = pz.zone_group_id
 AND csd.zone_id = pz.zone_id
 AND csd.item = im.item
 AND im.item_level = im.tran_level
 AND im.status = 'A'
 AND im.pack_ind = 'N'
 AND rv.driver_value = csh.clearance
 AND rv.num_threads = :ora_restart_num_threads
 AND rv.thread_val = :ora_restart_thread_val
 AND (csh.clearance > NVL(:ora_restart_clearance, -999) OR
 (csh.clearance = :ora_restart_clearance AND
 (csd.sku > :ora_restart_item OR
 (csd.sku = :ora_restart_item AND
 (pzgs.store > :ora_restart_store))))))
 ORDER BY csh.clearance, sku, store;
```

### Program Flow

N/A

### Shared Modules

STKLEDGR\_SQL.TRAN\_DATA\_INSERT

PRCCHGTYP

GET\_STANDARD\_UOM

UOM\_SQL.CONVERT

CURRENCY\_SQL.CONVERT

### **Function Level Description**

#### Init()

Initialize restart recovery.

Determine vdate, contract\_ind and currency\_code.

Call function size\_arrays to allocate memory size to arrays.

#### Process()

Select all clearance details with an active date of tomorrow. It will call the function get\_item\_info to obtain the clearance items dept, class, subclass and supplier. Function update\_item\_loc is then called to convert the new selling unit retail to the standard unit retail, compute the tran type, insert into tran\_data, clear\_sell\_through, price\_hist and sup\_data, and update item\_loc.

#### Get\_Item\_Info()

This function selects the items dept, class, subclass, and supplier and fetches them into variables to be inserted into tables later in program.

#### Update\_Item\_Loc()

This function selects unit retail, unit cost, and stock on hand info from the ITEM\_LOC and ITEM\_LOC\_SOH tables. It then converts the new unit retail for clearance items to standard unit retail by calling the convert\_new\_unit\_retail function. The function also calls functions calc\_tran\_type to compute tran type, insert\_tran to insert into the TRAN\_DATE and SUP\_DATA tables, clear\_sell\_through to insert into the CLEAR\_SELL\_THROUGH table, and price\_hist to insert into the PRICE\_HIST table. The function also updates the ITEM\_LOC table with the new selling uom, selling unit retail and standard unit retail.

#### Convert\_New\_Unit\_Retail()

Converts the new selling unit retail to the standard unit retail by calling the UOM\_SQL.CONVERT package.

#### Calc\_Tran\_Type()

This function computes the tran type by calling the stored procedure PRCCHGTYP.

#### Insert\_Tran()

If the selected item's stock qty > 0, this function calls the package STKLEDGR\_SQL.TRAN\_DATA\_INSERT to insert into the TRAN\_DATA table. If the local currency is not the same as the primary currency, this function calls function convert\_currency to convert to the primary currency. Then insert into the SUP\_DATA table. If there is a second tran type and tran type amt, then the function calls STKLEDGR\_SQL.TRAN\_DATA\_INSERT again, and inserts into the SUP\_DATA table again.

#### Convert\_Currency()

Function is called to compute the dollar amount in primary currency by calling the package CURRENCY\_SQL.CONVERT.

#### Price\_Hist()

This function inserts into the PRICE\_HIST table.

#### Clear\_Sell\_Through()

This function inserts into the CLEAR\_SELL\_THROUGH table.

#### Post\_Change()

This function deletes an item from the REPL\_DAY and REPL\_ITEM\_LOC tables when the item goes on clearance.

#### Size\_Arrays()

This function allocates memory sizes to the arrays that have been fetched into by the driving cursor.

#### Final()

This function closes restart recovery.

### **I/O Specification**

N/A

### **Technical Issues**

N/A

## Clearance event deletions [pccprg]

### Design Overview

The Clearance Purge (pccprg) module removes completed, canceled, and rejected clearance events from the system. Before a clearance event can be deleted, a pre-determined number of months must have passed since the reset date of the clearance. The pre-determined number of months is based on the value entered for the system option Clearance Retention Months.

Tables affected:

| Tables             | Index | Select | Insert | Update | Delete |
|--------------------|-------|--------|--------|--------|--------|
| CLEAR RESET CALC   | No    | Yes    | No     | No     | Yes    |
| CLEAR SELL THROUGH | No    | No     | No     | No     | Yes    |
| CLEAR SUSP DTL     | No    | Yes    | No     | No     | Yes    |
| CLEAR SUSP HEAD    | No    | Yes    | No     | No     | Yes    |
| PERIOD             | No    | Yes    | No     | No     | No     |
| PRICE EVENT ZONE   | No    | No     | No     | No     | Yes    |
| PRICE EVENT SKU    | No    | No     | No     | No     | Yes    |
| UNIT OPTIONS       | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

N/A

### Program Flow

N/A

### Shared Modules

N/A

### Function Level Description

N/A

### I/O Specification

N/A

Technical Issues

N/A

## Clearance reset pricing POS extract [pccrdnld]

### Design Overview

The Clearance Reset Pricing Information (pccrdnld) module is used to send clearance reset pricing information to the point of sale system. When a clearance event is within a pre-determined number of days of its reset date, this program will gather current clearance pricing information and the pricing information to which an item will be reset. This information will be written out to a table that will be used as an interface point with the point of sale. Clearance records will be updated to indicate that they have had reset prices downloaded to the point of sale by setting a reset downloaded date with the current date.

| Tables                 | Index | Select | Insert | Update | Delete |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 | No    | Yes    | No     | No     | No     |
| UNIT_OPTIONS           | No    | Yes    | No     | No     | No     |
| CLEAR_RESET_CALC       | No    | Yes    | No     | Yes    | No     |
| CLEAR_SUSP_HEAD        | No    | Yes    | No     | No     | No     |
| CLEAR_SUSP_DETAIL      | No    | Yes    | No     | No     | No     |
| PRICE_ZONE_GROUP_STORE | Yes   | Yes    | No     | No     | No     |
| POS_MODS               | No    | No     | Yes    | No     | No     |
| ITEM MASTER            | Yes   | Yes    | No     | No     | No     |
| ITEM LOC               | Yes   | Yes    | No     | No     | No     |
| ITEM_ZONE_PRICE        | Yes   | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: pcdnld should complete processing before this module begins

Post-Processing: N/A

Threading Scheme: Clearance

### Restart Recovery

The driving cursor was changed because of changes to the tables and also the system indicator is no longer used.

```
EXEC SQL DECLARE c_get_reset CURSOR FOR
 SELECT crc.clearance,
 crc.item,
 TO_CHAR(crc.reset_date, 'DDMMYYYY'),
 pzgs.store,
 ROWIDTOCHAR(crc.rowid),
 im.item_level,
 ':' || TO_CHAR(crc.clearance) ||
 ':' || crc.item ||
```



```

 ',' || TO_CHAR(crc.zone_group_id) ||
 ',' || TO_CHAR(crc.zone_id)
FROM clear_reset_calc crc,
 clear_susp_head csh,
 price_zone_group_store pzgs,
 item_master im,
 v_restart_clearance rv
WHERE crc.reset_date is NOT NULL
AND (
 (csh.status = 'C'
 AND EXISTS (SELECT 'x'
 FROM clear_susp_detail csd
 WHERE csd.clearance = csh.clearance
 AND csd.item = crc.item
 AND csd.zone_group_id = crc.zone_group_id
 AND csd.zone_id = crc.zone_id
 AND csd.downloaded_date is NOT NULL))
 OR
 (csh.status = 'A'
 AND :pd_pos_extract_days >= (crc.reset_date - to_date(:ps_vdate,'DDMMYYYY'))
)
)
AND csh.clearance = crc.clearance
AND crc.zone_group_id = pzgs.zone_group_id
AND crc.zone_id = pzgs.zone_id
AND crc.reset_downloaded_date is NULL
AND crc.item = im.item
AND im.status = 'A'
AND im.pack_ind = 'N'
AND im.item_level = im.tran_level
AND rv.driver_value = csh.clearance
AND rv.driver_name = :ps_restart_driver_name
AND rv.num_threads = :pi_restart_num_threads
AND rv.thread_val = :pi_restart_thread_val
ORDER BY crc.clearance,
 crc.item,
 crc.zone_group_id,
 crc.zone_id;

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Retek clearance reset price update [pccrex]

## Design Overview

The function of this program is to reset the ITEM/store retail prices back to the regular retail prices. Clearance events with reset dates on the following business day are selected and the appropriate item/location tables are updated with the regular retail price and a new clearance indicator value. Transaction-level stock ledger, price history, supplier history and clearance sell through records are also written.

| Tables                 | Index | Select | Insert | Update | Delete |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 | No    | Yes    | No     | No     | No     |
| CLEAR_SUSP_DETAIL      | No    | Yes    | No     | No     | No     |
| CLEAR_SUSP_HEAD        | No    | Yes    | No     | Yes    | No     |
| PRICE_SUSP_DETAIL      | Yes   | Yes    | No     | No     | No     |
| PRICE_SUSP_HEAD        | No    | Yes    | No     | No     | No     |
| ITEM_MASTER            | Yes   | Yes    | No     | No     | No     |
| ITEM_LOC               | Yes   | Yes    | No     | Yes    | No     |
| ITEM_LOC_SOH           | Yes   | Yes    | No     | No     | No     |
| PRICE_ZONE             | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS         | No    | Yes    | No     | No     | No     |
| TRAN_DATA              | No    | No     | Yes    | No     | No     |
| SUP_DATA               | No    | No     | Yes    | No     | No     |
| PRICE_HIST             | No    | No     | Yes    | No     | No     |
| ITEM_ZONE_PRICE        | Yes   | Yes    | No     | No     | No     |
| PRICE_ZONE_GROUP_STORE | Yes   | Yes    | No     | No     | No     |
| CLEAR_RESET_CALC       | No    | Yes    | No     | No     | No     |
| ITEM_SUPPLIER          | No    | Yes    | No     | No     | No     |

```

Indexes: PRICE_SUSP_DETAIL(diff_id)
 ITEM_MASTER(item_parent)
 (item_grandparent)
 ITEM_LOC(loc)
 ITEM_LOC_SOH(loc)
 ITEM_ZONE_PRICE(zone_group_id, zone_id),
 PRICE_ZONE_GROUP_STORE(zone_group_id, zone_id),
 (zone_group_id, store),

```

## Scheduling Constraints

Processing Cycle: Phase 3

Scheduling Diagram: N/A

Pre-Processing: pccxt and pccxt must complete before this module begins

Post-Processing: N/A

Threading Scheme: Clearance

### Restart Recovery

```
EXEC SQL DECLARE c_reset CURSOR FOR
 SELECT csh.clearance,
 csh.status,
 csh.reason,
 crc.item,
 crc.zone_group_id,
 crc.zone_id,
 TO_CHAR(crc.reset_date, 'DDMMYYYY'),
 pzgs.store,
 '||TO_CHAR(csh.clearance)||'
 '||crc.item||'
 '||TO_CHAR(pzgs.store)
FROM v_restart_clearance rv,
 price_zone_group_store pzgs,
 clear_reset_calc crc,
 clear_susp_head csh
WHERE ((csh.status = 'A'
 AND crc.reset_date = to_date(:tomorrow, 'DDMMYYYY'))
 OR (csh.status = 'C'
 AND EXISTS (SELECT 'x'
 FROM clear_susp_detail csd
 WHERE csd.clearance = csh.clearance
 AND csd.item = crc.item
 AND csd.zone_group_id = crc.zone_group_id
 AND csd.zone_id = crc.zone_id
 AND csd.downloaded_date is not null)))
 AND csh.clearance = crc.clearance
 AND crc.zone_group_id = pzgs.zone_group_id
 AND crc.zone_id = pzgs.zone_id
 AND rv.driver_value = csh.clearance
 AND rv.driver_name = :ps_restart_driver_name
 AND rv.num_threads = :ps_restart_num_threads
 AND rv.thread_val = :ps_restart_thread_val
 AND (csh.clearance > NVL(:ps_restart_clearance, -999) OR
 (csh.clearance = :ps_restart_clearance AND
 (crc.item > :ps_restart_item OR
 (crc.item = :ps_restart_item AND
 (pzgs.store > :ps_restart_store))))))
ORDER BY csh.clearance, item, store;
```

### Program Flow

N/A

### Shared Modules

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Price change POS extract [pcdnld]

### Design Overview

The Price Change Download (pcdnld) module selects permanent price change records, which are set to go into effect a pre-determined number of days after today. The details of these records are then written out to the POS\_MODS table, which is used as an interface point to with the point of sale system. The number of days before the price change goes into effect that the details are downloaded to the POS system is determined by the system option pos\_extract\_days on UNIT\_OPTIONS. After this program has processed a price change, the status of the price change is changed to Extracted.

| Table                  | Index | Select | Insert | Update | Delete |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 | No    | Yes    | No     | No     | No     |
| UNIT_OPTIONS           | No    | Yes    | No     | No     | No     |
| PRICE_SUSP_HEAD        | Yes   | Yes    | No     | Yes    | No     |
| PRICE_SUSP_DETAIL      | Yes   | Yes    | No     | Yes    | No     |
| V_RESTART_PRICE_CHG    | Yes   | Yes    | No     | No     | No     |
| PRICE_ZONE_GROUP_STORE | Yes   | Yes    | No     | No     | No     |
| ITEM_ZONE_PRICE        | Yes   | Yes    | No     | No     | No     |
| POS_MODS               | No    | No     | Yes    | No     | No     |
| CLEAR_SUSP_HEAD        | Yes   | Yes    | No     | No     | No     |
| CLEAR_SUSP_DETAIL      | Yes   | Yes    | No     | No     | No     |
| CLEAR_RESET_CALC       | Yes   | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Pccrdnld should run only after this module has completed

Threading Scheme: Price change

### Restart Recovery

The restart string is price change + ITEM + store.

Records will be fetched from an outer loop into arrays (sized according to restart\_max\_ctr on RESTART\_CONTROL) and each record will be individually processed in a loop. Commits will occur after each record in the for-loop has been processed and before the next set of records are fetched into the arrays. This can happen because status is part of the select criteria on both the price change header and detail tables.

**Program Flow**

N/A

**Shared Modules**

TICKET\_SQL.PRICE\_CHANGE

**Function Level Description****init()**

This function will, in addition to performing standard restart tasks, fetch the current date from the period table and the number of days buffer required by the point of sales system for pricing modifications (pos\_extract\_days).

**process()**

This function selects the impending price change records (price change + ITEM + store combinations) for processing. The records are selected into arrays (size determined by commit\_max\_ctr on the RESTART\_CONTROL table) and then each record in the array is processed by a for-loop.

Inside this internal loop the ITEM information is retrieved from the ITEM\_MASTER table (item, status, & pack\_ind). If the status is approved, then process\_item is called. If the price change number has changed since the last check then the psh\_update() function is called to update the header status. When the PRICE\_SUSP\_DETAIL rowid changes then a row is added to an array that will update the status of the PRICE\_SUSP\_DETAIL table.

After the internal loop has processed the individual records, the pos\_mods\_insert() function is called to insert POS\_MODS records created in the process\_item() function and the price\_susp\_detail rows that have completed processing are updated by calling the psd\_update() function.

**process\_item()**

This function will select the current/old retail pricing information from ITEM\_ZONE\_PRICE. First it is determined whether the item is on clearance at the price change location by calling the item\_on\_clearance() function. If it is, then the function is exited (price information is not retrieved and a POS\_MODS record is not written). If the price change is a regular price change then the information can be selected using the zone information (zone\_id & zone\_group\_id) on the price\_change tables. Otherwise, the “old” zone information must be retrieved from the PRICE\_ZONE\_GROUP\_STORE table based on the store location of the price change. This table is joined with the ITEM\_ZONE\_PRICE table to retrieve old retail information. Once the old retail information is retrieved then an array to insert into POS\_MODS table is filled in the fill\_pos\_mods() function.

After the call to the fill\_pos\_mods function, the package TICKET\_SQL.PRICE\_CHANGE is called to create the appropriate records for printing tickets based on the price change.

**fill\_pos\_mods()**

The main task of this function is to fill elements of arrays with information that will be inserted into the POS\_MODS table. In addition to filling the arrays, it will also resize the insert (incrementing the size with the commit\_max\_ctr on RESTART\_CONTROL) arrays when the array indexes will be extended.

**pos\_mods\_insert()**

Insert into POS\_MODS from the insert arrays. The base insert adds item and pricing information derived from the price change tables. An additional insert adds reference item information to the POS\_MODS table.

**Is\_item\_on\_clearance()**

This function checks the clearance suspense and reset tables to see if the current item is on clearance at the current store.

**size\_input\_arrays()**

This function sizes two different sets of arrays. The arrays into which the price change information is fetched are allocated memory. Also the array used to hold the PRICE\_SUSP\_DETAIL rowids whose status will be updated is sized here. The commit\_max\_ctr field, on the RESTART\_CONTROL table, determines the size of the arrays.

**psd\_update()**

The PRICE\_SUSP\_DETAIL table is updated, the status set to 'E', for all of the records whose rowids are in the update array.

**I/O Specification**

N/A

**Technical Issues**



## Retek price change update [pcext]

### Design Overview

The Price Change Extract (pcext) module selects permanent price change records, which are set to go into effect tomorrow and updates the RMS item/location and item/zone tables with the new prices. The price changes processed by this program include both price change events set up in the pricing module and price changes generated by an items zone group changing. In addition to the updates of the retail prices in the system, records are also written for price history and transaction-level stock ledger. If an item/zone price change results in a change to the price in the base zone, then the retail at all warehouse locations will also be updated. (Note: If the price change is for an item above the transaction level, then this price change also affects all of the transaction level items which are below this item.) For ITEM zone group changes, all stores are updated with the price from the base zone. For regular price changes the new retail is applied to the appropriate stores within the zone. Once this program has processed a price change, the status of the price change is changed to Delete.

| Tables                 | Index | Select | Insert | Update | Delete |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 | No    | Yes    | No     | No     | No     |
| PRICE_SUSP_HEAD        | No    | Yes    | No     | Yes    | No     |
| PRICE_SUSP_DETAIL      | No    | Yes    | No     | Yes    | No     |
| PRICE_ZONE             | No    | Yes    | No     | No     | No     |
| PRICE_ZONE_GROUP_STORE | Yes   | Yes    | No     | No     | No     |
| ITEM_ZONE_PRICE        | Yes   | Yes    | Yes    | Yes    | Yes    |
| ITEM_SUPPLIER          | No    | Yes    | No     | No     | No     |
| ITEM_MASTER            | Yes   | No     | No     | Yes    | No     |
| ITEM_LOC               | Yes   | No     | No     | Yes    | No     |
| ITEM_LOC_SOH           | Yes   | No     | No     | Yes    | No     |
| PRICE_HIST             | No    | No     | Yes    | No     | No     |
| SUP_DATA               | No    | No     | Yes    | No     | No     |
| TRAN_DATA              | No    | No     | Yes    | No     | No     |

### Scheduling Constraints

Processing Cycle: Phase 3 - after pccrxt

Scheduling Diagram: No changes

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: The threading mechanism will be price change. The v\_restart\_price\_chg view will be used.

### Restart Recovery

The Logical Unit of Work for pcext is price change transaction type (Regular or Zone price changes), price change and item. Processing is done as master-detail, however, price\_susp\_detail

records are updated as they are processed and so commits can occur at any time during the processing (i.e. the commit process does not need to occur after an update to the header record).

### Shared Modules

ITEM\_ATTRIB\_SQL.GET\_BASE\_COST\_RETAIL – gets the base cost/retail in the primary currency.

SUPP\_ITEM\_SQL.GET\_PRI\_SUP\_COST – gets primary supplier and unit cost for a given item or item/location.

CURRENCY\_SQL.CONVERT\_BY\_LOCATION – converts currencies

UOM\_SQL.CONVERT – converts to the standard unit retail.

STKLEDGR\_SQL.TRAN\_DATA\_INSERT - inserts records to tran\_data table

PRCCHGTYP - determines type of price change (markup, markdown, etc.)

### Function Level Description

#### init()

This function fetches the system date for tomorrow.

#### process()

This function will loop through the records retrieved by the driving cursor. Each item within a price change will be processed. Once the item has been processed, the price\_susp\_detail record is updated to indicate that it has been processed. If the price change is a zone price change, then the item\_zone\_price table is purged of records for that item that have the old zone information. Finally, if all of the items within the price change have been processed the price\_susp\_head record status is updated to deleted (to be deleted). Functions called from process() include: SIZE\_ARRAYS(), GET\_ADDITIONAL\_ITEM\_INFO(), PROCESS\_ZONE(), PROCESS\_ITEM(), POST\_ALL(), SET\_PSH\_PSD\_UPDATE(), RETEK\_FORCE\_COMMIT(), and SET\_IZP\_DELETE().

#### Get\_additional\_item\_info()

Fetches additional item information which is not fetched by the driving cursor. Includes calls to get\_stock\_on\_hand(), get\_cost\_at\_location(), get\_base\_zone(), and get\_old\_retails().

#### Get\_stock\_on\_hand()

Retrieves the total of the stock\_on\_hand plus the in\_transit\_qty and whether the item is on clearance or not for each location selected.

#### Get\_cost\_at\_location()

If the item is not a pack item, the unit cost is selected from the item\_loc table. If it is a pack item, a call is made to package function GET\_PRI\_SUP\_COST.

#### Get\_base\_zone()

For regular price changes, the base zone can be retrieved from the item\_zone\_price table, but for zone price changes the base zone must be retrieved from the price\_zone table.

#### Get\_old\_retails()

Gets old pricing information from the item\_zone\_price table and fills the pos\_mods array for future inserts.

**Get\_base\_cost\_retail\_prim()**

Called only if the current zone is the base zone. Gets the base cost/retail in the primary currency.

**Vendor\_funded\_credit()**

-- Not applicable in phase 2 --

**Convert\_uom()**

Wrapper function to call package function UOM\_SQL.CONVERT. (In order to use the package function to convert unit retails, the uom's are inversed.)

**Set\_psh\_psd\_update()**

Updates the array pt\_psh\_psd\_update with the new rowid and changes the status to 'D' for deleted. It then increments the counter ior\_psh\_psd\_update.l\_count by 1.

**Post\_psh\_psd\_update()**

Contains the SQL statements which updates the status of the price\_susp\_detail and price\_susp\_head tables to a status of 'D' for deleted.

**Set\_izp\_delete()**

Updates the array par\_izp\_delete with the new item and zone\_group\_id, and then increments the counter par\_izp\_delete.l\_count by 1.

**Post\_izp\_delete()**

Contains the SQL statements which delete records from table item\_zone\_price.  
The counter par\_izp\_delete.l\_count is then set to 0.

**Set\_im\_update()**

Updates the array par\_im\_update with the new rowid and retail\_zone\_group\_id and then increments the counter par\_im\_update.l\_count by 1.

**Post\_im\_update()**

Contains the SQL statements that update the retail\_zone\_group\_id, last\_update\_datetime, and last\_update\_id on the item\_master table.  
The counter par\_im\_update.l\_count is then set to 0.

**Set\_izp\_update()**

Updates the array par\_izp\_update with the new item, zone\_group\_id, zone\_id, unit\_retail, selling\_unit\_retail, selling\_uom, multi\_units, multi\_unit\_retail, and multi\_selling\_uom. The counter par\_izp\_update.l\_count is then incremented by 1.

**Post\_izp\_update()**

Contains the SQL statements that update the unit\_retail, selling\_unit\_retail, selling\_uom, multi\_units, multi\_unit\_retail, multi\_selling\_uom, last\_update\_datetime, and last\_update\_id on the table item\_zone\_price. The counter par\_izp\_update.l\_count is then set to 0.

**Set\_il\_update()**

Updates the array par\_il\_update with the new rowid, unit\_retail, selling\_unit\_retail, and selling\_uom and then increments the counter par\_il\_update.l\_count by 1.

**Post\_il\_update()**

Contains the SQL statements that update the unit\_retail, selling\_unit\_retail, selling\_uom, last\_update\_datetime, and last\_update\_id on the item\_loc table. Counter par\_il\_update.l\_count is then set to 0.

**Set\_izp\_insert()**

Updates the array `par_izp_insert` with the new `item`, `zone_group_id`, `zone_id`, `unit_retail`, `selling_unit_retail`, `selling_uom`, `multi_units`, `multi_unit_retail`, `multi_selling_uom`, and `base_retail_ind` and then increments the counter `par_izp_insert.l_count` by 1.

#### **Post\_izp\_insert()**

Contains the SQL statements that insert records into the `item_zone_price` table. The counter `par_izp_insert.l_count` is then set to 0.

#### **Set\_ph\_insert()**

Updates the array `par_ph_insert` with the new `tran_type`, `reason`, `item`, `loc`, `loc_type`, `unit_cost`, `unit_retail`, `selling_unit_retail`, `selling_uom`, `action_date`, `multi_units`, `multi_unit_retail`, and `multi_selling_uom` and then increments the counter `par_ph_insert.l_count` by 1.

#### **Post\_ph\_insert()**

Contains the SQL statements that insert records into the `price_hist` table. The counter `par_ph_insert.l_count` is then set to 0.

#### **Set\_sd\_insert()**

Updates the array `par_sd_insert` with the new `dept`, `supplier`, `day_date`, `tran_type`, and `amount` and then increments the counter `par_sd_insert.l_count` by 1.

#### **Post\_sd\_insert()**

Contains the SQL statements that insert records into the `sup_data` table. The counter `par_sd_insert.l_count` is then set to 0.

#### **Post\_all()**

Includes calls to additional functions which post all changes to the database.

#### **Process\_zone()**

Does any processing that needs to happen only once per zone and performs housekeeping for the previous Logical Unit of Work

#### **Common\_zone\_processing()**

This does the work that is common to both if branches in function `process_zone()`, such as converting the selling unit retail to the standard unit retail and prepares the data for insert/update into table `item_zone_price`. If a price is not changing, it is set to the old price. If the item is in approved status, it calls the function `set_ph_insert` to insert record into table `price_hist`.

#### **Process\_item()**

Implements business logic on the item whose price is changing. The item is skipped if it is a clearance item. If the zone group has changed, the table `item_master` is updated with the new zone group. Otherwise, if it is a regular price change, it ensures that the zone group didn't change. If the unit retail has changed, the single price changed and there exists an `item_loc` record and the item is not a pack, the function `set_il_update()` is called to update the `item_loc` table. If the item is in approved status, and an `item_loc` record exists, call function `set_ph_insert()` to insert into table `price_hist`.

#### **Convert\_by\_loc()**

The `CURRENCY_SQL.CONVERT` package will convert currencies according to passed parameters. The intent of this function is to reduce the amount of code by allowing the package to be called generically from anywhere in `pcext`.

#### **Update\_wh\_base()**

Loop through all item/warehouse combinations. First convert the new unit retail to the warehouse currency by calling `convert_by_loc`. If the retail is different, update the

item/warehouse table with the new retail and call insert\_tran to insert tran\_data records for the price change.

**Insert\_tran()**

This function is called every time the retail price is changed at a stock location. If the total stock quantity (stock on hand + transfer quantity) is greater than zero then the transaction type (markdown, etc) is determined by the calc\_tran\_type function. The retail information is converted into the primary currency, and then tran\_data (via the STKLEDGR\_SQL.TRAN\_DATA\_INSERT package) records are inserted.

**Calc\_tran\_type()**

The PRCCHGTYP stored procedure is called to determine the tran\_data type code.

**Size\_arrays()**

Sizes the arrays to hold data from the different fetches.

**Free\_arrays()**

Frees the memory allocations of the arrays and global variables.

**I/O Specification**

N/A

**Technical Issues**

N/A

## Price change impact [pcimpc]

### Design Overview

The Price Change Impact (pcimpc) program updates the impact quantity and impact amount fields on the price change header table to take into account the impact of the price change, in both dollars and units, based on the current stock on hand.

| TABLE           | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-----------------|-------|--------|--------|--------|--------|
| PRICE_SUSP_HEAD | Yes   | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: price\_change

### Restart Recovery

N/A

### Program Flow

N/A

### Shared Modules

PCIMPACT\_HEAD – This procedure does the actual calculations used to determine the impact of the price change

### Function Level Description

N/A

### I/O Specification

N/A

### Technical Issues

N/A

## Price change overlap [pcovrl]

### Design Overview

The Price Change Overlap program searches through all submitted price change records and checks for conflicts with other price changes in submitted, approved or extracted status. Conflicts that are found in this program are written to a price change conflicts table, which can be used for reporting purposes. Each time this program is run, the price conflict table is cleared, so that all conflicts on the table are current conflicts.

| TABLE          | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------|-------|--------|--------|--------|--------|
| PRICE_CONFLICT | No    | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: N/A

Pre-Processing: The table price\_conflict is truncated in pre-processing

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

N/A

### Program Flow

N/A

### Shared Modules

PCOVERLAP\_BATCH – This is the procedure that actually does all of the overlap checking.

### Function Level Description

N/A

### I/O Specification

N/A

### Technical Issues

N/A

## Promotion overlap analysis [pcovrlpq]

### Design Overview

When submitting a promotion for approval, the event must be checked to ensure that none of the affected items are on promotion at the affected locations. This process can take an inordinate amount of time when performed online, so promotions needing analysis are queued up for later processing. pcovrlpq (Queued Promotion Overlap Analysis) cycles through submitted promotions, finds any overlaps with other promotions, and writes them to the PRICE\_OVERLAP\_LOG table for reporting.

### Scheduling Constraints

|                     |                                                                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Processing Cycle    | Ad-hoc; can run during the online day if needed.                                                                                              |
| Scheduling Diagram: | This program has no functional dependencies, but it should avoid running concurrently with prmxtpc to reduce the risk of resource contention. |
| Pre-Processing:     | N/A                                                                                                                                           |
| Post-Processing:    | N/A                                                                                                                                           |
| Threading Scheme:   | N/A                                                                                                                                           |

### Restart Recovery

N/A.

### Program Flow

N/A.

### Shared Modules

PRICING\_OVERLAP\_SQL.PROM\_OVERLAP: PL/SQL stored procedure that actually does all the work of finding overlaps for a given promotion.

### Function Level Descriptions

#### main

Logs on to the database, calls process() to handle the work, writes messages to the daily logs.

#### process

Fetches promotions from the PRICE\_OVERLAP\_QUEUE table and calls PR\_overlap() for each one.

#### PR\_overlap

Calls the PRICING\_OVERLAP\_SQL.PROM\_OVERLAP stored procedure to find and log overlaps.

### I/O Specification

N/A.



Technical Issues

N/A.

## Purge price changes [pcprg]

### Design Overview

The Price Change Purge program is responsible for removing old price changes from the system. Price changes are removed from the system using the following criteria:

- the status of the price changes is Canceled or Deleted
- the status is Extracted, but the effective date was yesterday
- the status of the price change is Rejected and the effective date of the price change has met the requirement for the number of days that rejected price changes are held.

Note: The number of days that rejected price changes are held is determined by a system option.

| TABLE                     | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---------------------------|-------|--------|--------|--------|--------|
| PERIOD                    | No    | Yes    | No     | No     | No     |
| PRICE_CONFLICT            | No    | No     | No     | No     | Yes    |
| PRICE_CHANGE_CONFLICT_LOG | No    | No     | No     | No     | Yes    |
| PRICE_EVENT_ZONE          | No    | No     | No     | No     | Yes    |
| PRICE_SUSP_DETAIL         | No    | No     | No     | No     | Yes    |
| PRICE_SUSP_HEAD           | No    | Yes    | No     | No     | Yes    |
| UNIT_OPTIONS              | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

Re-run:

- If this program terminates normally, the price change suspense tables must be recovered.
- If this program terminates abnormally, restart without recovery.

### Program Flow

N/A

### Shared Modules

N/A

### Function Level Description

All of the functionality of this program is performed in the process() function:

This function will select records from price\_susp\_head that should be deleted or purged from the system. This includes all cancelled or deleted records, all rejected records that are older than allowed according to the number\_days\_rejects\_held field on the unit\_options table, and all records that are in extracted status and have an activation date of yesterday.

The selected records are fetched into a loop where the deletes occur. Inside the while loop records are deleted from price\_conflict, price\_even\_zone, price\_susp\_detail, price\_change\_conflict\_log, and price\_susp\_head where the price\_change on those tables is the same as the selected price change.

**I/O Specification**

N/A

**Technical Issues**

N/A

## Price transactions download for store zone changes [pctrandn]

### Design Overview

The Price Transactions Download for Store Zone Changes (pctrandn) module performs the first step in processing pricing transactions that are created when a store is moved from one zone to another. This module will download the price changes generated from store zone change. It will select all items that exist at the store that will be moved and are associated with the zone group in which the move was made. Then, records are written out to the POS interface table, similar to permanent price changes, such that the store(s) affected will be notified.

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Phase 3 - Nightly

Scheduling Diagram: Prior to running Pctranex.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Store

| TABLE                  | INDEX | SELECT | INSERT | UPDATE | DELETE |
|------------------------|-------|--------|--------|--------|--------|
| PERIOD                 |       | X      |        |        |        |
| PRICE_BATCH_TRAN       |       | X      |        |        |        |
| WIN_SKUS               | X     | X      |        |        |        |
| RAG_SKUS               | X     | X      |        |        |        |
| RAG_STYLE              |       | X      |        |        |        |
| WIN_STORE              | X     | X      |        |        |        |
| RAG_SKUS_ST            | X     | X      |        |        |        |
| PACKHEAD               | X     | X      |        |        |        |
| PACKSTORE              | X     | X      |        |        |        |
| POS_MOD                |       |        | X      |        |        |
| ITEM_ZONE_PRICE        | X     | X      |        |        |        |
| PRICE_ZONE_GROUP_STORE | X     | X      |        |        |        |
| SYSTEM_OPTIONS         |       | X      |        |        |        |

### Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

The restart string is new zone group + store + old zone group + item identifier. The restart is split between two cursors, with c\_tran being the main cursor and one of c\_win\_skus, c\_packhead, or c\_rag\_skus being used depending on whether the item being processed is a staple item, a pack item, or a fashion item. The commit counter should be set to 10,000 records.

**Driving Cursor:**

```
EXEC SQL DECLARE c_tran CURSOR FOR
 SELECT pbt.zone_group_id,
 pbt.store,
 pbt.old_zone_id
 FROM v_restart_store rv,
 price_batch_tran pbt
 WHERE rv.driver_value = pbt.store
 AND rv.driver_name = :restart_driver_name
 AND rv.thread_val = :restart_thread_val
 AND rv.num_threads = :restart_num_threads
 AND (pbt.zone_group_id >= NVL(:restart_zone_group, -999) OR
 (pbt.zone_group_id = :restart_zone_group AND
 (pbt.store > :restart_store OR
 (pbt.store = :restart_store AND
 (pbt.old_zone_id >= :restart_zone_id)))))
 AND pbt.effective_date <= L_vdate + L_pos_ext_days
 ORDER BY pbt.zone_group_id, pbt.store, pbt.old_zone_id;
```

**Examples of Sku\_processing cursors:**

```
/* Cursor to retrieve records from win_skus table */
EXEC SQL DECLARE c_win_skus CURSOR FOR
 SELECT w.sku,
 w.dept,
 w.class,
 w.subclass,
 NVL(s.clear_ind, 'N'),
 DECODE(:os_std_av_ind, 'S',
 NVL(s.unit_cost, 0), NVL(s.av_cost, 0)),
 s.stock_on_hand + in_transit_qty,
 ';' || TO_CHAR(:pbt_zone_group) ||
 ';' || TO_CHAR(:old_zone_id) ||
 ';' || TO_CHAR(:store) ||
 ';' || TO_CHAR(w.sku) ||
 ';' || 'S'
 FROM win_skus w,
 win_store s
 WHERE w.zone_group_id = :pbt_zone_group
 AND s.sku = w.sku
 AND s.store = :store
 AND w.sku > :ps_restart_sku
 ORDER BY w.sku;
```

```
/* Cursor to retrieve records from packhead table */
EXEC SQL DECLARE c_packhead CURSOR FOR
 SELECT packhead.pack_no,
```

```

 packhead.dept,
 packhead.class,
 packhead.subclass,
 '||TO_CHAR(:pbt_zone_group)||
 '||TO_CHAR(:store)||
 '||TO_CHAR(:old_zone_id)||
 '||TO_CHAR(packhead.pack_no)||
 '||P'
FROM packhead,
WHERE packhead.zone_group_id = :pbt_zone_group
 AND packhead.pack_no > :restart_sku
ORDER BY packhead.pack_no;

/* Cursor to get fashion SKU records*/
EXEC SQL DECLARE c_rag_skus CURSOR FOR
 SELECT rsku.sku,
 rstyle.dept,
 rstyle.class,
 rstyle.subclass,
 NVL(rstore.clear_ind, 'N'),
 DECODE(:os_std_av_ind, 'S',
 NVL(rstore.unit_cost, 0), NVL(rstore.av_cost, 0)),
 rstore.stock_on_hand + in_transit_qty,
 '||TO_CHAR(:pbt_zone_group)||
 '||TO_CHAR(:old_zone_id)||
 '||TO_CHAR(:store)||
 '||TO_CHAR(rsku.sku)||
 '||f'
 FROM rag_skus rsku,
 rag_style rstyle,
 rag_skus_st rstore
 WHERE rstore.sku = rsku.sku
 AND rstore.store = :store
 AND rsku.style = rstyle.style
 AND rstyle.zone_group_id = :pbt_zone_group
 AND rsku.sku > :ps_restart_sku
 ORDER BY rsku.sku;

```

### Program Flow

Structure Chart

N/A

### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

**POS\_UPDATE\_SQL.POS\_MODS\_INSERT** – This procedure will be called to do the inserts into the POS interface table (pos\_mods).

### Function Level Description

All database interactions required and error handling considerations

init()

In addition to performing standard restart tasks, the current date (vdate) is selected from the period table and the POS\_extract\_days is selected from the system\_options table.

#### **process()**

Price\_batch\_tran is the driving table for the program. Store Zone Changes are selected in this function based on the number of POS extract days and the effective date of the zone change. If today's date (vdate) is within the number of POS extract days from the zone change effective date, then the store information including new and old zone information will be selected in this function. If the zone change is within POS extract days of its effective date then call the sku\_processing function for each item type ('S' staple, 'P' pack, and 'f' fashion). Each call will be made passing in the item type parameter. The sku\_processing function will execute one of three cursors (depending on the item type) to retrieve all of the items in the zone group of the zone change.

#### **create\_pos\_mods()**

This function will be called by the sku\_processing function and will perform inserts into pos\_mods for a price change and also performs the tasks necessary to perform the insert. Sku information for pos\_mods table inserts is fetched from desc\_look. Old and new retail information is compared to determine the type of price change being sent to the point of sales. Finally, records are inserted into pos\_mods for the sku by calling POS\_UPDATE\_SQL.POS\_MODS\_INSERT.

#### **sku\_processing()**

This function will be called from the process function for each item type ('S', 'P', 'f') and for each zone change that must be downloaded to the stores. Information for the sku or pack item is now fetched and processed in this function by executing one of three cursors. One for each of the three item types. If the item exists at the current store and is not on clearance in the old zone then processing continues, otherwise skip to the next sku. The old and new retail information is fetched from item\_zone\_price based on the new and old zone information selected from price\_batch\_tran. If either the single or multi-unit price changes, then the create\_pos\_mods() function is called to insert pos\_mods records for the POS.

#### **final ()**

This function will perform the standard restart/recovery close logic.

#### **I/O Specification**

All files layouts input and output

None.

#### **Technical Issues**

None.

#### **Test Scenarios**

1. Run program and ensure that only price\_batch\_tran records with an effective date that is within the number of POS extract days of today are processed.
2. Only SKU's that meet the following criteria should have records inserted into POS\_MODS:
  - Must belong in the zone group of the zone change.
  - Are stocked at the store changing zones.
  - Item/Store is not on clearance.

- Have a different unit retail in the old zone than in the new zone.
3. POS\_MODS records are inserted for stores changing zones and are for the correct unit retail (new zone) of each item.
  4. No records are deleted or updated from any of the tables in this module.



# Price transactions extract for store zone changes [pctranex]

## Design Overview

The Price Transactions Extract for Store Zone Changes module performs the final steps in processing pricing transactions that are created when a store is moved from one zone to another. This module will extract the price changes generated from store zone change that will become effective tomorrow and update the RMS tables to reflect the new prices. It will select all items that exist at the store that was moved and are associated with the zone group in which the move was made. Then, updates are made to the item/store prices, along with inserts into price history, supplier history, and transaction-level stock ledger tables.

## Scheduling Constraints

### Pre/Post Logic Description

Processing Cycle: Phase 3 - Daily

Scheduling Diagram: After the pctrandn.pc module is run

Pre-Processing: N/A

Post-Processing: Processed records will be deleted from the price\_batch\_tran table.

Threading Scheme: Store

| TABLE                  | INDEX | SELECT | INSERT | UPDATE | DELETE |
|------------------------|-------|--------|--------|--------|--------|
| ITEM_SUPPLIER          | X     | X      |        |        |        |
| PERIOD                 |       | X      |        |        |        |
| PRICE_BATCH_TRAN       |       | X      |        |        |        |
| PRICE_HIST             |       |        | X      |        |        |
| PRICE_ZONE             | X     |        |        |        | X      |
| ITEM_ZONE_PRICE        | X     | X      |        |        | X      |
| PRICE_ZONE_GROUP_STORE | X     | X      |        | X      |        |
| SUP_DATA               |       |        | X      |        |        |
| SYSTEM_OPTIONS         |       | X      |        |        |        |
| ITEM_MASTER            | X     | X      |        |        |        |
| ITEM_LOC               | X     | X      |        | X      |        |

## Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

The restart string is new zone + store + old zone + item identifier. The restart is split between two cursors, with c\_tran being the main cursor and c\_item. The commit counter should be set to 10,000 records.

### **Driving Cursor:**

```

/* Cursor to retrieve records from price_batch_tran table */
EXEC SQL DECLARE c_tran CURSOR FOR
 SELECT pbt.zone_group_id,
 pbt.old_zone_id,
 pbt.store,
 pbt.new_zone_id,
 pbt.new_primary_store
 FROM v_restart_store rv,
 price_batch_tran pbt
 WHERE pbt.effective_date = TO_DATE(:os_tomorrow, 'YYYYMMDD')
 AND rv.driver_value = pbt.store
 AND rv.driver_name = :ps_restart_driver_name
 AND rv.thread_val = :pi_restart_thread_val
 AND rv.num_threads = :pi_restart_num_threads
 AND (pbt.zone_group_id > NVL(:ps_restart_zone_group, -
999)
 OR (pbt.zone_group_id = :ps_restart_zone_group
 AND (pbt.old_zone_id > :ps_restart_zone_id
 OR (pbt.old_zone_id = :ps_restart_zone_id
 AND (pbt.store >=
:ps_restart_store))))))
 ORDER BY pbt.zone_group_id, pbt.old_zone_id, pbt.store;

```

### **Program Flow**

Structure Chart

N/A

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

**PRCCHGTYP** – Calculates the tran\_type for the tran\_data insert.

**STKLEDGR\_SQL.TRAN\_DATA\_INSERT** – Inserts a record into tran\_data.

**SUPP\_ITEM\_SQL.GET\_PRI\_SUP\_COST** – Gets the primary supplier cost for an item.

**ITEMLOC\_QTY\_SQL.GET\_TRANSFER** – Determines stock amounts in transit (for tran\_data insert).

### **Function Level Description**

All database interactions required and error handling considerations

**item\_processing()**

This function is called from the process function. Information for the item is now fetched and processed in this function. If the item exists at the current store, processing continues. If it does not, skip to the next item. If the fetched item is a pack and it is not orderable, set its unit\_cost to 0, otherwise call SUPP\_ITEM\_SQL.GET\_PRI\_SUP\_COST to get cost info for the pack. The old and new retail information, including the new selling unit\_retail and uom information, is fetched from item\_zone\_price based on the new and old zone information selected from the price\_batch\_tran table. If the single unit retail price changes then the item\_loc retail information is updated (including the new selling\_unit\_retail and uom fields) to hold the new values. If either the single or multi-unit price changes and the stock\_qty is greater than zero (which will not occur for packs), then insert\_tran() is called to insert records into tran\_data, and create\_price\_hist is called to insert price\_hist records. If neither retails change then the item should be skipped, since it is not changing in price at the store when the store moves zones.

#### **calc\_tran\_type()**

This function is called from the item\_processing function and calls the stored procedure prchgttype to determine the tran\_type to be inserted into the tran\_data table.

#### **insert\_tran()**

First this function calls calc\_tran\_type() to determine the type of tran\_data insert. The store's stock on hand and the quantity in transit must be added to determine the stock\_qty, so the ITEMLOC\_QTY\_SQL.GET\_TRANSFER stored procedure is called. If the stock\_qty is positive the STKLEDGR\_SQL.TRAN\_DATA\_INSERT package is called to insert the tran\_data record. If the price change is a markup or markdown cancellation then supplier information is inserted into sup\_data with a transaction type of 10 (markdown at retail). Because the transaction type is a markdown the markup amount should be multiplied by -1 to get the appropriate sign for the insert (to effectively create a markup). A similar process occurs for the second set of tran\_types and tran\_type amounts (for multi\_unit information) to insert tran\_data and sup\_data records.

#### **init()**

In addition to performing standard restart tasks, the current date is selected from the period table, and the std\_av\_ind is selected from the system\_options table to be used in selecting the correct unit\_cost in the item\_processing function for insert into the price\_hist table in the create\_price\_hist function.

#### **process()**

Price\_batch\_tran is the driving table for the program. Store Zone Changes are selected in this function based on the effective date of the zone change. If today's date (vdate) is the day before the effective date, then the store information including new and old zone information will be selected in this function. If the zone change effective date is tomorrow then call the update\_new\_zone function to actually move the store to the new zone and the delete\_old\_zone function is called, which will delete the old zone information. Next, call the item\_processing function for each zone change that will be effective tomorrow. Each call will be made passing in the item type parameter. The item\_processing function will execute one of three cursors (depending on the item type) to retrieve all of the items in the zone group of the zone change.

#### **create\_price\_hist()**

This function is called from the item\_processing function and inserts a record into price\_hist. A different transaction type will be inserted depending on the type of price that changed: 4 for single unit price changes only, 10 for multi-unit price changes only, and 11 for single and multi-unit price changes.

**delete\_old\_zone()**

This function is called from the process function and will delete rows from zone tables where stores no longer exist in a zone\_id. First the price\_zone\_group\_store table is queried for all the stores in the old zone\_id. If there are no records then rows with the old zone id/zone group are deleted from the appropriate zone tables: item\_zone\_price and price\_zone.

**update\_new\_zone()**

This function is also called from the process function and will update the price\_zone\_group\_store table for each store changing zones, setting the zone id equal to the new zone id on the price\_batch\_tran table.

**final ()**

This function will perform the standard restart/recovery close logic.

**I/O Specification**

All files layouts input and output

None.

**Technical Issues**

None.

**Test Scenarios**

- 1 Run program and ensure that only price\_batch\_tran records with an effective date that is equal to tomorrow are processed.
- 2 Only ITEM's that meet the following criteria should have been updated:
  - Must belong in the zone group of the zone change.
  - Are stocked at the store changing zones.
  - Item/Store is not on clearance.
  - Have a different unit retail in the old zone than in the new zone.
- 3 Price\_hist records are inserted for the price changes for approved items generated as a result of the store zone change.
- 4 Tran\_data records are inserted for each approved item changing price at the store changing zones.
- 5 Sup\_data records are inserted for each approved items at the dept/supplier affected by the price\_changes.
- 6 Price\_zone\_group\_store records are updated for stores changing zones, setting the zone id equal to the new zone id from the price\_batch\_tran table.

## Class level planned sales units acceptance [plncupld]

### Design Overview

The purpose of this batch module is to accept new planned sales units at a class/store/end-of-week level from an outside planning system. This data is then inserted into or used to update the planned sales data on the CLASS\_SALES\_HIST table.

Detail processing is very limited. If there is not a row on the table to update, a new one is inserted with department, class, store, end-of-week date, and planned sales amount taken directly from the input file. The date procedures CAL\_TO\_454 and CAL\_TO\_454\_HALF are used to calculate the half, month and week numbers to be inserted with that row. If the table is being updated, the old planned sales amount is overwritten with the planned sales amount extracted from the input file. Records with an end-of-week date before the current week's end will be rejected since only current and future plan numbers will be accepted into Retek.

### Scheduling Constraints

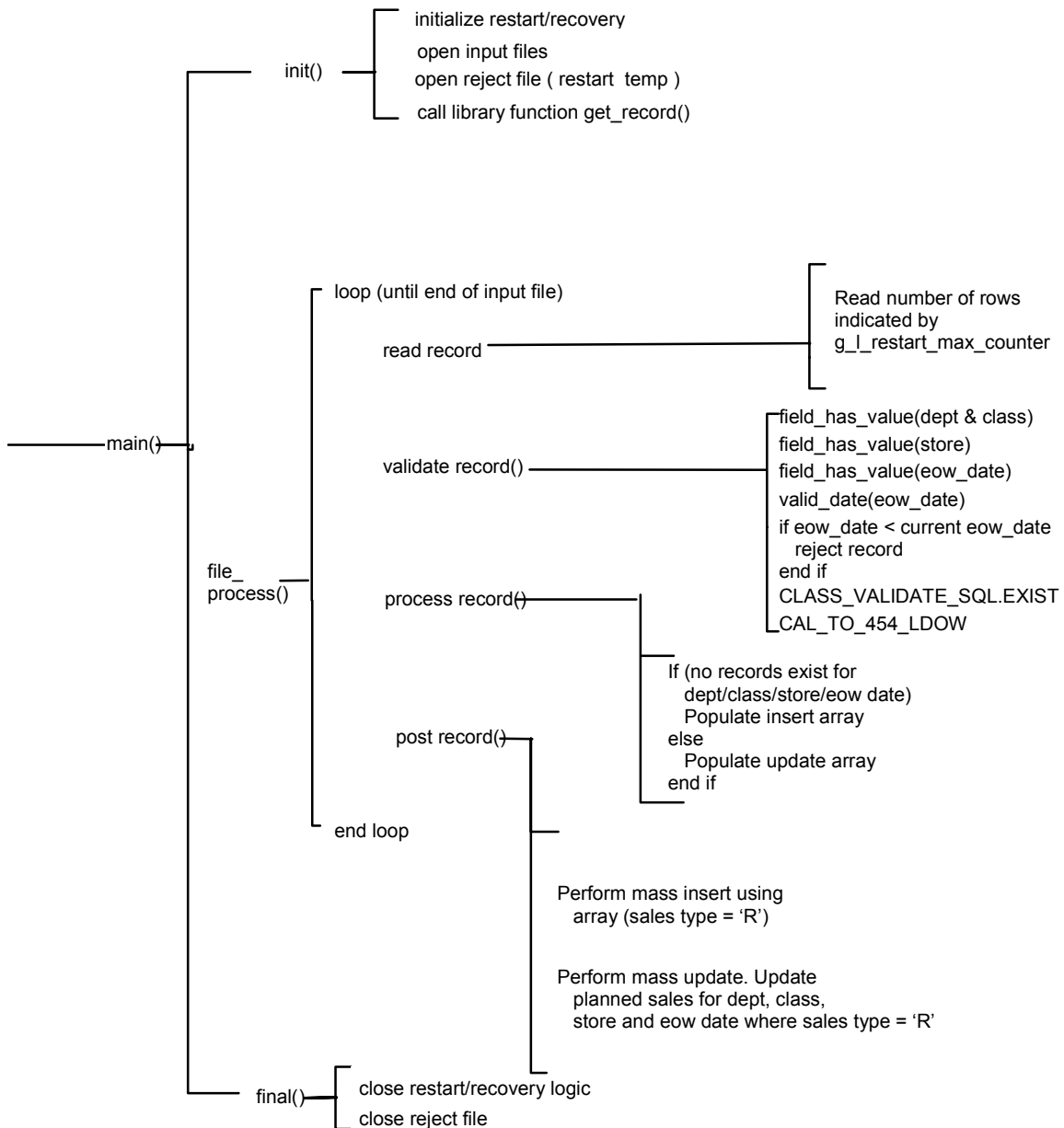
|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| Processing Cycle:   | At user discretion.                                                              |
| Scheduling Diagram: | This program is independent of all other batch processing and can be run ad hoc. |
| Pre-Processing:     | N/A                                                                              |
| Post-Processing:    | N/A                                                                              |
| Threading Scheme:   | N/A                                                                              |

### Restart Recovery

The logical unit of work for the planned sales import module is the discrete planned sales transaction. If a fatal error forces a restart, the restart will occur from the point in the file where the last commit occurred.

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE\_ERROR library function.

The recommended commit counter setting is 10,000. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

**Program Flow****Shared Modules**

**CLASS\_VALIDATE\_SQL.EXIST:** This packaged function will validate that a department/class combination exists on the class table.

**CAL\_TO\_454\_LDOW:** This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

**Function Level Description**

`init()`:

- Open files for input and output.
- Initialize restart/recovery.

file\_process():

- Read a record from the input file.
- Determine if the record is valid by calling the validate\_record() function.
- If the record is valid, call the process\_record() function to populate an insert or update array.
- Save the records to the CLASS\_SALES\_HIST table by calling the post\_record() function.

validate\_record():

- Ensure that the department, class, store and end-of-week date fields all contain values by calling the function field\_has\_value() for each.
- Check that the input record's end-of-week date is a valid date in the correct format by calling the valid\_date() function. If so, check that it is also a valid end-of-week date using CAL\_TO\_454\_LDOW.
- Check that the input record's end-of-week date is greater than or equal to the current week's end date. Historic planned sales will not be interfaced into Retek. If the end-of-week date is before the current week's end, reject the record.
- Call the CLASS\_VALIDATE\_SQL.EXIST function to validate that the department/class combination exists on the CLASS table.

process\_record():

- If no row exists on the CLASS\_SALES\_HIST table for the input record's department, class, store and end-of-week date, populate an array to insert a new one. Otherwise, populate an array to update the existing row.

post\_record():

- Perform a mass update and/or insert using the two array structures. For the mass insert, the sales type should always be inserted with an 'R' (for regular sales). The mass update should update the planned sales amount for the input record's department, class, subclass, store and end-of-week date where the sales type is equal to 'R'.

final():

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Exit the program.

on NON-FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Increment g\_l\_rej\_cnt.

## I/O Specification

Input File:

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

If multi-threading will be used, the file name must contain a suffix (e.g. *filename####*) to identify the thread value so that multiple input files can be used concurrently.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDDHHMISS' in order to validate it using `valid_date()`.

A sample detail record would look like the following:

FDETL12345678901234567890123412341234123456789119980101120000123456789012

| Record Name | Field Name                     | Field Type | Default Value                | Description                                                                             |
|-------------|--------------------------------|------------|------------------------------|-----------------------------------------------------------------------------------------|
| File Header | File Type Record Descriptor    | Char(5)    | FHEAD                        | Identifies file record type                                                             |
|             | File Line Identifier           | Char(10)   | Line number in file          | Unique line ID                                                                          |
|             | File Type Definition           | Char(4)    | PLCI                         | Identifies file as 'Planned Class Sales Import'                                         |
|             | File Create Date               | Char(14)   | create date                  | Date file was written by external system (formatted as 'YYYYMMDDHHMISS')                |
| File Detail | File Type Record Descriptor    | Char(5)    | FDETL                        | Identifies file record type                                                             |
|             | File Line Identifier           | Char(10)   | Line number in file          | Unique line ID                                                                          |
|             | Transaction Set Control Number | Char(14)   | specified by external system | Unique ID                                                                               |
|             | Department Identifier          | Char(4)    | numeric dept identifier      | The internal Retek id number of a department - numeric character                        |
|             | Class Identifier               | Char(4)    | numeric class identifier     | The internal Retek id number of a class within the department given - numeric character |
|             | Store Identifier               | Char(10)   | numeric store identifier     | The internal Retek id number of a store - numeric character                             |
|             | Eow Date                       | Char(14)   | end of week date             | The end of week date for the budgeted week (formatted as 'YYYYMMDDHHMISS')              |
|             | Planned Sales                  | Number(12) | Planned Sales                | Planned sales units for the                                                             |



|              |                                      |                            |                        |                                                                           |
|--------------|--------------------------------------|----------------------------|------------------------|---------------------------------------------------------------------------|
|              | Units                                | )<br>4 implied<br>decimals | Units                  | specified<br>dept/class/store/week                                        |
| File Trailer | File Type<br>Record<br>Descriptor    | Char(5)                    | FTAIL                  | Identifies file record type                                               |
|              | File Line<br>Identifier              | Char(10)                   | Line number in<br>file | Unique line ID                                                            |
|              | Control<br>Number File<br>Line Count | Number(10<br>)             | total detail<br>lines  | Sum of all transaction<br>lines, not including file<br>header and trailer |

**Reject File:**

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

**Error File:**

Standard Retek batch error handling 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**

This program will read from an input flat file. It will use restart/recovery, array processing and will commit to the database at a frequency determined by the commit\_max\_ctr on the RETEK\_CONTROL table.

## Department level planned sales units acceptance [plndupld]

### Design Overview

The purpose of this batch module is to accept new planned sales units at a department/store/end-of-week level from an outside planning system. This data is then inserted into or used to update the planned sales data on the DEPT\_SALES\_HIST table.

Detail processing is very limited. If there is not currently a row on the table to update, a new row is inserted with department, store, end-of-week date and planned sales amount taken from the input file. The date procedures CAL\_TO\_454 and CAL\_TO\_454\_HALF are used to calculate the half, month and week numbers to be inserted with that row. If the table is being updated, the old planned sales amount is overwritten with the planned sales amount extracted from the input file. Records with an end-of-week date before the current week's end will be rejected since only current and future plan numbers will be accepted into Retek.

| TABLE               | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---------------------|-------|--------|--------|--------|--------|
| DEPT_SALES_HI<br>ST | No    | No     | Yes    | Yes    | No     |
| DEPT                | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program is independent of all other batch processing and can be run ad hoc.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

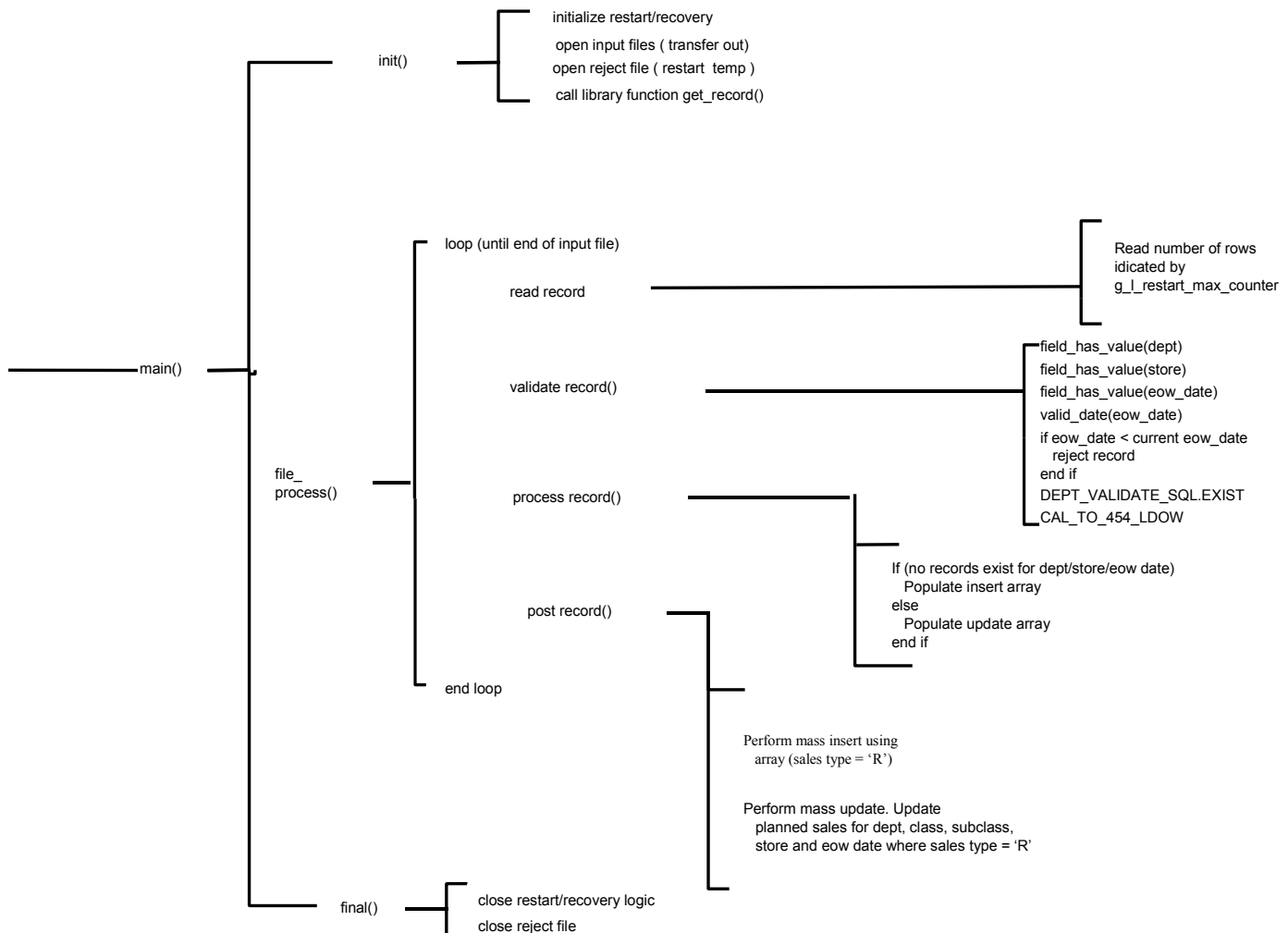
### Restart Recovery

The logical unit of work for the planned sales import module is the discrete planned sales transaction. If a fatal error forces a restart, the restart will occur from the point in the file where the last commit occurred

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE\_ERROR library function.

The recommended commit counter setting is 10,000. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

## Program Flow



## Shared Modules

`DEPT_VALIDATE_SQL.EXIST`: This packaged function will validate that a department exists on the DEPS table.

`CAL_TO_454_LDOW`: This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

## Function Level Description

`init()`:

- Open files for input and output.
- Initialize restart/recovery.

`file_process()`:

- Read a record from the input file.
- Determine if the record is valid by calling the `validate_record()` function.

- If the record is valid, call the `process_record()` function to populate an insert or update array.
- Save the records to the `DEPT_SALES_HIST` table by calling the `post_record()` function.

`validate_record()`:

- Ensure that the department, store and end-of-week date fields all contain values by calling `field_has_value()` function for each.
- Check that the input record's end-of-week date is a valid date in the correct format by calling the `valid_date()` function. If so, check that it is also a valid end-of-week date using `CAL_TO_454_LDOW`.
- Check that the input record's end-of-week date is greater than or equal to the current week's end date. Historic planned sales will not be interfaced into Retek. If the input record's end-of-week date is before the current week's end, reject the record.
- Call the `DEPT_VALIDATE_SQL.EXIST` function to validate that the record's department exists on the department table.

`process_record()`:

- If no record exists on the `DEPT_SALES_HIST` table for the input record's department, store and end-of-week date, populate an array to insert a new one. Otherwise, populate an array to update the existing row.

`post_record()`:

- Perform a mass update and/or insert using the two array structures. For the mass insert, the sales type should always be inserted with an 'R' (for regular sales). The mass update should update the planned sales amount for the input record's department, store and end-of-week date where the sales type is equal to 'R'.

`final()`:

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Exit the program.

on NON-FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Increment `g_l_rej_cnt`.

### **I/O Specification**

Input File:

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

If multi-threading will be used, the file name must contain a suffix (i.e. *filename###*) to identify the thread value so that multiple input files can be used concurrently.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDDHHMISS' in order to validate it using `valid_date()`.

A sample detail record would look like the following:

FDETL1234567890123456789012341234123456789019980101050000123456789012

| Record Name  | Field Name                         | Field Type                    | Default Value                | Description                                                                |
|--------------|------------------------------------|-------------------------------|------------------------------|----------------------------------------------------------------------------|
| File Header  | File Type Record Descriptor        | Char(5)                       | FHEAD                        | Identifies file record type                                                |
|              | File Line Identifier               | Char(10)                      | Line number in file          | Unique line ID                                                             |
|              | File Type Definition               | Char(4)                       | PLDI                         | Identifies file as 'Planned Department Sales Import'                       |
|              | File Create Date                   | Char(14)                      | create date                  | Date file was written by external system (formatted as 'YYYYMMDDHHMISS')   |
| File Detail  | File Type Record Descriptor        | Char(5)                       | FDETL                        | Identifies file record type                                                |
|              | File Line Identifier               | Char(10)                      | Line number in file          | Unique line ID                                                             |
|              | Transaction Set Control Identifier | Char(14)                      | specified by external system | Unique ID                                                                  |
|              | Department Identifier              | Char(4)                       | numeric dept identifier      | The internal Retek id number of a department - numeric character           |
|              | Store Identifier                   | Char(10)                      | numeric store identifier     | The internal Retek id number of a store - numeric character                |
|              | Eow Date                           | Char(14)                      | end of week date             | The end of week date for the budgeted week (formatted as 'YYYYMMDDHHMISS') |
|              | Planned Sales Units                | Number(12) 4 implied decimals | Planned Sales Units          | Planned sales units for the specified dept/store/week                      |
| File Trailer | File Type Record Descriptor        | Char(5)                       | FTAIL                        | Identifies file record type                                                |
|              | File Line Identifier               | Char(10)                      | Line number in file          | Unique line ID                                                             |
|              | Control Number File Line Count     | Number(10)                    | total detail lines           | sum of all transaction lines, not including file header and trailer        |

#### Reject File:

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only

of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

**Error File:**

Standard Retek batch error handling 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**

This program will read from an input flat file. It will use restart/recovery, array processing and will commit to the database at a frequency determined by the `commit_max_ctr` on the `RETEK_CONTROL` table.

## Subclass level planned sales units acceptance [plnsupld]

### Design Overview

The purpose of this batch module is to accept new planned sales units at a subclass/store/end-of-week level from an outside planning system. This data is then inserted into or used to update the planned sales data on the SUBCLASS\_SALES\_HIST table.

Detail processing is very limited. If there is not currently a row on the table to update, a new row is inserted with the department, class, subclass, store, end-of-week date, and planned sales amount taken directly from the file. The date procedures CAL\_TO\_454 and CAL\_TO\_454\_HALF are used to calculate the half, month and week numbers to be inserted with that row. If the table is being updated, the old planned sales amount is overwritten with the planned sales amount extracted from the import file. Records with an end-of-week date before the current week's end will be rejected since only current and future plan numbers will be accepted into Retek.

| TABLE               | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---------------------|-------|--------|--------|--------|--------|
| SUBCLASS_SALES_HIST | No    | No     | Yes    | Yes    | No     |
| SUBCLASS            | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program is independent of all other batch processing and can be run ad hoc.

Pre-Processing: N/A

Post-Processing: N/A

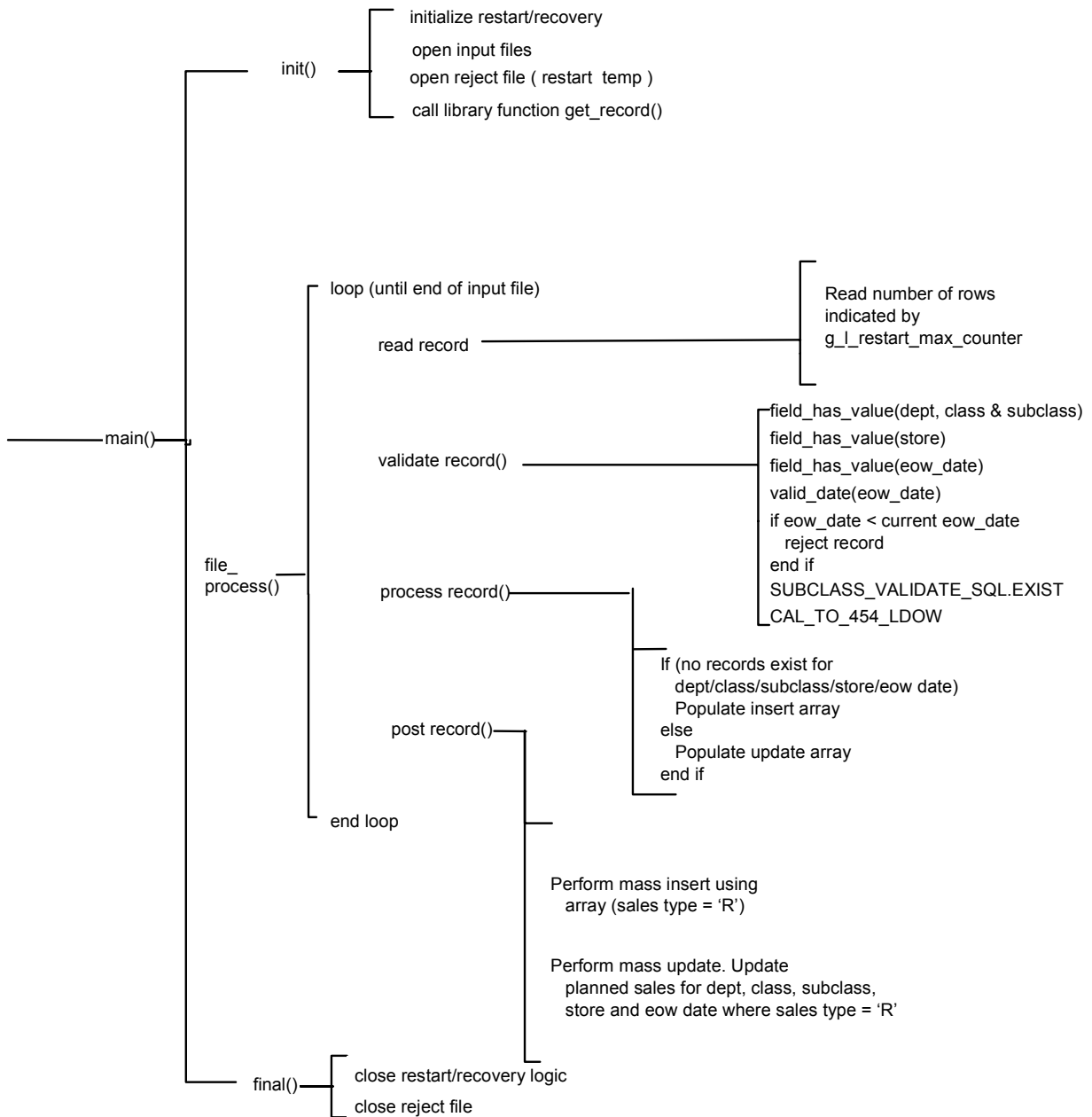
Threading Scheme: N/A

### Restart Recovery

The logical unit of work for the planned sales import module is the discrete planned sales transaction. If a fatal error forces a restart, the restart will occur from the point in the file where the last commit occurred

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE\_ERROR library function.

The recommended commit counter setting is 10,000. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

**Program Flow****Shared Modules**

**SUBCLASS\_VALIDATE\_SQL.EXIST:** This packaged function will validate that a department/class/subclass combination exists on the subclass table.

**CAL\_TO\_454\_LDOW:** This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

**Function Level Description**

**init():**



- Open files for input and output.
- Initialize restart/recovery.

file\_process():

- Read a record from the input file.
- Determine if the record is valid by calling the validate\_record() function.
- If the record is valid, call the process\_record() function to populate an insert or update array.
- Save the records to the SUBCLASS\_SALES\_HIST table by calling the post\_record() function.

validate\_record():

- Ensure that the department, class, subclass, store and end-of-week date fields all contain values by calling the function field\_has\_value() for each.
- Check that the input record's end-of-week date is a valid date in the correct format by calling the valid\_date() function. If so, check that it is also a valid end-of-week date by using CAL\_TO\_454\_LDOW.
- Check that the input record's end-of-week date is greater than or equal to the current week's end date. Historic planned sales will not be interfaced into Retek. If the input record's end-of-week date is less than the current week's end, reject the record.
- Call the SUBCLASS\_VALIDATE\_SQL.EXIST function to validate that the input record's department/class/subclass combination exists on the SUBCLASS table.

process\_record():

- If no row exists on the SUBCLASS\_SALES\_HIST table for the input record's department, class, subclass, store and end-of-week date, populate an array to insert a new one. Otherwise, populate an array to update the existing row.

post\_record():

- Perform a mass update and/or insert using the two array structures. For the mass insert, the sales type should always be inserted with an 'R' (for regular sales). The mass update should update the planned sales amount for the input record's department, class, subclass, store and end-of-week date where the sales type is equal to 'R'.

final():

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Exit the program.

on NON-FATAL ERROR

- Use standard batch error handling to write to the error log file.
- Increment g\_l\_rej\_cnt.

## I/O Specification

Input File:

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

If multi-threading will be used, the file name must contain a suffix (i.e. *filename###*) to identify the thread value so that multiple input files can be used concurrently.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDDHHMISS' in order to validate it using `valid_date()`.

A sample detail record would look like the following:

FDETL123456789012345678901234123412341234123456789119980101500000123456789012

| Record Name | Field Name                     | Field Type | Default Value                | Description                                                                             |
|-------------|--------------------------------|------------|------------------------------|-----------------------------------------------------------------------------------------|
| File Header | File Type Record Descriptor    | Char(5)    | FHEAD                        | Identifies file record type                                                             |
|             | File Line Identifier           | Char(10)   | Line number in file          | Unique line ID                                                                          |
|             | File Type Definition           | Char(4)    | PLSI                         | Identifies file as 'Planned Subclass Sales Import'                                      |
|             | File Create Date               | Char(14)   | create date                  | Date file was written by external system (formatted as 'YYYYMMDDHHMISS')                |
| File Detail | File Type Record Descriptor    | Char(5)    | FDETL                        | Identifies file record type                                                             |
|             | File Line Identifier           | Char(10)   | Line number in file          | Unique line ID                                                                          |
|             | Transaction Set Control Number | Char(14)   | specified by external system | Unique ID                                                                               |
|             | Department Identifier          | Char(4)    | numeric dept identifier      | The internal Retek id number of a department - numeric character                        |
|             | Class Identifier               | Char(4)    | numeric class identifier     | The internal Retek id number of a class within the department given - numeric character |
|             | Subclass Identifier            | Char(4)    | numeric subclass identifier  | The internal Retek id number of a subclass within the class given - numeric character   |
|             | Store Identifier               | Char(10)   | numeric store identifier     | The internal Retek id number of a store - numeric character                             |
|             | Eow Date                       | Char(14)   | end of week date             | The end of week date for the budgeted week (formatted as                                |

|              |                                |                                  |                     |                                                                      |
|--------------|--------------------------------|----------------------------------|---------------------|----------------------------------------------------------------------|
|              |                                |                                  |                     | 'YYYYMMDDHHMISS')                                                    |
|              | Planned Sales Units            | Number(12)<br>4 implied decimals | Planned Sales Units | Planned sales units for the specified dept/class/subclass/store/week |
| File Trailer | File Type Record Descriptor    | Char(5)                          | FTAIL               | Identifies file record type                                          |
|              | File Line Identifier           | Char(10)                         | Line number in file | Unique line ID                                                       |
|              | Control Number File Line Count | Number(10)                       | total detail lines  | Sum of all transaction lines, not including file header and trailer  |

**Reject File:**

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

**Error File:**

Standard Retek batch error handling 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**

This program will read from an input flat file. It will use restart/recovery, array processing and will commit to the database at a frequency determined by the commit\_max\_ctr on the RETEK\_CONTROL table.

## POS configuration download [poscdnld]

### Design Overview

This program will handle the download of all Retek 'POS configurations' from the RMS database to a flat file for loading into a POS/Back Office application. The program will utilize a series of table statuses to identify delta records to send down on a nightly basis, thus keeping interface volume down.

Currently there are 7 identified POS configurations that will be handled by this program. They are: Coupons, Product Restrictions, Money Orders, Tender Types, Touch Buttons, Paid In-Out, and Supplier Payment Options at the Store. Of these 7 POS configurations, 5 contain data at the store level only, while 2 contain data at the item/store level. Furthermore, for the 2 POS configurations that are item/store specific, the items are associated with the POS configurations based on user-entered merchandise criteria. For example, a coupon configuration may apply to all items in departments 100, 200, and 300. The system does not store each coupon/item relationship, but rather the merchandise criteria (dept 100, dept 200, dept 300).

In all cases except one, delta records will be sent down nightly for POS configurations and only records whose effective dates are the next day will be sent down. Therefore, if a store is added to a POS configuration, only the new store would receive the new configuration and not all stores associated with it. The one exception is changes to the merchandise criteria. Once a merchandise configuration is modified, all items that meet the new criteria will be sent down (not the deltas). The overlay of items will be sent to the POS because RMS will not save the items that belong to the criteria, only the criteria itself. By saving criteria instead of criteria and all the items that meet the criteria, there will be a substantial savings in disk space.

In addition to being responsible for downloading data to the stores, this program will handle the delete logic for the POS configurations. If a POS configuration, store (on pos\_store), or merchandise criteria is set to delete status online, this program will send the delete notification to the store and purge the record(s) from the RMS database.

A post process will be run in the prepost.pc batch module to reset the status fields on money\_ord\_head, pos\_coupon\_head, pos\_prod\_rest\_head, pos\_sup\_pay\_criteria, pos\_button\_head, pos\_tender\_type\_head, pos\_payinout\_head, pos\_merch\_criteria, and pos\_store to NULL and set the extract\_req\_ind = 'N'.

### Scheduling Constraints

#### Pre/Post Logic Description

Processing Cycle: 4

Scheduling: This program should be run directly after posdnld.pc.

Pre-Processing: None

Post-Processing: Run prepost.pc to set statuses back to NULL.

Threading Scheme: None

### Restart Recovery

Logical Unit of Work ( recommended Commit check points )

Driving Cursor

### Logical unit of work

Pos\_config\_id and Pos\_config\_type

### Driving Cursor:

```

SELECT payinout_id,
 'PYIO',
 pos_config_status,
 ROWIDTOCHAR(rowid),
 ';' || TO_CHAR(payinout_id) || ';' || 'PYIO'
FROM pos_payinout_head
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'PYIO') < 'PYIO')
 OR (NVL(:ps_restart_pos_config_type,'PYIO') = 'PYIO'
 AND payinout_id > TO_NUMBER(:ps_restart_pos_config_id)))
UNION ALL
SELECT tender_type_id,
 'TTYP',
 pos_config_status,
 ROWIDTOCHAR(rowid),
 ';' || TO_CHAR(tender_type_id) || ';' || 'TTYP'
FROM pos_tender_type_head
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'TTYP') < 'TTYP')
 OR (NVL(:ps_restart_pos_config_type,'TTYP') = 'TTYP'
 AND tender_type_id > TO_NUMBER(:ps_restart_pos_config_id)))
UNION ALL
SELECT button_config_id,
 'BTTN',
 pos_config_status,
 ROWIDTOCHAR(rowid),
 ';' || TO_CHAR(button_config_id) || ';' || 'BTTN'
FROM pos_button_head
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'BTTN') < 'BTTN')
 OR (NVL(:ps_restart_pos_config_type,'BTTN') = 'BTTN'
 AND button_config_id > TO_NUMBER(:ps_restart_pos_config_id)))
UNION ALL
SELECT sup_pay_id,
 'SPAY',
 pos_config_status,
 ROWIDTOCHAR(rowid),
 ';' || TO_CHAR(sup_pay_id) || ';' || 'SPAY'
FROM pos_sup_pay_criteria
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'SPAY') < 'SPAY')
 OR (NVL(:ps_restart_pos_config_type,'SPAY') = 'SPAY'
 AND sup_pay_id > TO_NUMBER(:ps_restart_pos_config_id)))
UNION ALL
SELECT pos_prod_rest_id,
 'PRES',
 pos_config_status,
 ROWIDTOCHAR(rowid),

```

```

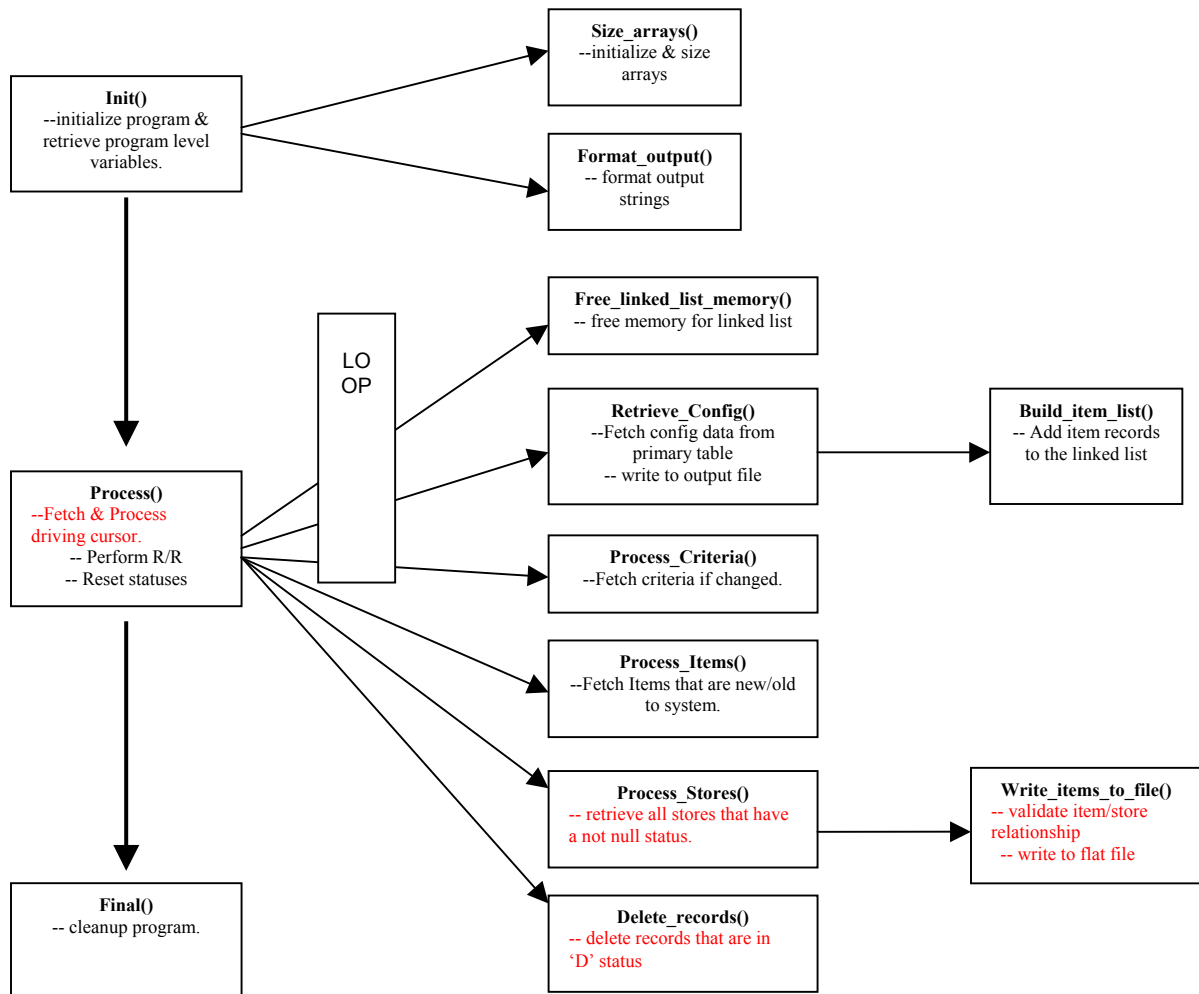
 '||TO_CHAR(pos_prod_rest_id)||';'||'PRES'
FROM pos_prod_rest_head
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'PRES') < 'PRES')
 OR (NVL(:ps_restart_pos_config_type,'PRES') = 'PRES'
 AND pos_prod_rest_id > TO_NUMBER(:ps_restart_pos_config_id)))
UNION ALL
SELECT coupon_id,
 'COUP',
 pos_config_status,
 ROWIDTOCHAR(rowid),
 '||TO_CHAR(coupon_id)||';'||'COUP'
FROM pos_coupon_head
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'COUP') < 'COUP')
 OR (NVL(:ps_restart_pos_config_type,'COUP') = 'COUP'
 AND coupon_id > TO_NUMBER(:ps_restart_pos_config_id)))
UNION ALL
SELECT money_ord_id,
 'MORD',
 pos_config_status,
 ROWIDTOCHAR(rowid),
 '||TO_CHAR(money_ord_id)||';'||'MORD'
FROM pos_money_ord_head
WHERE extract_req_ind = 'Y'
 AND effective_date <= TO_DATE(:ps_tomorrow,'YYYYMMDD')
 AND ((NVL(:ps_restart_pos_config_type,'MORD') < 'MORD')
 OR (NVL(:ps_restart_pos_config_type,'MORD') = 'MORD'
 AND money_ord_id > TO_NUMBER(:ps_restart_pos_config_id)))
ORDER BY 2,1;

```

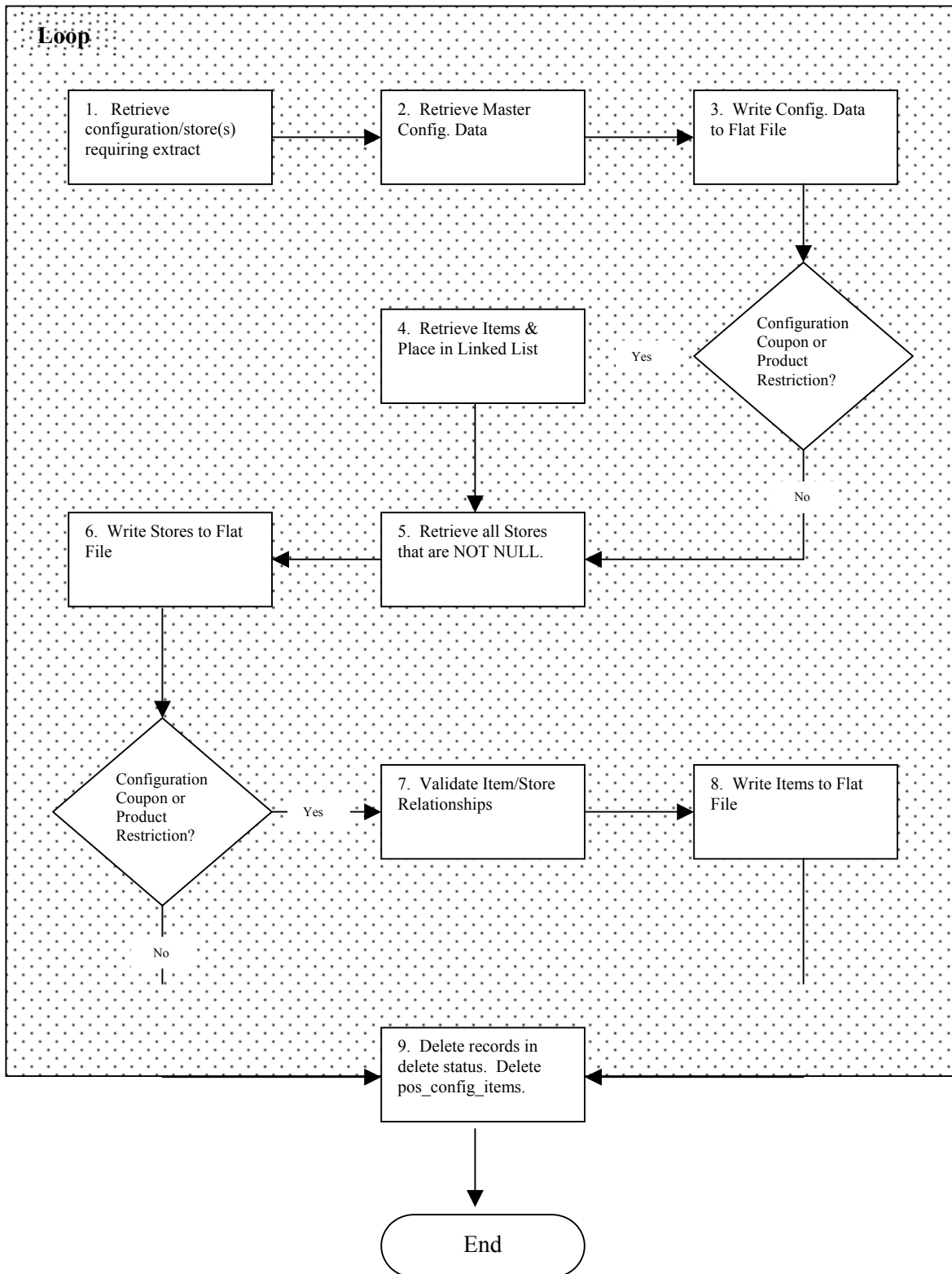
## Program Flow

### Structure Chart

### Function Topology



## Process Topology





## Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

ITEMLOC\_ATTRIB\_SQL.ITEM\_LOC\_EXIST – validates if an item/store relationship exists in RMS.

## Function Level Description

All database interactions required and error handling considerations

### Main()

Perform standard main() function processing (logon, call init(), process(), and final() functions).

### Init()

Perform standard Retek restart/recovery initialization (i.e. calling restart\_file\_init, initializing restart multithread variables, etc.). Select the vdate and place it in a variable called ps\_vdate. Also figure vdate +1 and create a variable for that value and call it ps\_tomorrow. This variable will be used in the driving cursor to select records whose effective day is the next day. Create a final and temporary (use the tmpname C function) flat file for output. Call the size\_arrays() function to initialize the various arrays used within the program. Then, call the format\_output() function to format the output strings for each record type in the flat file. Finally, write the FHEAD record to the output file.

### Size\_arrays()

Size the POS configuration, item fetch, and store arrays to the size of the restart\_control.commit\_max\_ctr. If memory allocation fails, abort the program.

### Format\_output()

Format the output strings for each record type (see I/O Specification section for detail) written to the output file.

### Process()

Within a loop, fetch the driving cursor into the POS configuration array. Invoke a for loop to process the array. Within the loop, call the retrieve\_config() function to retrieve fields relative to the POS configuration type being processed. For example, if the POS config type is 'COUP' (coupons), go retrieve the necessary information from the coupon\_head table. *NOTE: See the flat file to get an idea of what data needs to be selected off of each header table. Remember to select by rowid.*

Next, if the POS configuration type is coupon, product restriction, or button, call free\_linked\_list\_memory(), process\_criteria() and process\_items() functions. These functions will populate an item array with all items that are associated with the current POS configuration. These item records will not be written to a flat file yet, but stored in a linked list to be downloaded later with the stores.

The `process_stores()` function will retrieve all stores tied to the POS configuration and place them in a store array. It will then loop through the store array, writing store records to the output file. In the case where the POS configuration type is coupons or product restrictions, it will also loop through items and write them to the flat file.

After processing the POS configuration and writing it to a flat file, call the `delete_records()` function to delete any records in 'D' status and to purge the `pos_config_items` table. Then, call the `restart_commit()` and `restart_file_write()` function will be called to commit the data and update the restart/recovery tables.

### **Retrieve\_config()**

Based on the current `pos_config_type` and `pos_config_id` retrieve all information from the appropriate master table and write it to the flat file by calling the appropriate retrieve function based on the `pos_config_type`. If the `pos_config_type` is Payinout call the `retrieve_pyio_config()`. If the `pos_config_type` is Tender Type call the `retrieve_ttyp_config()`. If the `pos_config_type` is Buttons call the `retrieve_btn_config()`. If the `pos_config_type` is Supplier Payment Type call the `retrieve_spay_config()`. If the `pos_config_type` is Product Restriction call the `retrieve_pres_config()`. If the `pos_config_type` is Coupon call the `retrieve_coup_config()`. If the `pos_config_type` is Money Order call the `retrieve_mord_config()`.

#### **Retrieve\_pyio\_config()**

Retrieve all Payinout information from `pos_payinout_head` and write it to the flat file according to Retek standards.

#### **Retrieve\_ttyp\_config()**

Retrieve all Tender Type information from `pos_tender_type_head` and write it to the flat file according to Retek standards.

#### **Retrieve\_btn\_config()**

Retrieve all Button information from `pos_button_head` and `pos_button_detail` and write it to the flat file according to Retek standards.

#### **Retrieve\_spay\_config()**

Retrieve all Supplier Payment information from `pos_sup_pay_criteria` and write it to the flat file according to Retek standards.

#### **Retrieve\_pres\_config()**

Retrieve all Product Restriction information from `pos_prod_rest_head` and `pos_day_time_date` and write it to the flat file according to Retek standards.

#### **Retrieve\_coup\_config()**

Retrieve all Tender Type information from pos\_coupon\_head and write it to the flat file according to Retek standards.

Retrieve\_mord\_config()

Retrieve all Money Order information from pos\_money\_ord\_head and write it to the flat file according to Retek standards.

### Process\_criteria

Fetch any changed merchandise criteria associated with the current POS configuration into an array. Since the number of records returned is variable and the information will not be processed until later (after the stores are selected), it will be loaded into a linked list. After the fetch, loop through the array and call the build\_item\_list() function to allocate memory and enter the item record into a linked list. Load the status field with the value of 'C' (changed).

The syntax of the select statement should be similar to the following:

```
select distinct im.item,
from item_master im,
 deps d,
 pos_merch_criteria p1
where dl.dept = d.dept
and p1.dept = d.dept
and p1.group_no = d.group_no
and p1.pos_config_type = :ps_pos_config_type
and p1.pos_config_id = :ps_pos_config_id
and im.item_level = im.tran_level
and ((p1.dept is NULL)
or (im.dept = p1.dept
and p1.classt is NULL)
or (iml.dept = p1.dept
and im.class = p1.class
and p1.subclass is NULL)
or (im.dept = p1.dept
and p1.class = im.class
and p1.subclass = im.subclass
and p1.item is NULL)
or (im.dept = p1.dept
and p1.class = im.class
and p1.subclass = im.subclass
and p1.item = im.item
and p1.exclude_ind = 'N')
and NOT EXISTS (select p2.pos_config_id
from pos_merch_criteria p2
where p2.item = p1.item
and p2.pos_config_id = :ps_pos_config_id
and p2.pos_config_type = :ps_pos_config_type
and p2.exclude_ind = 'Y'))
order by 1
```

The item linked list definition is as follows:

```

typedef struct item_rec
{
 char as_item[LEN_ITEM];
 char as_status[LEN_STATUS];
 struct item_rec *ar_next_rec;
} item_linked_list;

```

Note: the as\_store variable will only be populated from records added in the process\_items() function.

### Process\_items

For each coupon and product restriction record, fetch any items that exist on the pos\_config\_items table. Since the process\_criteria() function has potentially added items to the item linked list, a check must be made to determine if the item has already been added. It is also imperative the correct status is associated with the items. Below is a matrix indicating how different possible situations should be handled with the item array.

|                                | Exist in Linked List<br>(in 'C'hanged status) | Doesn't Exist in Linked List                |
|--------------------------------|-----------------------------------------------|---------------------------------------------|
| Item in 'A'dd status           | Do nothing.                                   | Add item to linked list in 'A'dd status.    |
| Item in 'D'elele status        | Change status in linked list to 'D'           | Add item to linked list in 'D'elele status. |
| Item in 'R'etail change status | Do nothing                                    | Add item to linked list in 'R'etail status  |

The syntax of the select statement to retrieve items should be similar to the following:

```

Select item,
 store,
 status,
 retail
From pos_config_items
Where pos_config_type = :ps_pos_config_type
And pos_config_id = :ps_pos_config_id;

```

### Process\_stores

If the POS configuration has changed, the merch criteria has changed, and/or, records exist on pos\_config\_items, select all stores from pos\_store into the store array. If none of the conditions are met (i.e. only changes are stores being added to and deleted from the POS configuration) then select stores from pos\_store where status is not null.

The store status will be set to 'A' when:

1. The POS configuration is new to the store (*header.config\_status = 'A'*).
2. The store is added to a POS configuration (*pos\_store.status = 'A'*).

The store status will be set to 'C' when:

1. The POS configuration has been modified (*header.config\_status = 'C'*).
2. The merchandise criteria has been modified (*pos\_merch\_criteria.status = 'A', 'C', or 'D'*).
3. Additional items have been added/deleted/reclassified to POS configurations at the store (*pos\_config\_items.status = 'A', 'D', 'R'*).

The store status will be set to 'D' when:

1. The POS configuration has been deleted (*header.config\_status = 'D'*).
2. The store is deleted from a POS configuration (*pos\_store.status = 'D'*).

The syntax of the select statement to retrieve stores should be similar to the following:

```
Select store,
 status,
 retail
From pos_store
Where pos_config_type = :ps_pos_config_type
And pos_config_id = :ps_pos_config_id
And status is NOT NULL;
```

In a loop, write the records from the store array to the flat file. If the POS configuration type is coupon or product restriction call the `write_item_to_file()` function to validate the item/store relationship and write items associated to the output file.

### **Write\_item\_to\_file()**

In a loop, call the `ITEMLOC_ATTRIB_SQL.ITEM_STORE_EXISTS` function to validate that the item/store exists in the RMS database. If the relationship exists, write the item to the flat file. If the relationship does not exist, skip the item and do not write it to the flat file.

### Delete\_records()

This function will delete records from the appropriate tables that are in delete status. It is imperative that tables are deleted in the correct order.

1. Delete `pos_config_items` for the `pos_config_type/pos_config_id`
2. Delete `pos_store` for the `pos_config_type/pos_config_id` and status is 'D'
3. Delete `pos_merch_criteria` for `pos_config_type/pos_config_id` and status is 'D'
4. Delete from all the header tables (9 tables) where status = 'D'

Note: In the case of Product Restrictions information from the `pos_day_time_date` table will need to be deleted from the system before the deletion from `pos_prod_rest_head` can occur. Delete all records from `pos_day_time_date` where `pos_day_time_date.pos_prod_rest_id = pos_prod_rest_head.pos_prod_rest_id` and `pos_prod_rest_head.status = 'D'`. Then continue with the deletion from `pos_prod_rest_head`.

### Build\_item\_list()

This function will copy records from the items array to the items linked list. If the linked list is empty, define the current item record as the first record in the linked list and allocate memory for it. If the linked list is not empty, save the record address as the next record in the linked list, then move ahead to the next record in the linked list and allocate memory. After memory allocation has occurred, copy in the array record into the linked list. Finally, set the next record pointer to NULL.

#### Free linked list memory()

Starting at the first record of the linked list, loop through the list and free memory until the last record is reached (current record pointer is NULL). When completed, set the first and current record pointers to NULL.

#### **I/O Specification**

All files layouts input and output

#### **Record type relationship**

```

FHEAD
 TCOUP
 TSTOR
 TITEM
 TPRES
 TSTOR
 TITEM
 TMORD
 TSTOR
 TTTYF
 TSTOR
 TBTTN
 TSTOR
 TITEM
 TPYIO
 TSTOR
 TSPAY
 TSTOR
FTAIL

```

| Record Type | Field       | Datatype   | Valid Values | Comments                                             |
|-------------|-------------|------------|--------------|------------------------------------------------------|
| FHEAD       | Record Type | Char(5)    | 'FHEAD'      | Record Identifier                                    |
|             | Line ID     | Number(10) | 0000000001   | Sequential Line Identifier                           |
|             | File Name   | Char(4)    | 'POSC'       | File Identifier                                      |
|             | File Date   | Char(14)   |              | Date the file was created in 'YYYYMMDDHHMMSS' format |
| TCOUP       | Record Type | Char(5)    | 'TCOUP'      | Record Identifier                                    |
|             | Line ID     | Number(10) |              | Sequential Line Identifier                           |
|             | Coupon ID   | Number(6)  |              |                                                      |

|       |                              |              |             |                                                                                |
|-------|------------------------------|--------------|-------------|--------------------------------------------------------------------------------|
|       | Coupon Desc                  | Char(40)     |             |                                                                                |
|       | Currency Code                | Char2(3)     |             |                                                                                |
|       | Max Discount Amount          | Number(20,4) |             |                                                                                |
|       | Amount                       | Number(20,4) |             |                                                                                |
|       | Percent Ind                  | Char(1)      |             |                                                                                |
|       | Profit Center                | Char(6)      |             |                                                                                |
|       | Tax Class                    | Char(6)      |             |                                                                                |
|       | Export Code                  | Char(6)      |             |                                                                                |
|       | Effective Date               | Char(14)     |             |                                                                                |
|       | Expiration Date              | Char(14)     |             |                                                                                |
|       | Prompted Ind                 | Char(1)      |             |                                                                                |
|       | Display Ind                  | Char(1)      |             |                                                                                |
|       | Status                       | Char(1)      | 'A','C','D' | Indicates if the POS configuration is new, has been changed, or being deleted. |
| TPRES | Record Type                  | Char(5)      | 'TPRES'     | Record Identifier                                                              |
|       | Line ID                      | Number(10)   |             | Sequential Line Identifier                                                     |
|       | POS Product Restriction ID   | Number(6)    |             |                                                                                |
|       | POS Product Restriction Desc | Char(40)     |             |                                                                                |
|       | POS Product Restriction Type | Char(6)      |             |                                                                                |
|       | Effective Date               | Char(14)     |             |                                                                                |
|       | Currency Code                | Char(3)      |             |                                                                                |
|       | Product Restriction Amount   | Number(20,4) |             |                                                                                |
|       | Age Minimum                  | Number(2)    |             |                                                                                |
|       | Date Restriction             | Char(14)     |             |                                                                                |
|       | Before Time Restriction      | Char(6)      |             |                                                                                |
|       | After Time Restriction       | Char(6)      |             |                                                                                |
|       | Day Restriction              | Char(6)      |             |                                                                                |
|       | Max Qty Amount               | Number(12,4) |             |                                                                                |
|       | Tender Type Group            | Char(6)      |             |                                                                                |
|       | Status                       | Char(1)      | 'A','C','D' | Indicates if the POS                                                           |

|       |                   |               |             |                                                                                |
|-------|-------------------|---------------|-------------|--------------------------------------------------------------------------------|
|       |                   |               |             | configuration is new, has been changed, or being deleted.                      |
| TMORD | Record Type       | Char(5)       | 'TMORD'     | Record Identifier                                                              |
|       | Line ID           | Number(10)    |             | Sequential Line Identifier                                                     |
|       | Money Order ID    | Number(6)     |             |                                                                                |
|       | Money Order Desc  | Char(40)      |             |                                                                                |
|       | Currency Code     | Char(3)       |             |                                                                                |
|       | Fee               | Number (20,4) |             |                                                                                |
|       | Max Face Value    | Number (20,4) |             |                                                                                |
|       | Max Sale Amount   | Number (20,4) |             |                                                                                |
|       | Tax Class         | Char (6)      |             |                                                                                |
|       | Cash Only Ind     | Char (1)      |             |                                                                                |
|       | Serial Input Ind  | Char(1)       |             |                                                                                |
|       | Pack Size         | Number (6)    |             |                                                                                |
|       | Refund Fee Ind    | Char (1)      |             |                                                                                |
|       | Expiration Date   | Char(14)      |             |                                                                                |
|       | Effective Date    | Char(14)      |             |                                                                                |
|       | Status            | Char(1)       | 'A','C','D' | Indicates if the POS configuration is new, has been changed, or being deleted. |
| TTTYP | Record Type       | Char(5)       | 'TTTYP'     | Record Identifier                                                              |
|       | Line ID           | Number(10)    |             | Sequential Line Identifier                                                     |
|       | Tender Type ID    | Char(6)       |             |                                                                                |
|       | Tender Type Desc  | Char (40)     |             |                                                                                |
|       | Tender Type Group | Char (6)      |             |                                                                                |
|       | Effective Date    | Char(14)      |             |                                                                                |
|       | Currency Code     | Char(3)       |             |                                                                                |
|       | Preset Amount     | Number(20,4)  |             |                                                                                |
|       | Authorize Minimum | Number(20,4)  |             |                                                                                |
|       | Open Drawer Ind   | Char (1)      |             |                                                                                |
|       | Exact Change Ind  | Char (1)      |             |                                                                                |



|       |                            |            |             |                                                                                |
|-------|----------------------------|------------|-------------|--------------------------------------------------------------------------------|
|       | Accumulate Cash Intake Ind | Char (1)   |             |                                                                                |
|       | Next Dollar Ind            | Char (1)   |             |                                                                                |
|       | Deposit in Bank Ind        | Char (1)   |             |                                                                                |
|       | Deposit Override Ind       | Char (1)   |             |                                                                                |
|       | Automatic Deposit Ind      | Char (1)   |             |                                                                                |
|       | Pay In Deposit Ind         | Char (1)   |             |                                                                                |
|       | Ask For Invoice Ind        | Char (1)   |             |                                                                                |
|       | Imprint Ind                | Char (1)   |             |                                                                                |
|       | Imprint Tender Type        | Char (20)  |             |                                                                                |
|       | Show in Breakdown Ind      | Char (1)   |             |                                                                                |
|       | Display Ind                | Char (1)   |             |                                                                                |
|       | Discrepancy Display Type   | Char (6)   |             |                                                                                |
|       | Processor Type             | Char (6)   |             |                                                                                |
|       | Export Code                | Char(6)    |             |                                                                                |
|       | Profit Center              | Char(6)    |             |                                                                                |
|       | Phone Authorization Type   | Char(6)    |             |                                                                                |
|       | Status                     | Char(1)    | 'A','C','D' | Indicates if the POS configuration is new, has been changed, or being deleted. |
| TBTTN | Record Type                | Char(5)    | 'TBTTN'     | Record Identifier                                                              |
|       | Line ID                    | Number(10) |             | Sequential Line Identifier                                                     |
|       | Button configuration ID    | Number (6) |             |                                                                                |
|       | Button Config Desc         | Char(40)   |             |                                                                                |
|       | Button ID                  | Number (6) |             |                                                                                |
|       | Button Desc                | Char (40)  |             |                                                                                |
|       | Layer Number               | Number (1) |             |                                                                                |
|       | Sequence Number            | Number (4) |             |                                                                                |
|       | Button Type                | Char(6)    |             | Valid Values include:<br>U = Multi Selection<br>S = Single Selection           |

|       |                          |            |             |                                                                                                  |
|-------|--------------------------|------------|-------------|--------------------------------------------------------------------------------------------------|
|       |                          |            |             | I = Single Item<br>M = Major Category<br>N = Minor Category<br>X = No Button<br>P = Programmable |
|       | Effective Date           | Char(14)   |             |                                                                                                  |
|       | Text Color               | Char (6)   |             |                                                                                                  |
|       | Background Color         | Number (6) |             |                                                                                                  |
|       | Department               | Char(4)    |             |                                                                                                  |
|       | Text                     | Char (10)  |             |                                                                                                  |
|       | Parent Button ID         | Char(6)    |             |                                                                                                  |
|       | Status                   | Char(1)    | 'A','C','D' | Indicates if the POS configuration is new, has been changed, or being deleted.                   |
| TPYIO | Record Type              | Char(5)    | 'TPYIO'     | Record Identifier                                                                                |
|       | Line ID                  | Number(10) |             | Sequential Line Identifier                                                                       |
|       | Pay In Out ID            | Number(6)  |             |                                                                                                  |
|       | Pay In Out Desc          | Char(40)   |             |                                                                                                  |
|       | Pay In Out Type          | Char(1)    |             |                                                                                                  |
|       | Profit Center            | Char(6)    |             |                                                                                                  |
|       | Export Code              | Char(6)    |             |                                                                                                  |
|       | Tax Chart Code           | Char(6)    |             |                                                                                                  |
|       | Invoice Link             | Char(1)    |             |                                                                                                  |
|       | Display Ind              | Char(1)    |             |                                                                                                  |
|       | Effective Date           | Char(14)   |             |                                                                                                  |
|       | Status                   | Char(1)    | 'A','C','D' | Indicates if the POS configuration is new, has been changed, or being deleted.                   |
| TSPAY | Record Type              | Char(5)    | 'TSPAY'     | Record Identifier                                                                                |
|       | Line ID                  | Number(10) |             | Sequential Line Identifier                                                                       |
|       | Supplier                 | Number(10) |             |                                                                                                  |
|       | Pay Type                 | Char(6)    |             | Valid Values include:<br>CASH = Cash<br>MP = Money Order<br>INVC = Invoice                       |
|       | Supplier Payment Type ID | Number(6)  |             |                                                                                                  |
|       | Effective Date           | Char(14)   |             |                                                                                                  |
|       | Status                   | Char(1)    | 'A','C','D' | Indicates if the POS configuration is new, has been changed, or being deleted.                   |

|       |             |            |                                           |                                                                                                                                 |
|-------|-------------|------------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| TSTOR | Record Type | Char(5)    | 'TSTOR'                                   | Record Identifier                                                                                                               |
|       | Line ID     | Number(10) |                                           | Sequential Line Identifier                                                                                                      |
|       | Store       | Number(4)  |                                           | Store Identifier                                                                                                                |
|       | Status      | Char(1)    | 'A' – Add<br>'D' – Delete                 | Indicates to the POS if the POS configuration must be added or deleted.                                                         |
| TITEM | Record Type | Char(5)    | 'TITEM'                                   | Record Identifier                                                                                                               |
|       | Line ID     | Number(10) |                                           | Sequential Line Identifier                                                                                                      |
|       | Item        | Char(25)   |                                           |                                                                                                                                 |
|       | Status      | Char(1)    | 'A' – Add<br>'D' – Delete<br>'C' – Change | Indicates the item's status at the POS. Overlays of items as a result of a change to the merch criteria will have a 'C' status. |
|       | Button ID   | Number(6)  | ID                                        | Indicates the identification number of the button.                                                                              |

## Point of sale download [posdnld]

### Design Overview

The posdnld program is used to download pos\_mods records created in the RMS to the store POS systems. This program has one output file which contains all records for all stores in a given run. This program uses the Retek standard file format FHEAD, FDETL, FTAIL.

### Program Flow



### Stored Procedures / Shared Modules (Maintainability)

pos\_config\_sql.check\_item - Updates POS item configuration information that is downloaded to the stores by poscdnld.pc.

### Input Specifications

All input comes from the pos\_mods table. All columns of this table can be NULL with the exception of tran\_type and store. Most columns should default to blank (spaces) with the exception of:

- new\_price, new\_multi\_units, new\_multi\_units\_retail, proportional\_tare\_pct and fixed\_tare\_value. These should default to zero (0).
- start\_date, start\_time and end\_time. These should default to period.vdate + 1.

### Output Specifications

#### Output File

| Record Name | Field Name                        | Field Type       | Default Value                            | Description                                          |
|-------------|-----------------------------------|------------------|------------------------------------------|------------------------------------------------------|
| File Header | File Type<br>Record<br>Descriptor | Char(5)          | FHEAD                                    | Identifies file record type                          |
|             | File Line<br>Identifier           | Number<br>ID(10) | Sequential number<br>Created by program. | ID of current line being<br>created for output file. |
|             | File Type<br>Definition           | Char(4)          | POSD                                     | Identifies file as 'POS<br>Download'                 |

|             |                             |               |                                        |                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|-----------------------------|---------------|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             | File Create Date            | Char(8)       | Create date (vdate).                   | Current date, formatted to 'YYYYMMDD'.                                                                                                                                                                                                                                                                                                                                           |
| File Detail | File Type Record Descriptor | Char(5)       | FDETL                                  | Identifies file record type                                                                                                                                                                                                                                                                                                                                                      |
|             | File Line Identifier        | Number ID(10) | Sequential number. Created by program. | ID of current line being created for output file.                                                                                                                                                                                                                                                                                                                                |
|             | Location Number             | Number(10)    | Store                                  | Contains the store location that has been affected by the transaction                                                                                                                                                                                                                                                                                                            |
|             | Update Type                 | Char(1)       | Update type. Created by program.       | Code used for client specific POS system. <ul style="list-style-type: none"> <li>• 1 - Transaction Types 1 &amp; 2.</li> <li>• 2 - Transaction Types 10 thru 18, 31 &amp; 32, 50 thru 57, 59 thru 64.</li> <li>• 3 - Transaction Types 21 &amp; 22</li> <li>• 4 - Transaction Types 25 &amp; 26</li> <li>• 0 - All other Transaction Types. These should never exist.</li> </ul> |
|             | Start Date                  | Char(8)       | Start_date or vdate + 1 if NULL.       | The effective date for the action determined by the transaction type of the record. Formatted to 'YYYYMMDD'.                                                                                                                                                                                                                                                                     |
|             | Time                        | Char(6)       | Start_time, End_time or start_date.    | This field will be used in conjunction with starting a promotion (Transaction Type = 31). Start time will indicate the time of day that the promotion is scheduled to start. This field will also be used in conjunction with ending a promotion (Transaction Type = 32). Any other Transaction Type will use the time from the start_date column. Formatted to 'HH24MISS'.      |
|             | Transaction Type            | Number(2)     | Tran_type                              | Indicates the type of transaction to determine                                                                                                                                                                                                                                                                                                                                   |

|  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--|--|--|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  |  |  |  | <p>what Retek action is being sent down to the stores from the Retek pos_mods table.</p> <p>Valid values include:</p> <ul style="list-style-type: none"> <li>• 01 - Add new transaction level item</li> <li>• 02 - Add new lower than transaction level item</li> <li>• 10 - Change Short Description of existing item</li> <li>• 11 - Change Price of an existing item</li> <li>• 12 - Change Description of an existing item</li> <li>• 13 - Change Department/Class/Sub class of an existing item</li> <li>• 16 - Put Item on Clearance</li> <li>• 17 - Change existing item's Clearance Price</li> <li>• 18 - Remove Item from Clearance and Reset</li> <li>• 21 - Delete existing transaction level item</li> <li>• 22 - Delete existing lower than transaction level item</li> <li>• 25 - Change item's status</li> <li>• 26 - Change item's taxable indicator</li> <li>• 31 - Promotional item - Start maintenance</li> <li>• 32 - Promotional item - End maintenance</li> <li>• 50 - Change item's launch date</li> <li>• 51 - Change item's quantity key options</li> <li>• 52 - Change item's manual price entry</li> </ul> |
|--|--|--|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|  |                  |           |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--|------------------|-----------|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  |                  |           |                  | <p>options</p> <ul style="list-style-type: none"> <li>• 53 - Change item's deposit code</li> <li>• 54 - Change item's food stamp indicator</li> <li>• 55 - Change item's WIC indicator</li> <li>• 56 - Change item's proportional tare percent</li> <li>• 57 - Change item's fixed tare value</li> <li>• 58 - Change item's rewards eligible indicator</li> <li>• 59- Change item's electronic marketing clubs</li> <li>• 60 - Change item's return policy</li> <li>• 61 - Change item's stop sale indicator</li> <li>• 62 – Change item's returnable indicator</li> <li>• 63 – Change item's refundable indicator</li> <li>• 64 – Change item's back order indicator</li> </ul> |
|  | Item Number ID   | Char(25)  | Item             | This field identifies the unique alphanumeric value for the transaction level item. The ID number of a item from the Retek item_master table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|  | Item Number Type | Char(6)   | Item_number_type | This field identifies the type of the item number ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|  | Format ID        | Char(1)   | Format_id        | This field identifies the type of format used if the item_number_type is 'VPLU'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|  | Prefix           | Number(2) | Prefix           | This field identifies the prefix used if the item_number_type is 'VPLU'. In case of single digit prefix, the field will be right-justified with blank padding.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

|  |                            |            |                      |                                                                                                                                                                                                                                                     |
|--|----------------------------|------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Reference Item             | Char(25)   | Ref_item             | This field identifies the unique alphanumeric value for an item one level below the transaction level item.                                                                                                                                         |
|  | Reference Item Number Type | Char(6)    | Ref_Item_number_type | This field identifies the type of the ref item number ID.                                                                                                                                                                                           |
|  | Reference Item Format ID   | Char(1)    | Ref_Format_id        | This field identifies the type of format used if the ref item_number_type is 'VPLU'.                                                                                                                                                                |
|  | Reference Item Prefix      | Number(2)  | Ref_Prefix           | This field identifies the prefix used if the ref item_number_type is 'VPLU'. In case of single digit prefix, the field will be right-justified with blank padding.                                                                                  |
|  | Item Short Description     | Char(20)   | Item_short_desc      | Contains the short description associated with the item.                                                                                                                                                                                            |
|  | Item Long Description      | Char(100)  | Item_long_desc       | Contains the long description associated with the item.                                                                                                                                                                                             |
|  | Department ID              | Number(4)  | Dept                 | Contains the item's associated department.                                                                                                                                                                                                          |
|  | Class ID                   | Number(4)  | Class                | Contains the item's associated class.                                                                                                                                                                                                               |
|  | Subclass ID                | Number(4)  | Subclass             | Contains the item's associated subclass.                                                                                                                                                                                                            |
|  | New Price                  | Number(20) | New_price            | Contains the new effective price in the selling unit of measure for an item when the transaction type identifies a change in price. Otherwise, the current retail price is used to populate this field. This field is stored in the local currency. |
|  | New Selling UOM            | Char(4)    | New_selling_UOM      | Contains the new selling unit of measure for an item's single-unit retail.                                                                                                                                                                          |



|  |                        |            |                        |                                                                                                                                                                                                                                                                                                                                               |
|--|------------------------|------------|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | New Multi Units        | Number(12) | New_multi_units        | Contains the new number of units sold together for multi-unit pricing. This field is only filled when a multi-unit price change is being made.                                                                                                                                                                                                |
|  | New Multi Units Retail | Number(20) | New_multi_units_retail | Contains the new price in the selling unit of measure for units sold together for multi-unit pricing. This field is only filled when a multi-unit price change is being made. This field is stored in the local currency.                                                                                                                     |
|  | New Multi Selling UOM  | Char(4)    | New_multi_selling_UOM  | Contains the new selling unit of measure for an item's multi-unit retail.                                                                                                                                                                                                                                                                     |
|  | Status                 | Char(1)    | Status                 | Populates if tran_type for the item is 1 (new item added) or 25 (change item status) or 26 (change taxable indicator).<br>Contains the current status of the item at the store.<br>Valid values are: <ul style="list-style-type: none"> <li>• A = Active</li> <li>• I = Inactive</li> <li>• D = Delete</li> <li>• C = Discontinued</li> </ul> |
|  | Taxable Indicator      | Char(1)    | Taxable_ind            | Populates if tran_type for the item is 1 (new item added) or 25 (change item status) or 26 (change taxable indicator).<br>Indicates whether the item is taxable at the store.<br>Valid values are 'Y' or 'N'.                                                                                                                                 |
|  | Promotion Number       | Number(10) | Promotion              | This field contains the number of the promotion for which the discount originated. This field, along with the Mix Match Number or Threshold Number is used to isolate a list of items that tie together with discount information.                                                                                                            |

|  |                      |            |                 |                                                                                                                                                                                                                                                                                                                                                                |
|--|----------------------|------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Mix Match Number     | Number(10) | Mix_match_no    | This field contains the number of the mix and match in a promotion for which the discount originated. This field, along with the promotion, is used to isolate a list of items which tie together with the mix and match discount information.                                                                                                                 |
|  | Mix Match Type       | Char(1)    | Mix_match_type  | This field identifies which types of mix and match record this item belongs to. The item can either be a buy (exists on PROM_MIX_MATCH_BUY) or a get (exists on PROM_MIX_MATCH_GET) item. This field is only populated when the MIX_MATCH_NO is populated.<br>Valid values are: <ul style="list-style-type: none"> <li>• B - Buy</li> <li>• G - Get</li> </ul> |
|  | Threshold Number     | Number(10) | Threshold_no    | This field contains the number of the threshold in a promotion for which the discount originated. This field, along with the promotion, is used to isolate a list of items that tie together with discount information.                                                                                                                                        |
|  | Launch Date          | Char(8)    | Launch_date     | Date that the item should first be sold at this location, formatted to 'YYYYMMDD'.                                                                                                                                                                                                                                                                             |
|  | Quantity Key Options | Char(6)    | Qty_key_options | Determines whether the price can/should be entered manually on a POS for this item at the location. Valid values are in the code_type 'RPO'. Current values include 'R - required', 'P - Prohibited'.                                                                                                                                                          |

|  |                           |            |                       |                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--|---------------------------|------------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Manual Price Entry        | Char(6)    | Manual_price_entry    | Determines whether the price can/should be entered manually on a POS for this item at the location. Valid values are in the code_type 'RPO'. Current values include 'R - required', 'P - Prohibited', and 'O - Optional'.                                                                                                                                                                                                       |
|  | Deposit Code              | Char(6)    | Deposit_code          | Indicates whether a deposit is associated with this item at the location. Valid values are in the code_type 'DEPO'. Additional values may be added or removed as needed. Deposits are not subtracted from the retail of an item uploaded to RMS, etc. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.                                              |
|  | Food Stamp Indicator      | Char(1)    | Food_stamp_ind        | Indicates whether the item is approved for food stamps at the location.                                                                                                                                                                                                                                                                                                                                                         |
|  | WIC Indicator             | Char(1)    | Wic_ind               | Indicates whether the item is approved for WIC at the location.                                                                                                                                                                                                                                                                                                                                                                 |
|  | Proportional Tare Percent | Number(12) | Proportional_tare_pct | Holds the value associated of the packaging in items sold by weight at the location. The proportional tare is the proportion of the total weight of a unit of an item that is packaging (i.e. if the tare item is bulk candy, this is the proportional of the total weight of one piece of candy that is the candy wrapper). The only processing RMS does involving the proportional tare percent is downloading it to the POS. |

|  |                           |            |                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--|---------------------------|------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Fixed Tare Value          | Number(12) | Fixed_tare_value        | Holds the value associated of the packaging in items sold by weight at the location. Fixed tare is the tare of the packaging used to (i.e. if the tare item is bulk candy, this is weight of the bag and twist tie). The only processing RMS does involving the fixed tare value is downloading it to the POS. Fixed tare is not subtracted from items sold by weight when sales are uploaded to RMS, etc. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application. |
|  | Fixed Tare UOM            | Char(4)    | Fixed_tare_uom          | Holds the unit of measure value associated with the tare value. The only processing RMS does involving the proportional tare value and UOM is downloading it to the POS. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.                                                                                                                                                                                                                                   |
|  | Reward Eligible Indicator | Char(1)    | Reward_eligible_in<br>d | Holds whether the item is legally valid for various types of bonus point/award programs at the location.                                                                                                                                                                                                                                                                                                                                                                                                                                |
|  | Elective Marketing Clubs  | Char(6)    | Elect_mtk_clubs         | Holds the code that represents the marketing clubs to which the item belongs at the location. Valid values can belong to the code_type 'MTKC'. Additional values can be added or removed from the code type as needed                                                                                                                                                                                                                                                                                                                   |

|              |                             |               |                                              |                                                                                                                                                          |
|--------------|-----------------------------|---------------|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | Return Policy               | Char(6)       | Return_pocily                                | Holds the return policy for the item at the location. Valid values for this field belong to the code_type 'RETP'.                                        |
|              | Stop Sale Indicator         | Char(1)       | Stop_sale_ind                                | Indicates that sale of the item should be stopped immediately at the location (i.e. in case of recall etc).                                              |
|              | Returnable Indicator        | Char(1)       | Returnable_ind                               | Indicates that the item is returnable at the location when equal to 'Y'es. Indicates that the item is not returnable at the location when equal to 'N'o. |
|              | Refundable Indicator        | Char(1)       | Refundable_ind                               | Indicates that the item is refundable at the location when equal to 'Y'es. Indicates that the item is not refundable at the location when equal to 'N'o. |
|              | Back Order Indicator        | Char(1)       | Back_order_ind                               | Indicates that the item is back orderable at the location when equal to 'Y'. Indicates that the item is not back orderable when equal to 'N'o.           |
| File Trailer | File Type Record Descriptor | Char(5)       | FTAIL                                        | Identifies file record type                                                                                                                              |
|              | File Line Identifier        | Number ID(10) | Sequential number. Created by program.       | ID of current line being created for output file.                                                                                                        |
|              | File Record Counter         | Number ID(10) | Number of FDETL records. Created by program. | Number of records/transactions processed in current file (only records between head & tail)                                                              |

### Function Level Description

init - This function initializes restart/recovery for this program. It also retrieves system variables (period.vdate and vdate + 1), opens the output file and write the FHEAD record.

process - This function drives the processing of the program. It calls size\_arrays() function to size the arrays used in this program and also, when done, it calls free\_arrays() to release any memory

it has been allocated. The driving cursor is opened and fetched here, which retrieves all the records from pos\_mods where the pos\_mods.store value is greater than zero.

If the Transaction Type is 31, then the time field returned by the cursor should be the start time, else if the Transaction Type is 32, then the time field should be the end time. If the Transaction Type is something else or if either the start time or end time is NULL, blanks should be used. Once the records are fetched, if the Transaction Type of the record fetched is 1 or 21 then pos\_config\_check() is called. The write\_rec() function is called to perform processing on all records fetched. Restart/Recovery and committing of records is also performed here.

**final** - This function will finish restart/recovery logic, write the FTAIL record and close the output.

**size\_arrays** - This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart\_control.

**free\_arrays** - This function frees the array allocated in size\_arrays.

**write\_rec** - This function will prepare records for insert into the output file and write them as FDETL records. The Transaction Type will determine the Update Type. If the Transaction Type is 1, 25 or 26 then the status and taxable\_ind columns must be outputted, otherwise these should remain blank.

**pos\_config\_check** - This function will call the package pos\_config\_sql.check\_item(). If the Transaction type is 1, then a status of 'A' will be passed in. If the Transaction Type is 21, then a status of 'D' will be passed in. This function body should be commented out for A&P Phase 1a.

### **Scheduling Considerations**

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program is run towards the end of the batch run when all pos\_mods records have been created for the transaction day.

Pre-Processing: N/A

Post-Processing: prepost.pc - posdnld\_post() – records in POS\_MODS are truncated.

Threading Scheme: v\_restart\_store

### **Locking Strategy**

None.

### **Restart/Recovery**

Restart/recovery for this program is set up at the store/item or item level. Threading is done by store using the v\_restart\_store view to thread properly.

### **Performance Considerations**

Both table and file restart/recovery must be used.

The `commit_max_ctr` field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

**Security Considerations**

Price changes for all stores are stored in a Unix file with the processes default permissions (umask). Care should be exercised so that this file cannot be tampered with.

**Design Assumptions**

Data columns required by a particular Transaction Type are filled in and correct.

## Group and department POS download [posgpdlld]

Occasionally, retailers need to perform department level sales for merchandise. To accommodate this, a new download program (posgpdlld.pc) is created to download group number/department information to individual Point-of-Sale (POS) systems.

Assumption: The dataset that defines group numbers and departments is small. With this assumption, a full extract of RMS group number/department combinations occurs for every run of the batch program.

### Scheduling Constraints

|                                   |
|-----------------------------------|
| <i>Pre/Post Logic Description</i> |
|-----------------------------------|

Processing Cycle: ad hoc

Scheduling Diagram: Run after reclsdly.pc

Pre-Processing: reclsdly.pc

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

Logical Unit of Work ( recommended Commit check points )  
Driving Cursor

### Logical Unit of Work

Group\_no/dept

### Driving Cursor

```
SELECT group_no,
 dept,
 ';' || to_char(group_no) || ';' || to_char(dept)
FROM deps
WHERE (group_no > NVL(:ps_restart_group_no, -999)
 OR (group_no = :ps_restart_group_no
 AND dept > :ps_restart_dept))
ORDER BY group_no, dept;
```

### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage  
N/A

### Function Level Description

All database interactions required and error handling considerations



Global structure of arrays for data processing:

```
struct fetch_rec
{
 char (*as_group_no) [NULL_GROUP_NO];
 char (*as_dept) [NULL_DEPT];
 char (*as_restart_string) [NULL_RESTART]
};
```

Called from init():

```
int get_date()
```

Fetches the vdate (business date) from the period table and places it into a program global variable.

```
int size_array(struct fetch_recs *ioa_data_buffer)
```

Sizes the arrays used in database FETCHES by using the calloc function.

```
void format_strings()
```

Creates format strings used in file outputs.

```
int init()
```

Performs standard Retek restart/recovery initialization. All file handling occurs here, such as declaration of a temp file, and opening of files. The FHEAD line is also written to the file, in the case of a fresh start.

```
int process()
```

The driving cursor is used to fetch records from the database. The group\_no and dept are written to the temp file using the FDETL format string defined previously. Once all data has been fetched an FTAIL line is written to the file.

```
int final()
```

Performs closing operations. A call to restart\_close() is made, and the temp file is removed from the system.

### I/O Specification

All files layouts input and output

| Record Type | Field       | Datatype   | Valid Values | Description                                          |
|-------------|-------------|------------|--------------|------------------------------------------------------|
| FHEAD       | Record Type | Char(5)    | 'FHEAD'      | Record Identifier                                    |
|             | Line ID     | Number(10) | 0000000001   | Line Sequence Identifier                             |
|             | File Name   | Char(4)    | 'GRPD'       | File Identifier                                      |
|             | File Date   | Char(14)   |              | Date the file was created in 'YYYYMMDDHHMMSS' format |
| FDETL       | Record Type | Char(5)    | 'FDETL'      | Record Identifier                                    |
|             | Line ID     | Number(10) |              | Line Sequence Identifier                             |

|       |             |                                                    |         |                                            |
|-------|-------------|----------------------------------------------------|---------|--------------------------------------------|
|       | Group no    | Numeric ID (4)<br>(left-justified character field) |         | Valid RMS Group Number.                    |
|       | Dept        | Numeric ID (4)<br>(left-justified character field) |         | Valid RMS Department for the Group Number. |
| FTAIL | Record Type | Char(5)                                            | 'FTAIL' | Record Identifier                          |
|       | Line ID     | Number(10)                                         |         | Line Sequence Identifier                   |
|       | Line Total  | Number(10)                                         |         | Total number of FDETL lines in the file.   |

## Point of sale upload [posupld]

### Design Overview

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

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

### Scheduling Constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program will likely be run at the beginning of the batch run during the POS polling cycle. It can be scheduled to run multiple times throughout the day, as POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

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

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

all database transactions. The file writing logic will append the temporary holding files to the final output files.

The `commit_max_ctr` field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

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

### **Program Flow**

N/A

### **Shared Modules**

`validate_all_numeric`: intrface library function.

`validate_all_numeric_signed`: intrface library function.

`valid_date`: intrface library function.

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

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

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

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

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

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

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

### **Function Level Description**

Declarations:

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

init()

initialize restart recovery

open input file (posupld)

- file should be specified as input parameter to program

fetch system variables

Retrieve all valid promotion types

declare final output filename (used in restart\_write\_file logic)

open reject file ( as a temporary file for restart )

- file should be specified as input parameter to program

open lock reject file ( as a temporary file for restart )

- file should be specified as input parameter to program

call restart\_file\_init logic

assign application image array variables- line counter (g\_l\_rec\_cnt), reject counter (g\_l\_rej\_cnt),

lock reject file counters (pl\_lock\_cnt, pl\_lock\_dtl\_cnt), store, transaction\_date

if fresh start (l\_file\_start = 0)

read file header record (get\_record)

write FHEAD to lock reject file

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

validate file type = 'POSU'

else fseek to l\_file\_start location

validate location and date are valid

set restart variables to ones from restart image

file\_process()

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

Next, the unit retail from price\_hist will be fetched by calling the get\_unit\_retail() function. The retail retrieved from this function will be compared with the actual retail sent in from the input file to determine any discrepancies in sale amounts.

Fetch all of the TDETL records that exist for the transaction currently being processed until a TTAIL record is encountered. Perform validation on the transaction detail records. If a detail record is found to be invalid, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. If a valid promotion type (code for mix & match, threshold promotions, etc.) was included in the detail record and it is not an employee disc record, write a record to the daily\_sales\_discount table. If it is an employee discount record write an employee discount record to tran\_data. Finally, accumulate the discount amounts for all transaction detail records for the current transaction, unless the record was an employee discount.

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

item\_process()

Check to see if any validation failed for the item before this function was called. If a lock error was found, call write\_lock\_rej() then return. If an other error was found, call write\_rej() and process\_detail\_error() then return.

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

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

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

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

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

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

If the item currently being processed is a packitem, calculate the retail markdown the item takes for being included in the pack and write a transaction data record as a promotional markdown. This markdown is calculated by comparing the retail contribution of the packitem's component item to the packitem to the component item's regular retail found on the `price_hist` table. The retail contribution for a component item is calculated by taking the component item's unit retail from `price_hist`, divided by the total retail of all component items in the packitem, and multiplying the packitem's unit retail. So if the retail contribution of a component item within packitem A is \$10, and the same component item's `price_hist` record has a retail of \$14, and there is only one packitem sold, and this component item has a quantity of one, a `tran_data` Record (`tran_type` 15) will be written for \$4 (assume no vat is used).

Write transaction data records for sales and returns. If the transaction is a sale, write a `tran_data` record with a transaction code of 1 with the total sales. If the system VAT indicator is on and the `system_options.stklmgr_vat_incl_retl_ind` is on, write a `tran_data` record with a transaction code

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

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

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

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

If an off\_retail amount was identified for the item/location, call the write\_off\_retail\_markdowns() function to write tran\_data records (tran\_type 15) to record the difference. If the system\_options.multi\_prom\_ind = 'N' and the item is on promotion, or if the system\_options.multi\_prom\_ind = 'Y' and the TDETL total discount amount is greater than zero, write a promotional markdown. Note: this will also record a tran\_data record (tran\_type 15) for a TDETL record that has a promotional transaction type with no promotion number in order to record the markdown.

If an employee discount TDETL record has been encountered, a tran\_data record with tran\_code 60 will be written.

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

#### process\_detail\_error()

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

#### set\_counters()

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

#### calc\_item\_totals()

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

#### calc\_prom\_totals()

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

#### process\_sales\_and\_returns()

If the item is on consignment and not a packitem, the `consignment_data()` function will be called to perform consignment processing. The function `write_tran` will be called to write a `tran_data` record with a `tran_type` 1 (always written), a `tran_type` 2 (if the `system_options.stkldgr_vat_incl_retl_ind = Y`), and a `tran_type` 4 (if the transaction was a return). If the transaction is a return, any `tran_data` records with `tran_types` of 1 and 2 will be written with negative retails. Also the `update_price_hist()` function will be called to update the most recent `price_hist` record.

#### posting\_and\_restart()

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

#### validate\_FHEAD()

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

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

#### validate\_THEAD()

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

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

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



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

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

### Check\_item\_lock

This function will lock this item/location's record in the RMS item\_loc table. Returns a lock error if lock failed due to contention, otherwise returns 0 if no errors occurred, or fatal if other errors occurred.

### Check\_pack\_lock

This function will call check\_item\_lock for every component item of the current pack item.

### validate\_TDETL

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

The quantity is multiplied by the UOM ratio determined in validate\_THEAD().

If a promotional transaction type is passed in, verify it is valid. If a promotional transaction type is passed in, but it is not valid, return non-fatal error then set non-fatal error flag to true. If a promotion number is passed in, validate it by checking the promhead table and set the promotional indicator to True.

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

### New\_item\_loc

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

### item\_store\_cursors

This function checks the item\_loc for the item / store combination. It is called by the item\_check function.

### item\_check

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

#### New\_pack\_loc

This function creates a new store item relationship for pack items. It is called by pack\_check.

#### pack\_check

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

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

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

#### get\_unit\_retail

This function retrieves the current unit retail and the retail price of the item at the time of the sale from price\_hist for the item/location being processed. If a tran\_code of 8 is returned, the item is on clearance.

#### process\_packitems

This function performs processing for the component items of the packitems. This would include updates/inserts into stake\_item\_loc, edi\_daily\_sales, item\_loc, item\_loc\_hist, vat\_history\_data, and tran\_data. All of these tables do not write records at the packitem level, but at the component item level. When figuring retails to write to these tables, the component items price ratio should always be applied against the packitems retail to come up with the correct retail for each component item. If an employee discount TDETL record has been encountered, an tran\_data record with tran\_code 60 will be written for each component item.

#### process\_daily\_sales\_discount()

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

#### write\_in\_store()

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

#### Remove\_stklgdr\_vat()

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

#### Write\_off\_retail()

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

#### Daily\_sales\_exist()

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

#### Daily\_sales\_insert\_update()

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

#### write\_off\_retail\_markdown()

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

#### write\_promotional\_markdown()

The `write_tran_data()` function will be called to write the promotional markdown unless the item `multi_prom_ind` is off and the transaction is a return, the item is on consignment, or the promotional markdown amount is zero. The `tran_data` new and old retails are only written if the `multi_prom_ind` is off.

#### Write\_wastage\_markdown()

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

#### vat\_convert()

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

#### process\_items()

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

#### process\_pack()

Update the stock on hand on the `item_loc_soh` table for Sales and Returns. Also, update the `item_loc_hist` table for Sale transactions. Do not update for returns.

write\_tran\_data()

Writes a record to the tran\_data insert array.

Write\_edi\_daily\_sales()

Writes a record to edi\_daily\_sales.

update\_snapshot()

Calls the UPDATE\_SNAPSHOT\_SQL.EXECUTE function to update the stake\_sku\_loc and edi\_daily\_sales tables for late transactions.

write\_vat\_err\_message()

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

vat\_history\_data()

Writes a record to the vat\_history table.

consignment\_data()

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

Also a completed order to the vendor should be found/created for the supplier with an orig\_ind = 4 (consignment). Consignment type invoices will be created for all PO's created for consignments

Also a tran\_data record (tran\_type 20) will be written to record the consignment transaction to the stock ledger. The retails should be VAT inclusive or exclusive, depending on the system\_options.stkldgr\_vat\_incl\_retl\_ind.

This function uses support functions: check\_order(), order\_head(), invc\_data(), to handle the order creation-update and the invoice creation-update.

get\_prom\_type\_info()

This function will retrieve all valid promotional transaction types from the code\_detail table. Valid promotional transaction types are those where the code\_type = 'PRMT'.

fill\_packitem\_array()

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

Write\_lock\_rej

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

write\_item\_store\_report()

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

**ON Fatal Error**

- Exit Function with -1 return code

**ON Non-Fatal Error**

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

**Input File**

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

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

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

| Record Name | Field Name            | Field Type | Default Value                | Description                                                                                                      |
|-------------|-----------------------|------------|------------------------------|------------------------------------------------------------------------------------------------------------------|
| File Header | File Type             | Char(5)    | FHEAD                        | Identifies file record type                                                                                      |
|             | Record Descriptor     |            |                              |                                                                                                                  |
|             | File Line Identifier  | Char(10)   | specified by external system | ID of current line being processed by input file.                                                                |
|             | File Type Definition  | Char(4)    | POSU                         | Identifies file as 'POS Upload'                                                                                  |
|             | File Create Date      | Char(14)   | create date                  | date file was written by external system                                                                         |
|             | Location Number       | Number(10) | specified by external system | Store identifier                                                                                                 |
|             | Vat include indicator | Char(1)    |                              | Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit |
|             | Vat region            | Number(4)  |                              | Vat region the given location is in. Not required                                                                |

|                    |                             |           |                              |                                                                                                                    |
|--------------------|-----------------------------|-----------|------------------------------|--------------------------------------------------------------------------------------------------------------------|
|                    | Currency code               | Char(3)   |                              | but populated by Retek sales audit Currency of the given location. Not required but populated by Retek sales audit |
|                    | Currency retail decimals    | Number(1) |                              | Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit        |
| Transaction Header | File Type Record Descriptor | Char(5)   | THEAD                        | Identifies transaction record type                                                                                 |
|                    | File Line Identifier        | Char(10)  | specified by external system | ID of current line being processed by input file.                                                                  |
|                    | Transaction Date            | Char(14)  | transaction date             | date sale/return transaction was processed at the POS                                                              |
|                    | Item Type                   | Char(3)   | REF<br>ITM                   | item type will be represented as a REF or ITM                                                                      |
|                    | Item Value                  | Char(25)  | item identifier              | the id number of an ITM or REF                                                                                     |
|                    | Dept                        | Number(4) | Item's dept                  | Dept of item sold or returned. Not required but populated by Retek sales audit                                     |
|                    | Class                       | Number(4) | Item's class                 | Class of item sold or returned. Not required but populated by Retek sales audit                                    |
|                    | Subclass                    | Number(4) | Item's subclass              | Subclass of item sold or returned. Not required but populated by Retek sales audit                                 |
|                    | Pack Indicator              | Char(1)   | Item's pack indicator        | Pack indicator of item sold or returned. Not required but populated by                                             |

|                         |            |                                  |                                                                                                                                                                                  |
|-------------------------|------------|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Item level              | Number(1)  | Item's item level                | Retek sales audit<br>Item level of item sold or returned. Not required but populated by Retek sales audit                                                                        |
| Tran level              | Number(1)  | Item's tran level                | Tran level of item sold or returned. Not required but populated by Retek sales audit                                                                                             |
| Wastage Type            | Char(6)    | Item's wastage type              | Wastage type of item sold or returned. Not required but populated by Retek sales audit                                                                                           |
| Wastage Percent         | Number(12) | Item's wastage percent           | Wastage percent of item sold or returned. Not required but populated by Retek sales audit                                                                                        |
| Transaction Type        | Char(1)    | 'S' – sales<br>'R' - return      | Transaction type code to specify whether transaction is a sale or a return                                                                                                       |
| Drop Shipment Indicator | Char(1)    | 'Y'<br>'N'                       | Indicates whether the transaction is a drop shipment or not. If it is a drop shipment, indicator will be 'Y'. This field is not required, but will be defaulted to 'N' if blank. |
| Total Sales Quantity    | Number(12) |                                  | Number of units sold at a particular location with 4 implied decimal places.                                                                                                     |
| Selling UOM             | Char(4)    |                                  | UOM at which this item was sold.                                                                                                                                                 |
| Sales Sign              | Char(1)    | 'P' - positive<br>'N' - negative | Determines if the Total Sales Quantity and Total Sales Value are positive or                                                                                                     |

|                     |                             |            |                                                      |                                                                                                                    |
|---------------------|-----------------------------|------------|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
|                     | Total Sales Value           | Number(20) |                                                      | negative.<br>Sales value, net sales value of goods sold/returned with 4 implied decimal places. For VBO future use |
|                     | Last Modified Date          | Char(14)   |                                                      |                                                                                                                    |
| Transaction Detail  | File Type Record Descriptor | Char(5)    | TDETL                                                | Identifies transaction record type                                                                                 |
|                     | File Line Identifier        | Char(10)   | specified by external system                         | ID of current line being processed by input file.                                                                  |
|                     | Promotional Tran Type       | Char(6)    | promotion type – valid values see code_detail table. | code for promotional type from code_detail, code_type = 'PRMT'                                                     |
|                     | Promotion Number            | Number(10) | promotion number                                     | promotion number from the RMS                                                                                      |
|                     | Sales Quantity              | Number(12) |                                                      | number of units sold in this prom type with 4 implied decimal places.                                              |
|                     | Sales Value                 | Number(20) |                                                      | value of units sold in this prom type with 4 implied decimal places.                                               |
|                     | Discount Value              | Number(20) |                                                      | Value of discount given in this prom type with 4 implied decimal places.                                           |
| Transaction Trailer | File Type Record Descriptor | Char(5)    | TTAIL                                                | Identifies file record type                                                                                        |
|                     | File Line Identifier        | Char(10)   | specified by external system                         | ID of current line being processed by input file.                                                                  |
|                     | Transaction Count           | Number(6)  | specified by external system                         | Number of TDETL records in this transaction set                                                                    |
| File Trailer        | File Type Record            | Char(5)    | FTAIL                                                | Identifies file record type                                                                                        |



| Descriptor           |            |                              |                                                                                             |  |
|----------------------|------------|------------------------------|---------------------------------------------------------------------------------------------|--|
| File Line Identifier | Number(10) | specified by external system | ID of current line being processed by input file.                                           |  |
| File Record Counter  | Number(10) |                              | Number of records/transactions processed in current file (only records between head & tail) |  |

#### Invalid Item/Store File:

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

#### VAT File:

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

#### Reject File:

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

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

#### Error File:

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

### Technical Issues

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

## Pre-cost calculation [precostcalc]

### Design Overview

This batch module is responsible for data maintenance tasks that are necessary before running costcalc.

Unprocessed records in RECLASS\_COST\_CHG\_QUEUE drive this program. The driving cursor is a set of seven cursors whose goal is to make sure the DEAL\_SKU\_TEMP table is inserted for three scenarios:

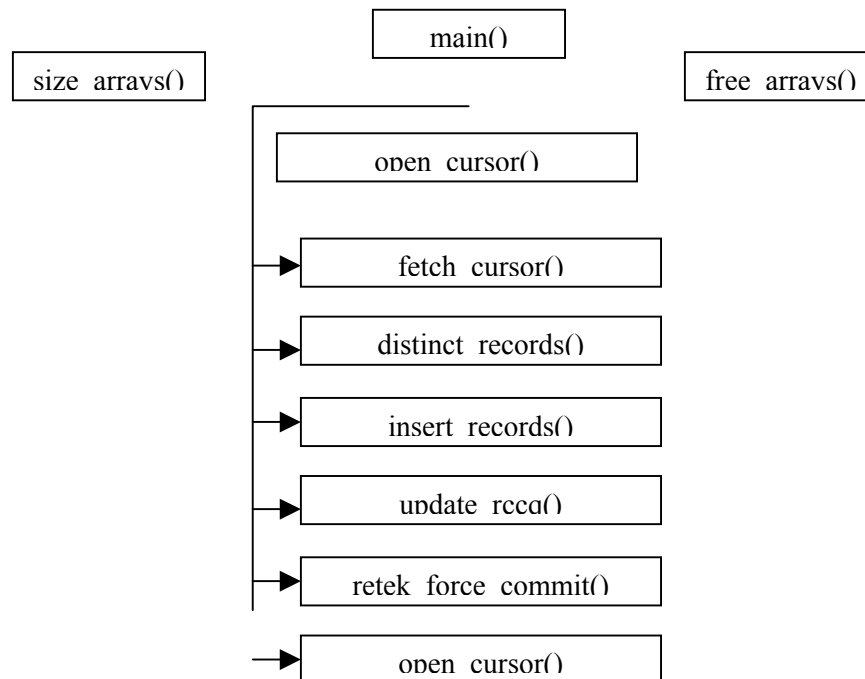
- a. If an unprocessed record exists on RECLASS\_COST\_CHG\_QUEUE, create record(s) on DEAL\_SKU\_TEMP for the RECLASS\_COST\_CHG\_QUEUE event. For reclassification events no location is given, therefore these events are blown out to all item-location locations before inserting to DEAL\_SKU\_TEMP.
- b. If an unprocessed record exists on RECLASS\_COST\_CHG\_QUEUE, create record(s) on DEAL\_SKU\_TEMP for any events on the FUTURE\_COST table that the RECLASS\_COST\_CHG\_QUEUE event affects (make the match using item, supplier, origin country id, location, and start date). For reclassification events (not cost change) no location is given and therefore these events are blown out to all item-location locations before seeking FUTURE\_COST matches.
- c. If a record exists on DEAL\_SKU\_TEMP that affects one or more FUTURE\_COST records insert for those records on FUTURE\_COST into DEAL\_SKU\_TEMP to make sure they are recalculated.

The program will also update RECLASS\_COST\_CHG\_QUEUE events that it processed so that the record's process flag reflects the fact that it has been processed. To avoid primary key violations in DEAL\_SKU\_TEMP, check if an item, location, supplier, origin country id, and start date combination exists on DEAL\_SKU\_TEMP in the driving cursors. When the program is done, it should delete any records from RECLASS\_COST\_CHG\_QUEUE that did not get processed. (This happens when somebody did a cost change or reclassified a component item of a case UPC. This record will not be picked up by the driving cursor and should not remain on RECLASS\_COST\_CHG\_QUEUE.) Also, all records that are of type G or N should also be deleted since once they have been inserted into DEAL\_SKU\_TEMP, There is no reason for them to remain in RECLASS\_COST\_CHG\_QUEUE. This should improve the performance of the program by keeping the size of RECLASS\_COST\_CHG\_QUEUE as small as possible.

The LUW of this module is a single record from any one of the driving cursors, which all pick up item/supplier/origin country/location/active date combinations from RECLASS\_COST\_CHG\_QUEUE, and FUTURE\_COST.

### Stored Procedures / Shared Modules (Maintainability)

## Program Flow



## Function Level Description

**Declare** a fetch array struct to hold array-fetched records from driving cursors.

**Declare driving cursors** The first eight cursors pick up unprocessed events from RECLASS\_COST\_CHG\_QUEUE that do not exist in DEAL\_SKU\_TEMP. There is eight of them because each handles a different type of record depending on whether location is given or not, and whether division is given or not. So, the combinations that are possible and therefore need to be fetched are:

| Location given? | Division given? |
|-----------------|-----------------|
| N               | N               |
| N               | Y               |
| Y               | N               |

Remember that the driving cursors for every one of the four above described combinations will be again split into two cursors depending on whether the location is a warehouse or a store. The reason so many driving cursors exist is because a single cursor that joined to all the tables at once was so slow that using such a cursor here was not an option. One more cursor is required: the ninth cursor will fetch records from the FUTURE\_COST table that have the same item/supplier/origin country/location combination as a DEAL\_SKU\_TEMP record with an active date that is greater than the DEAL\_SKU\_TEMP record's start date. This cursor is necessary to ensure that when an event is inserted into DEAL\_SKU\_TEMP, later events for this event in the FUTURE\_COST table may be affected by the event in DEAL\_SKU\_TEMP and should be re-inserted into DEAL\_SKU\_TEMP so costcalc can recalculate and reinsert them into FUTURE\_COST.

The cursors should not pick up any primary cost pack component items since those are never inserted into DEAL\_SKU\_TEMP. (See Design Assumptions below.) Also this cursor should never pick up items that are not approved, transaction level, or items that are buyer packs. Deals batch programs do not process those.

**Main():** Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

**Init():** Standard Retek init function. Calls retek\_init() and size\_arrays(). If this is a new start, set the restart cursor variable to 1.

**Process():** This is the main function that does all the work. It will array fetch each of the 9 cursors in the order of their definition above, one at a time until cursor is empty. For each cursor, fetched records will be array inserted into DEAL\_SKU\_TEMP, RECLASS\_COST\_CHG\_QUEUE will be updated after every second cursor fetch to set the process\_flag of covered records to 'Y' and commit logic will be executed. After the last cursor has been fetched empty and all records were processed, delete from RECLASS\_COST\_CHG\_QUEUE all records that belong to this thread and have not been picked up by any of the cursors (their process\_flag is still 'N'). Also delete any records whose rec\_type is 'G' or 'N'.

- Call open\_cursor to open the first cursor. (Cursors are opened and fetched in the order that they were declared in, which should be the same order they were described in above.)
- In a while loop (while the no records found indicator is not set):
  - Array fetch records from current driving cursor by calling fetch\_cursor().
  - Call distinct\_records() to copy distinct (item/supplier/origin country/location/start date) records from the fetch array (which has RECLASS\_COST\_CHG\_QUEUE's level of distinctness: item/supplier/origin country/location/start date/record type) to the insert array.
  - Call insert\_records() to array insert records from the insert array.
  - If we fetched empty a cursor with an even number call update\_rccq() to update RECLASS\_COST\_CHG\_QUEUE's process\_flags to 'Y' for the records covered by the last two cursors.
  - Call retek\_force\_commit() with the item, supplier and origin country, location, start date, and cursor number of the last record as the argument.
  - If the no records found indicator was set by fetch\_cursor(), increment the cursor counter. If the cursor counter is under 10, call open\_cursor() and reset the no records found indicator to 0 along with the total fetched records counter, which should be zeroed since we are about to start fetching from a new cursor. If the cursor number is 10 or above, we are done. Do not call open\_cursor() and leave the no records found indicator set. This will drop the process out of the while loop.
- Delete from RECLASS\_COST\_CHG\_QUEUE all records for this thread where the process\_flag is 'N'. (This may happen if the component item of a case UPC was inserted into RECLASS\_COST\_CHG\_QUEUE. The driving cursor will not pick these records up.) Also delete for records whose rec\_type is 'G' or 'N' regardless of their process\_flag. We do not need to keep those events around in RECLASS\_COST\_CHG\_QUEUE.

**Insert\_records():** This function performs an array insert into DEAL\_SKU\_TEMP. Its argument is the current array size (number of records fetched into array).

**Distinct\_records():** This function copies records from the fetch array (which has RECLASS\_COST\_CHG\_QUEUE's level of distinctness: item/supplier/origin country/location/start date/record type) to the insert array (which has DEAL\_SKU\_TEMP's level of distinctness: item/supplier/origin country/location/start date) so that primary key/index violation errors are avoided on subsequent inserts into DEAL\_SKU\_TEMP. This function will need local static strings to store any previous call's last copied DEAL\_SKU\_TEMP primary key so that it can be compared to the next call's first fetch array element's DEAL\_SKU\_TEMP primary key fields (item/supplier/origin country/location/start date). This is necessary if an array fetch cut the return of two identical item/supplier/origin country/location/start date records into two array blocks because of the max commit counter. We want to keep track of the fact that the item/supplier/origin country/location/start date record in question was already in the previous block. If this block still has that item/supplier/origin country/location/start date in the first field(s), don't copy it into the insert array again.

**Open\_cursor():** This function consists of a case-switch statement that depending on the argument of the function (an integer representing the number of the cursor that needs to be opened) will open the appropriate cursor.

**Fetch\_cursor():** This function consists of a case-switch statement that depending on the argument of the function (an integer representing the number of the cursor that needs to be fetched) will fetch the appropriate cursor and returns the number of records fetched, along with an indicator if the no data found indicator has been set by the fetch or not.

**Update\_rccq():** This function will be called for every other cursor that was fetched empty and it will update RECLASS\_COST\_CHG\_QUEUE records' process\_flag to 'Y' covered by the last two cursors. This strange method of updating those records was necessary for restart/recovery reasons. Since two cursors are used to fetch for a single RECLASS\_COST\_CHG\_QUEUE record (i.e.: no location or division given), one fetching for warehouse locations the other for store locations, the process flag of such a record can not be updated to Y until both cursors were fetched for this record. Otherwise the stores would not get picked up if we update the process flag right after warehouses were fetched. The update statements cover all records in RECLASS\_COST\_CHG\_QUEUE for which the most recent two cursors were fetched for, not only the ones for which the cursors actually returned records. A cursor may not return records, even though the RECLASS\_COST\_CHG\_QUEUE records were completely valid, simply because the returned values already exist in DEAL\_SKU\_TEMP and the cursors have a not exists clause.

**Size\_arrays():** Sizes the fetch array to the commit size.

**Free\_arrays():** Frees fetch array.

**Final():** Standard Retek final function. Calls free\_arrays() and retek\_close().

## Input Specifications

**Select data from:**

[illegible]

## ‘Table-To-Table’

**Insert into:**

This module must be run after ditinsrt and before costcalc in the deals batch cycle.

This module is multi-threaded by supplier. See volume-testing documentation for optimum thread value. (I suggest 15-30 threads.)

## Locking Strategy

### Restart/Recovery

This program has restart recovery based on item/supplier/origin country/location/start date/cursor number and is multi-threaded by supplier.

### Performance Considerations

The driving cursors should be small enough to be executed fast. If the DEAL\_SKU\_TEMP table holds too many records as the program runs and the cursors' NOT EXISTS statements are slowing things down because of the size of DEAL\_SKU\_TEMP (which may very well happen), the only remaining performance enhancement that could go into the program is to simply fetch records from the cursors without checking for duplicates at the time of fetch and check for duplicates at time of insert or handle primary key violations in the insert as a non-fatal error. Or fetch all the records from all the cursors and discretize the fetched records in batch based on DEAL\_SKU\_TEMP's primary key. These "solutions" may not be a performance enhancement; the second suggestion simply takes away the load of discretizing records from the database and keeps it in the batch. Therefore other programs will not suffer a performance loss from the database being slow while precostcalc runs.

### Security Considerations

N/A

### Unit Test Considerations

When program is tested, tester will probably need to run costcalc and prepost for complete results. The tables DEAL\_SKU\_TEMP and RECLASS\_COST\_CHG\_QUEUE should not be modified in any way during and between the two program runs outside the programs themselves.

See program's UTP for further instructions.

### Design Assumptions

Background: Costcalc is driven by the DEAL\_SKU\_TEMP table, which holds item/supplier/origin country/location/active date records that then need to be moved to the FUTURE\_COST table with their costs at the date specified on DEAL\_SKU\_TEMP. Costcalc simply takes these item/supplier/origin country/location/active date records, calculates the cost for this combination and inserts the result into FUTURE\_COST, along with a reset date for the item/supplier/country/location record if it has one. A reset date would be a deal closing that caused the cost change in the first place. This reset record on FUTURE\_COST is simply the item/supplier/origin country/location/close date and the cost. This design has a few inherent problems. Some deals have no close dates and item reclassifications also have no close dates but potentially may change the cost of the item since different deals may apply due to the new merchandise hierarchy classification. Therefore a record on DEAL\_SKU\_TEMP with no close date may affect records on FUTURE\_COST that have an active date later than the record on DEAL\_SKU\_TEMP. The solution is to move records from FUTURE\_COST that are potentially affected by DEAL\_SKU\_TEMP records back to DEAL\_SKU\_TEMP, thus guaranteeing that they get recalculated and the correct price will be set as they are re-inserted into FUTURE\_COST by costcalc. This process of checking for affected records and moving them into DEAL\_SKU\_TEMP is performed by precostcalc. Also a new table was created called RECLASS\_COST\_CHG\_QUEUE which holds reclassification, cost change, new item-location events for items along with a general record that simply holds an item which needs to be inserted

into DEAL\_SKU\_TEMP (if an item reclassification was cancelled, we still need to send in a record to DEAL\_SKU\_TEMP to make sure the item's record on FUTURE\_COST is re-calculated).

**Case UPCs:** Component items of case UPCs are never inserted into DEAL\_SKU\_TEMP. These items will be processed by costcalc as part of their case UPCs. Therefore if a case UPC component is inserted into RECLASS\_COST\_CHG\_QUEUE, the driving cursor should ignore it and the record should be deleted. Also an item should be approved, transaction level and not a buyer pack to qualify for insertion into DEAL\_SKU\_TEMP.

Three quick examples of what appears in RECLASS\_COST\_CHG\_QUEUE for a re-class, a cost change, a new item-location record or a general record:

**Re-classification:** I need an ITEM, -1 for LOCATION, SUPPLIER, ORIGIN\_COUNTRY\_ID, START\_DATE, DIVISION, GROUP\_NO, DEPT, CLASS, SUBCLASS, REC\_TYPE = 'R'.

**Cost change:** I need an ITEM, LOCATION, SUPPLIER, ORIGIN\_COUNTRY\_ID, START\_DATE, REC\_TYPE = 'C'. (For warehouse locations, only insert virtual warehouses!)

**New item location:** I need an ITEM, LOCATION, SUPPLIER, ORIGIN\_COUNTRY\_ID, START\_DATE, REC\_TYPE = 'N'. (For warehouse locations, only insert virtual warehouses!)

**General:** Primary key fields only. (These records appear as placeholders for cancelled events. For example, the merchandiser may cancel a future reclassification event in which case the original event's record on RECLASS\_COST\_CHG\_QUEUE would be updated to have a rec\_type of 'G' and a process\_flag of 'N'. This results in precostcalc reinserting this event into DEAL\_SKU\_TEMP, from there costcalc will re-insert the event into FUTURE\_COST. Thus external systems that exported the event earlier will see the event again in FUTURE\_COST with potentially a new cost and can export again if necessary. Once such a record has been migrated to FUTURE\_COST, there is no need to keep it in RECLASS\_COST\_CHG\_QUEUE too. It may be deleted. Same is true for records with a rec\_type 'N'.



## Pre- and post-functions for programs [prepost]

### Design Overview

The Pre/Post module facilitates multi-threading by allowing general system administration functions (such as table deletions or mass updates) to be completed after all threads of a particular program have been processed. A brief description of all pre- or post-processing functions included in this program can be found in the Function-Level Description section.

This program will take three parameters: username/password to log on to Oracle, a program before or after which this script must run and an indicator telling whether the script is a pre or post function. It will act as a shell script for running all pre-program and post-program updates and purges (the logic was removed from the programs themselves to enable multi-threading & restart/recovery).

For example, to run the pre-program script for the ctext program, the following should be entered on the command line:

```
prepost user/password rpl pre
```

Tables affected: (this is not complete but includes mods made for SIR 36126)

| TABLE                    | SELECT | INSERT | UPDATE | INDEX | DELETE | TRUNCATE | TRIGGER |
|--------------------------|--------|--------|--------|-------|--------|----------|---------|
| alloc_detail             |        |        |        |       |        | No       | Yes     |
| alloc_detail             |        |        |        |       |        | No       | Yes     |
| alloc_header             |        |        |        |       |        | No       | Yes     |
| alloc_header             |        |        |        |       |        | No       | Yes     |
| class_sales_forecast     |        |        |        | Yes   |        | Yes      |         |
| class_sales_forecast     |        |        |        | Yes   |        | Yes      |         |
| class_stock_on_hand      |        |        |        | Yes   |        | Yes      |         |
| cost_change_trigger_temp |        |        |        | Yes   |        | Yes      |         |
| cost_susp                | No     | No     | No     |       | Yes    |          |         |
| cost_susp_head           | No     | No     | Yes    |       | No     |          |         |
| deal_sku_temp            |        |        |        | Yes   |        | Yes      |         |
| dept_sales_forecast      |        |        |        | Yes   |        | Yes      |         |
| dept_stock_on_hand       |        |        |        | Yes   |        | Yes      |         |
| edi_ord_temp             |        |        |        | Yes   |        | Yes      |         |
| fif_receiving            |        |        |        | Yes   |        | Yes      |         |
| forecast_rebuild         |        |        |        | Yes   |        | Yes      |         |
| hist_rebuild_mask        |        |        |        | Yes   |        | Yes      |         |
| if_tran_data             |        |        |        | Yes   |        | Yes      |         |
| item_forecast            |        |        |        | Yes   |        | Yes      |         |
| mc_rejections            |        |        |        | Yes   |        | Yes      |         |
| mod_order_item_hts       | No     | No     | No     | Yes   | Yes    | Yes      |         |
| on_order_temp            | No     | No     | No     | Yes   | Yes    | Yes      |         |
| ord_missed               |        |        |        | Yes   |        | Yes      |         |
| ord_temp                 |        |        |        | Yes   |        | Yes      |         |
| ordhead                  | No     | No     | Yes    |       | No     | No       | Yes     |
| ordloc                   |        |        |        |       |        | No       | Yes     |
| ordsku                   |        |        |        |       |        | No       | Yes     |
| period                   | Yes    | No     | No     |       | No     |          |         |
| pos_mods                 | No     | No     | No     | Yes   | Yes    | Yes      |         |
| price_conflict           |        |        |        | Yes   |        | Yes      |         |
| reclass_item             |        |        |        | Yes   |        | Yes      |         |
| reclass_trigger_temp     |        |        |        | Yes   |        | Yes      |         |
| repl_attr_update_item    |        |        |        | Yes   |        | Yes      |         |
| repl_attr_update_loc     |        |        |        | Yes   |        | Yes      |         |
| repl_item_loc_updates    |        |        |        | Yes   |        | Yes      |         |
| req                      | Yes    | No     | No     |       | Yes    |          |         |
| reqdetail                | No     | No     | No     |       | Yes    |          |         |
| sec_user_loc_matrix      |        |        |        | Yes   |        | Yes      |         |
| sec_user_prod_matrix     |        |        |        | Yes   |        | Yes      |         |
| sec_user_zone_matrix     |        |        |        | Yes   |        | Yes      |         |
| sim_expl                 |        |        |        | Yes   |        | Yes      |         |
| subclass_sales_forecast  |        |        |        | Yes   |        | Yes      |         |
| subclass_stock_on_hand   |        |        |        | Yes   |        | Yes      |         |
| system_options           | Yes    | No     | No     | No    | No     |          |         |
| system_variables         | Yes    | No     | Yes    | No    | No     |          |         |
| tif_explode              |        |        |        | Yes   |        | Yes      |         |

**Scheduling Constraints**

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| Processing Cycle:   | PHASE ALL (daily)                                                                  |
| Scheduling Diagram: | See scheduling flow for description of all pre-post requirements in the daily run. |
| Pre-Processing:     | N/A                                                                                |
| Post-Processing:    | N/A                                                                                |
| Threading Scheme:   | N/A (single threaded)                                                              |

**Restart Recovery**

N/A

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

Functions to be used by the individual program functions:

`modify_indexes()`:

- Allows indexes to be disabled or rebuilt before and/or after the action that affects them. The individual program passes in the table name and mode (what action to take “disable” or “rebuild” ) and performs that action. The owner of the index is determined using the `synonym_trace` function in the library `oracle.pc`.

`modify_partition_indexes()`:

- Called by the `modify_indexes` function to determine if the indexes that need modified are partitioned indexes. If so, then the statement is modified to take that into account to accomplish the action. `Index_owner`, `index_name` and `mode` is passed to this function. Nothing is passed back out.

`truncate_table()`:

- The `table_name` is passed to this function so that it can be truncated. The owner of the table is determined by using the `synonym_trace` function in the library `oracle.pc`.

`modify_trigger()`:

- Allows triggers to be disabled or enabled before or after certain processes. The `table_name`, trigger name and `mode`( “DISABLE” or “ENABLE”) are passed to this function and the appropriate action is taken. No values are passed back to the calling function.

`policy_enable()`:

- enables or disables policies

`truncate_user_sec_table()`

- truncate the user security tables and prepare for the user product security rebuild, user location security rebuild or user zone security rebuild.
- This is a function used to run the sprdrld, slocrbld and szonrbld pre functions that will truncate the sec\_user\_prod\_matrix, sec\_user\_loc\_matrix and sec\_user\_zone\_matrix tables respectively.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

update\_security\_indicator():

- reset the update\_sec\_XXX\_ind to 'N' on the system\_variables table where xxx represents prd, loc or zon if it is user product security rebuild, user location security rebuild or user zone security rebuild, respectively. Running the sprdrld, slocrbld and szonrbld post programs will use this function.

### Individual Program Functions

salweek\_post():

- Updates the last end-of-week date on the SYSTEM\_VARIABLES table to today's date after all weekly stock ledger data has been processed.

salmth\_post():

- Updates the following SYSTEM\_VARIABLES columns to reflect the current date's values after all monthly stock ledger data has been processed:
  - last\_eom\_half\_no
  - last\_eom\_month\_no
  - last\_eom\_date
  - next\_eom\_date
  - last\_eom\_start\_half
  - last\_eom\_end\_half
  - last\_eom\_start\_month
  - last\_eom\_mid\_month
  - last\_eom\_next\_half\_no
  - last\_eom\_day
  - last\_eom\_week
  - last\_eom\_month
  - last\_eom\_year
  - last\_eom\_week\_in\_half

rpl\_pre():

- Truncates the following tables before replenishment extracts are performed.
  - ORD\_TEMP
  - ORD\_MISSED
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

pcovrl\_pre():

- Truncates the PRICE\_CONFLICT table before new conflicts are found and written.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

rplatupd\_pre():

- Truncates the MC\_REJECTIONS table so that it is free to hold new mass change rejections.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

rilmaint\_post():

- Truncates the REPL\_ITEM\_LOC\_UPDATES table after these records are processed so the table is free to hold new updates.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

rplatupd\_post():

- Truncates the holding tables REPL\_ATTR\_UPDATE\_ITEM and REPL\_ATTR\_UPDATE\_LOC after their records have been processed.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

pctranex\_post():

- Deletes from the table PRICE\_BATCH\_TRAN after the price changes it held have been processed.

supmth\_post():

- Truncates the table SUP\_DATA after all daily supplier data records have been rolled up to month level.

sccext\_post():

- Updates all processed supplier cost change record statuses to 'Extracted'.

rplbld\_pre():

- Updates replenishment order origins from 'Current System Generated' to 'Past System Generated' before processing. Enables the following triggers:  
RMS\_TABLE\_OHE\_AIUR  
RMS\_COL\_OHE\_STATUS\_AUR  
RMS\_TABLE\_ALH\_AIDR  
RMS\_TABLE\_OSK\_AIUR  
RMS\_COL\_ALD\_QA\_AUR

RMS\_TABLE\_ALD\_AIUDR  
 RMS\_TABLE\_OLO\_AUR  
 RMS\_COL\_OLO\_QTYORDERED\_AUR

rplapprv\_post():

- Disables the following triggers:  
 RMS\_TABLE\_OHE\_AIUR  
 RMS\_COL\_OHE\_STATUS\_AUR  
 RMS\_TABLE\_ALH\_AIDR  
 RMS\_TABLE\_OSK\_AIUR  
 RMS\_COL\_ALD\_QA\_AUR  
 RMS\_TABLE\_ALD\_AIUDR  
 RMS\_TABLE\_OLO\_AUR  
 RMS\_COL\_OLO\_QTYORDERED\_AUR

hstbld\_post():

- Truncates the holding table MASK\_REBUILD after building history records.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

poscdnld\_post():

- Clears the config\_status and loc\_grp\_status in POS\_LOC\_GRP. Sets all values of extract\_req\_ind to 'N'.
- Clears the status column in POS\_MERCH\_CRITERIA.
- Sets the status\_ind column in POS\_STORE to 'N'.

posdnld\_post():

- Clears the POS\_MODS table after all records have been downloaded to the POS.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

prmext\_post():

- Updates the promotion item status on the PROMSKU table, setting it to NULL if the promotion item has been extracted successfully or to 'Delete Processed' if the item was marked for deletion before extraction.

edidlprd\_post():

**THIS FUNCTION SHOULD BE RUN AFTER THE edidlprd PROGRAM!**

It needs to be run as follows from the command line:

prepost username/password **edidlprd** post

- Deletes old records from the EDI\_DAILY\_SALES table after they have been processed.

fcstrbld\_post():

- Truncates the holding table FORECAST\_REBUILD after all records have been processed.

- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

vrplbld\_post():

- Truncates the EDI\_ORD\_TEMP table after all replenishment orders have been build from the data held there.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

cntrordb\_post():

- Set the last\_cont\_order\_date on system\_variables to vdate.

fifgldn1\_post()

- If Oracle Financials is being used, delete everything from the fif\_receiving table and repopulate it from the if\_tran\_data table
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

fsadnld\_post():

- Updates the load\_sales\_ind to 'N' for all records on the appropriate domain table – domain\_dept, domain\_class, or domain\_subclass, where system\_options.domain\_level = 'D', 'C', or 'S', respectively.

whstrasg\_post():

- deletes all warehouse store assignment records from the warehouse store assignment table if the assignment date (wh\_store\_assign.assign\_date) is less than or equal to the current date (period.vdate) minus the warehouse store assignment history days (system\_options.wh\_store\_assign\_hist\_days).

costcalc\_post():

- Truncates the deal\_sku\_temp table.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

tifposdn\_post()

- truncate tif\_explode table
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.

`htsupld_pre()`

- Truncates the `mod_order_item_hts` table so that reports will be correct and not include data from previous runs of `htsupld`.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

`onordext_pre()`

- Truncates the `on_order_temp` table.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

`precostcalc_pre()`

- Processes records from the `COST_CHANGE_TRIGGER_TEMP` and `RECLASS_TRIGGER_TEMP` tables.
- `Reclass_trigger_temp` is populated only by database trigger and `cost_change_trigger_temp` is populated by database trigger and `edi_cost_change_sql.create_cost_chg`.
- This function will either insert new records or update existing ones on `reclass_cost_chg_queue`.
- Both tables, `COST_CHANGE_TRIGGER_TEMP` and `RECLASS_TRIGGER_TEMP` are truncated and their indexes rebuilt at the end of this function.
- The user running this program for this function must have been granted the 'drop any table' and 'alter any index' system privilege, or be the owning schema user.

`reclsdly_pre()`

- Disables the trigger `RMS_TABLE_RCS_BIDR` on the `reclass_item` table.
  - The user running this program for this function must have been granted the 'alter any trigger' system privilege, or be the owning schema user.

`reclsdly_post()`

- Enables the trigger `RMS_TABLE_RCS_BIDR` on the `reclass_item` table.
- The user running this program for this function must have been granted the 'alter any trigger' system privilege, or be the owning schema user.

`ibcalc_pre()`

- Updates the status on `ib_results` to 'U' nprocessed where the status = 'W' orksheet so after `ibcalc` is run, multiple records in 'W' orksheet status will not exist for each item/location.

`fcstprg_pre()`

- Disables any indexes prior to the truncation on following tables. This is run BEFORE the `fcstprg.pc` program on **PARTITIONED TABLES** only.
  - `ITEM_FORECAST`



- DEPT\_SALES\_FORECAST
- CLASS\_SALES\_FORECAST
- SUBCLASS\_SALES\_FORECAST
- The user running this program for this function must have been granted the ‘alter any index’ system privilege, or be the owning schema user.

#### fcstprg\_post()

- Rebuilds the indexes following truncation of following tables.
  - ITEM\_FORECAST
  - DEPT\_SALES\_FORECAST
  - CLASS\_SALES\_FORECAST
  - SUBCLASS\_SALES\_FORECAST
- The user running this program for this function must have been granted the ‘alter any index’ system privilege, or be the owning schema user.

#### Sohrlup\_pre()

- Truncates the subclass\_stock\_on\_hand, class\_stock\_on\_hand, and dept\_stock\_on\_hand tables.
- Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.
- The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

#### I/O Specification

N/A

#### Technical Issues

N/A

## Promotion impact [primpdep]

### Design Overview

The promotion department impact batch program, primpdep.pc, is one of two new batch programs that will calculate the estimated impact for a proposed promotion. Primpdep will call a package for each promotion on promhead where impact\_date is not null. The package called in the program, prm\_impact\_calc.build\_dept(), will generate the department-level impact analysis.

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: This process may be run on-demand, but should be scheduled at a minimum once per day.

Scheduling Diagram: primpdku.pc → (must come before) → primpdep.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: promotion

### Restart Recovery

Logical Unit of Work ( recommended Commit check points )

Driving Cursor

```
SELECT p.promotion
 FROM promhead p,
 v_restart_promotion rv
 WHERE p.impact_date <= to_date(:ps_vdate, 'YYYYMMDD')
 AND p.status is NOT NULL
 AND rv.driver_value = p.promotion
 AND rv.driver_name = :ps_restart_driver_name
 AND rv.num_threads = :ps_restart_num_threads
 AND rv.thread_val = :ps_restart_thread_val
 AND p.promotion > NVL(:ps_restart_promotion, -999)
 ORDER BY p.promotion;
```

Note: The view, v\_restart\_promotion, does not exist yet and must be created.

### Program Flow

Structure Chart

### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage  
**Prm\_impact\_calc.build\_dept()** – New function for calculating the DEPT impact of a proposed promotion.

**Function Level Description**

All database interactions required and error handling considerations

**Init()** – Size variables in the fetch for the driving cursor, initialize restart recovery.

**Process()** – Fetch from the driving cursor. For each id, call PRM\_IMPACT\_CALC.BUILD\_DEPT(). Pass the promotion number and pass the force\_recalc\_ind = 'N'. Set the promhead.impact\_date for the current promotion to null so it will not be picked up during the next batch cycle. After processing array, call restart\_force\_commit().

**Final()** – Call restart\_close().

**I/O Specification**

All files layouts input and output

**Technical Issues**

1. New view should be created for promotions.

## Promotion impact item [primpsku]

### Design Overview

The promotion SKU impact batch program is one of two new batch programs that will calculate the estimated impact for a proposed promotion. Primpsku will call a package for each promotion on promhead where impact\_date is not null. The package called in the program, prm\_impact\_calc.build\_sku, will generate the item-zone level impact analysis..

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: This process may be run on-demand, but should be scheduled at a minimum once per day.

Scheduling Diagram: primpsku.pc → (must come before) → primpdep.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: promotion

### Restart Recovery

Logical Unit of Work ( recommended Commit check points )

Driving Cursor

```
SELECT p.promotion
 FROM promhead p,
 v_restart_promotion rv
 WHERE p.impact_date <= to_date(:ps_vdate, 'YYYYMMDD')
 AND p.status is NOT NULL
 AND rv.driver_value = p.promotion
 AND rv.driver_name = :ps_restart_driver_name
 AND rv.num_threads = :ps_restart_num_threads
 AND rv.thread_val = :ps_restart_thread_val
 AND p.promotion > NVL(:ps_restart_promotion, -999)
 ORDER BY p.promotion;
```

Note: The view, v\_restart\_promotion, does not exist yet and must be created.

### Program Flow

Structure Chart

### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage  
**Prm\_impact\_calc.build\_sku()** – New function for calculating the SKU impact of a proposed promotion

**Function Level Description**

All database interactions required and error handling considerations

**Init()** – Size variables in the fetch for the driving cursor, initialize restart recovery.

**Process()** – Call PRM\_IMPACT\_CALC.BUILD\_SKU() for each promotion returned by the driving cursor. After processing array, call retek\_force\_commit().

**Final()** – Call retek\_close().

**I/O Specification**

All files layouts input and output

**Technical Issues**

1. New view should be created for promotions.

## Promotion price extract and download [prmext]

### Design Overview

The prmext program extracts promotions from the promotion master tables within the Retek system and sends promotion price details to the point of sale system. Additionally, promotional price history is stored for each valid item/store combination. When a store is within the POS threshold extraction date, all items on the promotion will be extracted to the store (provided the item is stocked at the store). Additionally, the prmext program has the ability to extract promotion item changes throughout the life of a promotion based on the promsku status. The promsku status will provide an indication of item details that have not been extracted. (Possible changes are new item / deleted item / changed promotion price.) The item will be re-extracted to stores that are currently active with the given promotion. Stores that are to be extracted will not differentiate between promotion item changes since all items will be extracted to the POS, provided the item has not been deleted from the promotion.

The promotional retail is stored in the history tables in the local currency of the store and this is also the price that is transmitted to POS. Since the extraction process is performed at a promotion store level, this allows different stores to be effectively on promotion for varying time frames. If the promotion start is within pos\_extract\_days from tomorrow, the promotion store will be extracted to the point of sale, and a price history record is written with a future action date. The status of the promotion store is updated to extracted. Once all stores on a promotion have been extracted the overall status of the promotion on the header is set to extracted. If the promotion store end date is within pos\_extract\_days from tomorrow, the regular price will be extracted to the POS and the promotion store status will be updated to completed. Once all stores have completed, the overall promotion header status is updated to completed.

For each promotion which is due to start or end within the number of days in pos\_extract\_days from UNIT\_OPTIONS, a POS\_MODS row is built containing the details necessary for the POS PLU update for each item included in the promotion. Note that if a parent item has been included in a promotion, it must be expanded to its transaction-level children. If the promotion has any diffs, the children must have the same diffs in order to be included.

This program checks overlap with price changes if an overlap is found, it does not insert price\_hist, but still inserts pos\_mods, with the price fetched from the price change tables.

| TABLE                      | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------------------|-------|--------|--------|--------|--------|
| UNIT_OPTIONS               | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS             | No    | Yes    | No     | No     | No     |
| PERIOD                     | No    | Yes    | No     | No     | No     |
| CODE_DETAIL                | No    | Yes    | No     | No     | No     |
| PROMHEAD                   | Yes   | Yes    | No     | Yes    | No     |
| PROMSTORE                  | Yes   | Yes    | No     | Yes    | No     |
| PROMSKU                    | Yes   | No     | No     | No     | No     |
| ITEM_MASTER                | Yes   | Yes    | No     | No     | No     |
| ITEM_LOC                   | No    | Yes    | No     | No     | No     |
| ITEM_ZONE_PRICE            | No    | Yes    | No     | No     | No     |
| PRICE_ZONE_GRO<br>UP_STORE | No    | Yes    | No     | No     | No     |

|                          |    |     |     |    |    |
|--------------------------|----|-----|-----|----|----|
| PROM_MIX_MATC<br>H HEAD  | No | Yes | No  | No | No |
| PROM_MIX_MATC<br>H BUY   | No | Yes | No  | No | No |
| PROM_MIX_MATC<br>H GET   | No | Yes | No  | No | No |
| PROM_THRESHOLD<br>HEAD   | No | Yes | No  | No | No |
| PROM_THRESHOLD<br>DETAIL | No | Yes | No  | No | No |
| PROM_THRESHOLD<br>SKU    | No | Yes | No  | No | No |
| PROM_THRESHOLD<br>-DEPT  | No | Yes | No  | No | No |
| POS_MODS                 | No | No  | Yes | No | No |
| PRICE_HIST               | No | No  | Yes | No | No |
| POS_PROM_DETAIL          | No | No  | Yes | No | No |
| V_RESTART_STORE          | No | Yes | No  | No | No |

### Restart Recovery

```

SELECT ph.promotion,
 ph.event,
 ph.currency_code,
 TO_CHAR(ps.start_date, 'YYYYMMDDHH24MISS'),
 TO_CHAR(ps.end_date, 'YYYYMMDDHH24MISS'),
 TO_CHAR(ph.start_time, 'YYYYMMDDHH24MISS'),
 TO_CHAR(ph.end_time, 'YYYYMMDDHH24MISS'),
 ps.store,
 ph.status,
 DECODE(ps.extract_status, 'E', 1, 0),
 DECODE(ps.extract_status, NULL, 1, 0),
 ph.rowid,
 ps.rowid,
 ';' || ph.promotion ||
 ';' || ps.store
FROM v_restart_store rv,
 promstore ps,
 promhead ph
WHERE ph.promotion = ps.promotion
 AND ph.status in ('A', 'E') /* all approved promotions */
 AND (
 (
 ps.start_date <= to_date(:ps_extract_date, 'YYYYMMDDHH24MISS')
 AND nvl(ps.extract_status, 'E') = 'E' /* all active promotion stores */
)
 OR
 (
 ps.end_date <= to_date(:ps_extract_date, 'YYYYMMDDHH24MISS')
 AND (ps.extract_status = 'E' /* extracted promotions or 1 day promos */

```

```

 OR (ps.extract_status is NULL
 AND ps.end_date = ps.start_date))
)
)
AND rv.driver_value = ps.store
AND rv.driver_name = :ps_restart_driver_name
AND rv.num_threads = :pi_num_threads
AND rv.thread_val = :pi_thread_val
AND (ps.promotion > NVL(:ps_restart_promotion, -999) OR
 ((ps.promotion = :ps_restart_promotion) AND
 (ps.store >= :ps_restart_store)))
ORDER BY 1,8;

```

### Scheduling Constraints

Processing Cycle: PHASE 1 – (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Post processing for prmxext resets all promotion item status after the item has been extracted (or re-extracted) to a store. Since stores have various start and stop dates and promotion maintenance can occur at any point during a promotion, the prepost program resets the change status on promsku to ensure that the same change is not forwarded down to POS more than once.

Threading Scheme: STORE  
V\_restart\_store

### Program Flow

N/A

### Shared Modules

```

CURRENCY_SQL.CONVERT_BY_LOCATION
ITEM_ATTRIB_SQL.GET_STANDARD_UOM
PRICING_ATTRIB_SQL.GET_CLEAR_RETAIL
PRICING_ATTRIB_SQL.GET_RETAIL
PROMOTION_ATTRIB_SQL.GET_PROMO_RETAIL
SUPP_ITEM_SQL.GET_PRI_SUP_COST
UOM_SQL.CONVERT

```

### Function Level Description

N/A

### I/O Specification

N/A



**Technical Issues**

N/A

**Other**

The price\_hist.tran\_type contains a code number that indicates the type of transaction that caused the price change. Valid values are:

- 0 = New item added
- 2 = Unit cost was changed
- 4 = Single unit retail was changed
- 8 = Single unit retail was changed in Clearance
- 9 = Single unit retail was changed in Promotion
- 10 = Multi-unit retail was changed
- 11 = Single-unit retail and Multi-unit retail were changed
- 99 = Item was deleted from file

The pos\_mods.tran\_type contains a code number that indicates the type of transaction that caused the price change. Valid values are:

- 31 = Promotion Item - Start Maintenance
- 32 = Promotion Item - End Maintenance

## Promotion price update [prmpcupd]

### Design Overview

This new program will update item\_loc table with promotion price information. It will update the promotion fields, promo\_retail, promo\_selling\_retail and promo\_selling\_uom, in the item\_loc table with promotion price information when a simple promotion applies the item/location combination. It will also update these promotion fields to null when a promotion ends today. In addition, it will update the item\_loc table with any promotion changes, including add promotion items to the extracted promotions, delete promotion items from the extracted promotions and make promotion price changes to the extracted promotions.

This program will run daily in nightly batch cycle and should be run after the prmxext.pc.

| TABLE       | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-------------|-------|--------|--------|--------|--------|
| promhead    | No    | Yes    | No     | No     | No     |
| promstore   | No    | Yes    | No     | No     | No     |
| Promsku     | No    | Yes    | No     | No     | No     |
| Item_loc    | No    | No     | No     | Yes    | No     |
| period      | No    | Yes    | No     | No     | No     |
| dual        | No    | Yes    | No     | No     | No     |
| Item_master | No    | Yes    | No     | No     | No     |

### Stored Procedures / Shared Modules (Maintainability)

PROMOTION\_ATTRIB\_SQL.GET\_PROMO\_RETAIL – Returns the promotion retail price for the item/location.

UOM\_SQL.CONVERT – converts the values between two UOMs.

### Function Level Description

- Define a structure that will be used to define the driving cursor array.
- Define another structure to be used to define the update array.

#### Main():

This function should follow standard Retek main function format. It should call init(), process() and final() function.

#### Init():

This function performs preliminary processing and populates global variables. It will call retek\_init function to handle the restart recovery logic and bring back the bookmark string in case of a restart. It will also retrieve the date of today (vdate) and tomorrow (vdate + 1) to be used in retrieving the valid promotions to process.

#### Process():

This function will select the promotion, promotion store, store promotion start date, store promotion end date, promotion currency code, promotion item, standard unit of measure, promotion item status, promotion item price change type, amount, selling unit of measure, adjust type and the price ends in from the PROMSTORE, PROMHEAD, PROMSKU and ITEM\_MASTER table. It will then call process\_prom\_item\_loc to check the item/location relation and retrieve the promotion price. It will also call the update\_item\_loc function to update the item\_loc table with the promotion price. The driving cursor should be similar as follows:

```

SELECT ph.promotion,
 ph.currency_code,
 TO_CHAR(ps.start_date, 'YYYYMMDD'),
 TO_CHAR(ps.end_date, 'YYYYMMDD'),
 ps.store,
 im.item,
 im.standard_uom,
 psku.status,
 psku.change_type,
 NVL(psku.change_amt,0),
 NVL(psku.selling_uom,''),
 psku.adjust_type,
 NVL(psku.ends_in,0)
FROM v_restart_store rv,
 promstore ps,
 promhead ph,
 promsku psku,
 Item_master im
WHERE ph.promotion = ps.promotion
 AND ph.status in ('E', 'M') /* all extracted promotions */
 AND ps.extract_status in ('E', 'M') /* all extracted promotion stores */
 AND psku.promotion = ps.promotion
 AND ps.start_date <= :tomorrow
 AND ps.end_date >= :today
 AND ps.promotion = psku.promotion
 AND psku.change_type != 'EX'
 AND im.item_level <= im.tran_level
 AND (im.item = psku.item
 OR (im.item_parent = psku.item
 AND (psku.diff_id is null
 OR (psku.diff_id is not null
 And (psku.diff_id = im.diff_1
 OR psku.diff_id = im.diff_2))))))
 OR (im.item_grandparent = psku.item
 AND (psku.diff_id is null
 OR (psku.diff_id is not null
 AND (psku.diff_id = im.diff_1
 OR psku.diff_id = im.diff_2))))))
 AND rv.driver_value = ps.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 (ps.promotion > NVL(:ora_restart_promotion, -999) OR
 ((ps.promotion = :ora_restart_promotion) AND
 (ps.store >= :ora_restart_store)))
ORDER BY 1,5;

```

The flow of this function should be as follows:

- Define an array, la\_prom\_store, to hold the data fetched by the driving cursor.
- Define another array, la\_item\_loc, to hold the promotion data to be updated to the item\_loc table.

- Call function `size_prom_array()`.
- Call function `size_item_loc_array()`.
- Open the driving cursor
- Array fetch the driving cursor to the `la_prom_store` array for `commit_max_ctr` records.
- In a for loop, loop through each record in the `la_prom_store` array
  - Call function `process_prom_item_loc()`. Pass in all the elements in current record to the function, as well as the `item_loc` array.
  - If the current promotion/store combination is different from last one, and the count of the `item_loc` array is greater than zero, call `update_item_loc()` to update the `item_loc` table. Pass in the `item_loc` array and the count of the records in the array. Note the count needs to be reset after the update.
- Commit and restart/recovery logic.
- Remember to update `item_loc` table with the last set of records in the `item_loc` array.

### **Size\_prom\_array():**

This function will allocate memory for the array `la_prom_store` to size of `commit_max_ctr`.

### **Size\_item\_loc\_array():**

This function will allocate memory for the `item_loc` array to size of `commit_max_ctr`.

### **Process\_prom\_item\_loc():**

This function should process as follows:

- Define local variables to hold the `promo_retail`, `promo_selling_retail`, `promo_selling_uom`, and `rowid`. Initialize these variables to null.
- Create a cursor `c_item_loc` to retrieve the `promo_retail`, `promo_selling_retail`, `promo_selling_uom` and `rowid` from `item_loc` where status is 'A'ctive and the item, location match the passed in item and location.
- If no record found, return true. If error found, return fatal.
- If the end of store promotion date is today, call `populate_item_loc_array` function. Pass in the `item_loc` array and the local variables `item`, `location`, `promo_retail`, `promo_selling_retail`, `promo_selling_uom` and `rowid`. Return whatever is returned from the `populate_item_loc` function.
- Check the store Promotion start date:
  - If the start date is less than or equal to tomorrow, and the `promsku.status` is 'DI' (Deleted Item) or 'DP' (Delete Processed), check
    - if the `promo_selling_unit_retail` is null. If it is null, stop further processing and the function should return true.
    - if the `promo_selling_unit_retail` is not null, call `populate_item_loc_array` function. Pass in the `item_loc` array and the local variables `item`, `location`, `promo_retail`, `promo_selling_retail`, `promo_selling_uom` and `rowid`. Return whatever is returned from the `populate_item_loc` function.
  - Otherwise,
    - Call `get_prom_retail()` to retrieve the promotion price. Pass in the current records in the `promo_store` array and the local variable `promo_selling_retail`. Return false if the function call failed.
    - Compare the `promo_selling_retail` and the `promo_selling_uom` obtained in the `get_prom_retail` with the `promo_selling_retail` and `promo_uom` in

the item\_loc table (retrieved from cursor c\_item\_loc). If they are same, stop further processing and return true.

- Call calc\_std\_retail() to convert the promotion retail to standard retail. Pass in the selling\_uom, standard\_uom, promo\_selling\_retail, promo\_retail, item and location. Return false if the function call failed.
- Call populate\_item\_loc\_array(). Pass in the item\_loc array and the local variables item, location, promo\_retail, promo\_selling\_retail, promo\_selling\_uom and rowid. Return whatever is returned from the populate\_item\_loc function.

#### **Populate\_item\_loc\_array():**

This function will process as follows:

- Check if the count of the total records in the item\_loc array plus 1 is greater than the size of the item\_loc array. If it is exceed the array size, call resize\_item\_loc array().
- Copy the promo\_retail, promo\_selling\_retail, promo\_selling\_uom and rowid to the item\_loc array. Populate the item\_loc array's last\_update\_datetime and last\_update\_id with the SYSDATE and UESER, respectively.
- Return ture.

#### **Resize\_item\_loc\_array():**

This function will allocate additional max\_commit\_ctr memory for the item\_loc array.

#### **Get\_prom\_retail():**

This function will call stored procedure PROMOTION\_ATTRIB\_SQL.GET\_PROMO\_RETAIL to retrieve the promo\_selling\_retail.

#### **Calc\_std\_retail():**

This function will first retrieve the convert factor from a selling\_uom to standard\_uom. Then calculate the promotion retail per standard unit of measure. It should process as the steps descript below:

- Call stored procedure UOM\_SQL.CONVERT to find the convert factor. Pass 1 to the from\_value, pass selling\_uom to the from\_uom and pass standard\_uom to the to\_uom.
- Set promo\_retail = promo\_selling\_retail / convert factor.

#### **Update\_item\_loc():**

This function will do an array update against the item\_loc table using the item\_loc array passed where the records rowid in the item\_loc table equal the rowids in the item\_loc array.

#### **Final():**

This function will call retek\_close to perform the restart/recovery closing logic, as well as the last commit of the database changes.

### **Scheduling Considerations**

Processing Cycle: Phase 3

Pro-Processing: Prmext.pc

Post-Processing:

Threading Scheme: store

**Restart/Recovery**

The logic unit of work is promotion/location.

**Design Assumptions**

This program will update the item\_loc table for simple promotions, no matter if the system\_options.multi\_promo\_ind flag is on or off.

When a promotion is changed, for example an item is added to the promotion, the promotion item status will be updated or removed after the execution of prmext.pc. In order to catch the promotion changes, this program will do a full scan for all valid promotions, and update the item\_loc table with new promotion prices and the changed promotion/item/location prices.

## Promotions purge [prmprg]

### Design Overview

All old promotions are purged when the number of months since the end\_date of the promotion is greater than prom\_history\_months from UNIT\_OPTIONS. Also, any promotion that is in deleted 'D' or canceled 'C' status will be purged from the system. Rejected promotions that remain in the system beyond the threshold number of promotion reject days will also be purged.

The following database tables are accessed by this program:

| TABLE               | SELECT | INSERT | UPDATE | DELETE |
|---------------------|--------|--------|--------|--------|
| UNIT_OPTIONS        | Yes    | No     | No     | No     |
| PERIOD              | Yes    | No     | No     | No     |
| PROMHEAD            | No     | No     | No     | Yes    |
| PROMSTORE           | No     | No     | No     | Yes    |
| PROMSKU             | No     | No     | No     | Yes    |
| PROM_THRESHOLD_HEAD | No     | No     | No     | Yes    |
| PROM_THRESHOLD_DEPT | No     | No     | No     | Yes    |
| PROM_THRESHOLD_SKU  | No     | No     | No     | Yes    |
| PROM_MIX_MATCH_HEAD | No     | No     | No     | Yes    |
| PROM_MIX_MATCH_BUY  | No     | No     | No     | Yes    |
| PROM_MIX_MATCH_GET  | No     | No     | No     | Yes    |
| RESTART_CONTROL     | Yes    | No     | No     | No     |
| SA_TRAN_DISC_REV    | No     | No     | No     | Yes    |
| SA_TRAN_DISC        | No     | No     | No     | Yes    |
| FIXED_DEAL          | No     | No     | No     | Yes    |
| FIXED_DEAL_DATES    | No     | No     | No     | Yes    |
| PROMSKU_IMPACT      | No     | No     | No     | Yes    |
| PROMDEPT_IMPACT     | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: PHASE 4 (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

```
SELECT p.promotion
FROM promhead p
```

```
WHERE (:prom_history_months <
 (MONTHS_BETWEEN(to_date(:vdate),p.end_date))
 AND p.status = 'M')
 OR p.status in ('D', 'C')
 OR ((p.start_date - :prom_reject_days) > to_date(:vdate)
 AND p.status = 'R');
```

**Program Flow**

N/A

**Shared Modules**

N/A

**Technical Issues**

N/A



## Item level promotions explode [prmxpld]

### Design Overview

The promotion explode module expands records from the promdept table to the promsku table. The program will be run prior to the prmext module that will extract records from the promsku table to be written to price\_hist and pos\_mods. Once a promotion has been expanded to item level, changes will not be allowed at the department level. Since individual items could be added, or changed after expansion and due to complications that could arise due to reclassification, maintenance functions will only be supported at the item level.

The following tables are accessed by this program:

| TABLE       | SELECT | INSERT | UPDATE | DELETE |
|-------------|--------|--------|--------|--------|
| PROMDEPT    | Yes    | No     | No     | No     |
| PROMSKU     | Yes    | Yes    | No     | No     |
| ITEM_MASTER | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: PHASE 1 (daily)

Scheduling Diagram: The prmxpld must be run daily prior to the prmext module

Pre-Processing: N/A

Post-Processing: Post process is required to reset the status of the promdept records from 'AI' to null

Threading Scheme: V\_restart\_promo (this program can be multi-threaded by active promotion)

```

select promotion
from promhead
where status_code = APPROVED
and ph.start_date -
 system_variables.pos_extract_days <=
 TOMORROW
and ph.end_date > TODAY

```

### Restart Recovery

The logical unit of work for this promotion explosion module will be the expansion of a promotion subclass combination. All rows on the promdept table will be expanded to the subclass level by the on-line promotion entry conversation. Once all the items have been expanded in a subclass a commit threshold will be checked. If the commit counter has reached the threshold level, a commit will be issued (always at a subclass boundary). The restart string will therefore consist of the promotion, department, class and subclass last processed prior to the commit. A reasonable commit frequency for this module would be 1000 records.

```

select promotion,
 dept,

```

```

 class,
 subclass
 to_char(promotion)||to_char(dept)||to_char(class)||to_char(subclass)
from promdept pd,
 v_restart_active_promo rv
where pd.promotion = rv.driver_value
and rv.thread_val = THREAD
and rv.num_threads = NUN_THREADS
and pd.change_type != 'EX'
and pd.status_code = 'AI'

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

init()

initialize restart recovery

---

process()

```

open driving cursor
loop for all promotion dept/class/subclass combinations to be processed
 fetch from driving cursor
 call fill_promsku_array()

 if commit threshold has been reached
 call insert_promsku()
 end if
endloop

```

---

fill\_promsku\_array

declare cursor to get all items in the given dept/class/subclass from the item\_master table, not including any items that are already on the promsku table for the current promotion.

```

open item cursor
loop for all items in the dept/class/subclass
 fetch from item cursor
 fill promsku array with all items
 increment commit threshold counter
endloop

```

---

insert\_promsku()

insert into the promsku table all records from the promsku array

## Promotions download to output file [promdnld]

### Design Overview

The promdnld.pc program will be used to send down promotional information for the mix\_match, threshold, and multi\_unit promotional types. This information will exist on the pos\_prom\_detail table that is populated in prmext.pc. This file will assist the POS in calculating the price of a sale as well as allow it to calculate the best price to give the customer when multiple promotions are at play.

Because the pos\_prom\_detail table only will contain information for the mix\_match, threshold, and multi\_unit promotional types, this program, this program will only need to be ran when the multi\_prom\_ind on system\_options is set to 'Y'. Therefore, in the init() function, if this indicator is retrieved as 'N', exit the program normally.

This program will also be written in the Retek standard file format. The format used for this program will be: FHEAD, FDETL, FTAIL.

### Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program is run towards the end of the batch run when all pos\_mods records have been created for the transaction day.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Threading done by store

### Restart Recovery

Restart/recovery for this program is set up at the store level. Likewise, threading is done by store using the v\_restart\_store view to thread properly.

The commit\_max\_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

## Program Flow



### Shared Modules

N/A

### Function Level Description

#### Init

This function initializes restart/recovery for this program. It also opens the output file, retrieves system variables and calls a function to size the arrays used in this program.

#### Process

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from pos\_prom\_detail where the pos\_prom\_detail.store value is greater than zero. Once the records are fetched, the process\_prom\_detail() function is called to perform processing on them. After each record is moved to the output file, pos\_prom\_delete() will remove the record from the table. Restart/Recovery and committing of records is also performed here.

#### Process\_prom\_detail

This function will prepare records for insert into the output file. This program used the Retek standard file format FHEAD, FDETL, FTAIL.

#### Pos\_prom\_delete

This function deletes the current record from the pos\_prom\_detail table.

#### Final

This function will finish restart/recovery logic, close the output file and delete the temporary output file used while the program processes.

Init\_format\_strs

This function formats the strings for the FHEAD, FDETL, and FTAIL records in the output file.

size\_arrays

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart\_control.

### I/O Specification

Promotion\_detail 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 Identifier        | Number(10) | Sequential number Created by program. | ID of current line being created for output file.                                                            |
|             | File Type Definition        | Char(4)    | PRMD                                  | Identifies file as 'PROM Download'                                                                           |
|             | File Create Date            | Char(14)   | create date                           | current date, formatted to 'YYYYMMDDHH24 MISS'.                                                              |
| File Detail | File Type Record Descriptor | Char(5)    | FDETL                                 | Identifies file record type                                                                                  |
|             | File Line Identifier        | Number(10) | Sequential number Created by program. | ID of current line being created for output file.                                                            |
|             | Location Number             | Number(10) | store identifier                      | Store identifier from the Retek system.                                                                      |
|             | Promotion Number            | Number(10) | promotion number                      | Promotion number for SKU from Retek promotion tables.                                                        |
|             | Transaction Type            | Char(6)    | promotion transaction type            | Transaction type indicating the type of promotion taken from the code_detail table where code_type = 'PRMT'. |
|             | Start time                  | Char(6)    | start time                            | start time of the promotion, formatted                                                                       |

|                  |            |                                                                   |                                                                                                                                 |
|------------------|------------|-------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| End time         | Char(6)    | end time                                                          | to 'HH24MISS' from Retek promotion tables.<br>end time of the promotion, formatted to 'HH24MISS' from Retek promotion tables.   |
| Mix Match Number | Number(10) | mix match number                                                  | mix match number for SKU from Retek promotion tables.                                                                           |
| Threshold Number | Number(10) | threshold number                                                  | threshold number for the SKU from Retek promotion tables.                                                                       |
| Apply To         | Char(1)    | 'C' for Compound or 'E' for Exclusive                             | Indicates if the promotion can be used with other promotions. From Retek promotion tables.                                      |
| Buy Type         | Char(1)    | 'A' for All, 'U' for Any                                          | Indicates how many of the prom mix match buy SKU's must be purchased to qualify for the promotion. From Retek promotion tables. |
| Buy Amount       | Number(20) | buy amount                                                        | The amount that has to be purchased to qualify for the promotion. From Retek promotion tables.                                  |
| Get Type         | Char(1)    | 'P' for Percent off, 'A' for Amount off, or 'F' for fixed amount. | Indicates how to lower the price for the prom mix match get SKU's. From Retek promotion tables.                                 |
| Get Amount       | Number(20) | get amount                                                        | The amount or percent off of the Get SKU(s) or the amount the SKU(s) are on promotion for. From Retek promotion tables.         |
| Selling UOM      | Varchar(4) | Selling Unit of Measure                                           | The selling unit of measure for an item's single unit retail.                                                                   |
| Threshold Amount | Number(20) | threshold amount                                                  | The threshold level needed to be reached to qualify for the promotion. From                                                     |

|              |                             |             |                                          |                                                                                                                      |
|--------------|-----------------------------|-------------|------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
|              | Discount Amount             | Number(20 ) | discount amount                          | Retek promotion tables.<br>The amount of discount when the threshold amount is reached. From Retek promotion tables. |
|              | Discount Type               | Char(1)     | Discount type                            | Discount type indicator.                                                                                             |
| File Trailer | File Type Record Descriptor | Char(5)     | FTAIL                                    | Identifies file record type                                                                                          |
|              | File Line Identifier        | Number(10 ) | Sequential number<br>Created by program. | ID of current line being created for output file.                                                                    |
|              | File Record Counter         | Number(10 ) |                                          | Number of records/transactions processed in current file (only records between head & tail)                          |



## Reclassification of item [reclsdy]

### Design Overview

Reclsdy will be modified to work with the new item dialog. Functionality that does not work for A&P Phase 1 will be turned off.

### Function Level Description

Remove the following macros :

NULL\_PACK\_TYPE.  
NULL\_SYSTEM\_IND.

Remove all EXEC SQL VARs.

Change the following variables :

Os\_restart\_sku[NULL\_SKU] to os\_restart\_item[NULL\_ITEM].  
Os\_sku[NULL\_SKU] to os\_item[NULL\_ITEM].  
Os\_system\_ind[NULL\_SYSTEM\_IND] to  
os\_item\_num\_type[NULL\_ITEM\_NUM\_TYPE].

In the reclass\_fetched\_array, char (\*ps\_sku)[NULL\_SKU] to char (\*ps\_item)[NULL\_ITEM]. Also add variables to this fetch array for a pack\_type, item\_level, and tran\_level.

Create variables os\_pack\_type, os\_item\_level, and os\_tran\_level.

The c\_reclass cursor needs to something like the following :

```
SELECT ri.rowid,
 Ri.reclass_no,
 Ri.item,
 Item_level,
 Tran_level,
 to_dept,
 to_class,
 to_subclass,
 dept,
 class,
 subclass,
 pack_type,
 orderable_ind
FROM v_restart_reclass,
 Reclass_item ri,
 Reclass_head,
 Item_master
WHERE reclass_head.reclass_no = reclass_item.reclass_no
 AND reclass_head.reclass_date < vdate
 AND reclass_item.item = item_master.item
```

```

 AND item_master.item_level = 1
UNION
SELECT ri.rowid,
 Ri.reclass_no,
 im.item,
 Item_level,
 Tran_level,
 rh.to_dept,
 rh.to_class,
 rh.to_subclass,
 im.system_ind,
 im.dept,
 im.class,
 im.subclass,
 pack_type,
 orderable_ind
FROM v_restart_reclass,
 Reclass_item ri,
 Reclass_head rh,
 Item_master im
WHERE rh.reclass_no = ri.reclass_no
AND rh.reclass_date <= VDATE
AND ri.item = im.item_parent
AND im.item_level = 2
UNION
SELECT ri.rowid,
 Ri.reclass_no,
 im.item,
 Item_level,
 Tran_level,
 rh.to_dept,
 rh.to_class,
 rh.to_subclass,
 im.system_ind,
 im.dept,
 im.class,
 im.subclass,
 pack_type,
 orderable_ind
FROM v_restart_reclass,
 Reclass_item ri,
 Reclass_head rh,
 Item_master im
WHERE rh.reclass_no = ri.reclass_no
AND rh.reclass_date <= VDATE
AND ri.item = im.item_grandparent
AND im.item_level = 3

```

The cursor should retain the current threading and driver functionality.

### Process()

The DELETE FROM reclass\_sku needs to be changed to DELETE FROM reclass\_item.

### Check\_domain\_exists()

#### Process\_sku()

Change name to Process\_item(). The input parameter, char \* is\_sku, needs to be changed to char \* is\_item.

The c\_get\_pack\_type cursor and all related functionality needs to be deleted.

The parameters passed to RECLASS\_SQL.ITEM\_PROCESS have changed.

```
RECLASS_SQL.ITEM_PROCESS(ls_error_message,
 :os_sku,
 :os_item_level,
 :os_tran_level,
 TO_DATE(:ps_vdate, 'YYYYMMDD'),
 :os_system_forecast_ind,
 :lb_domain_exists,
 :os_new_domain_id,
 :os_old_dept,
 :os_old_class,
 :os_old_subclass,
 :os_to_dept,
 :os_to_class,
 :os_to_subclass)
```

Comment out the last part of this function, starting here :

```
if((*is_system_ind == 'S') || (*is_system_ind == 'f'))...
```

#### Process\_security()

Local variable L\_sku needs to be changed to L\_item[NULL\_ITEM];

C\_sku cursor needs to be renamed to C\_item, and to select ITEM from ITEM\_MASTER where ITEM\_GRANDPARENT = :os\_item.

The DELETE FROM sec\_group\_prod\_matrix needs to be changed to :

```
DELETE FROM sec_group_prod_matrix
WHERE level_1 = :os_item
 OR level_2 = :os_item
 OR level_3 = :os_item
```

The DELETE FROM sec\_user\_prod\_matrix needs to be changed to :

```
DELETE FROM sec_user_prod_matrix
WHERE level_3 = :os_item
```

The parameters passed to LOC\_PROD\_SECURITY\_SQL.INSERT\_USER\_SEC have changed.

```
LOC_PROD_SECURITY_SQL.INSERT_USER_SEC(ls_error_message,
 :os_to_dept,
```

```

:os_to_class,
:os_to_subclass,
:os_sku, /* item level 1
*/
NULL, /* item level 2
*/
NULL) /* item level 3
*/

```

**Get\_order\_numbers()**

The input parameter char \* is\_fetched\_sku needs to be changed to char \* is\_fetched\_item.

All code relating to this function must be commented out.

**Call\_insert\_deal\_sku\_temp()**

The input parameter char \* is\_sku needs to be changed to char \* is\_item.

All code relating to this function must be commented out.

**Get\_supplier\_origin\_id()**

The input parameter char \* is\_sku needs to be changed to char \* is\_item.

All code relating to this function must be commented out.

**Order\_exists()**

All code relating to this function must be commented out.

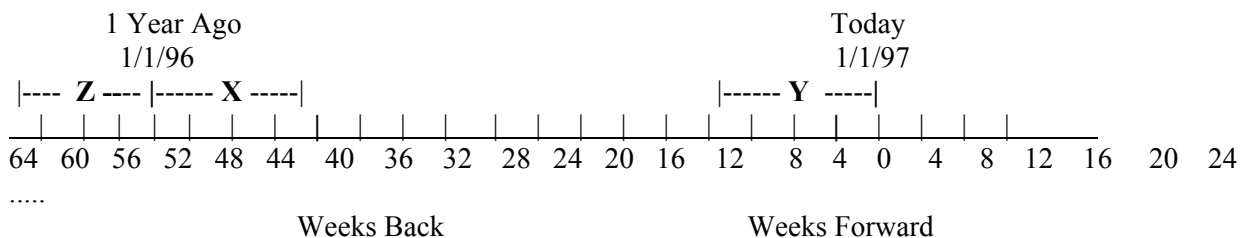
## Replenishment adjustment [repladj]

### Design Overview

This program will re-calculate the maximum stock levels for all fashion item locations with a replenishment method of 'F' (floating point). The floating model stock method will dynamically calculate an order-up-to-level. The maximum model stock is calculated using the sales history of various periods of time in order to accommodate seasonality as well as trend. The sales history is obtained from the item\_loc\_hist table:

$$\text{Maximum Model Stock} = \text{Sales Rate} + \text{service level factor} \times \text{SQRT}(\text{Sales Rate})$$

Sales Rate = Avg. daily sales for the upcoming 12 weeks of last year (see X in figure 1.0)  
 x Avg. daily sales for the previous 12 weeks  
 (see Y in figure 1.0)  
 / Avg. daily sales for the previous 12 weeks of last year  
 (see Z in figure 1.0)  
 x (Lead Time + days between reorder cycles)



Service Level Factor = 1 if the user sets the service level to an 85% in-stock service level, or 2 if the user sets the service level to an 98% in-stock service level.

When calculating the sales rates, only the weeks with clean regular sales are taken into account. Weeks with promo or clearance sales are not included. For example, using the following:

| Week      | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8 | 9 | 10 |
|-----------|----|----|----|----|----|----|----|---|---|----|
| Sales     |    |    |    |    |    |    |    |   |   |    |
| Regular   | 10 | 15 | 3  |    |    | 5  | 10 | 0 | 0 | 10 |
| Promotion |    |    | 20 | 25 | 20 | 10 |    |   |   |    |

$$\begin{aligned} \text{Daily Sales Rate} &= (\text{Week 1} + \text{Week 2} + \text{Week 7} + \text{Week 8} + \text{Week 9} \\ &\quad + \text{Week 10})/42 \\ &= (10 + 15 + 10 + 0 + 0 + 10)/42 \\ &= 1.0714 \end{aligned}$$

The calculation for the order quantity is the same for this method as for the Constant method (maximum stock - quantity available). The dynamically calculated maximum cannot go below the minimum stock level set by the user.

Substitute items are considered as well. If the item has substitute items and their sales histories are being used (i.e. use sales history indicator set to Yes), the sales histories for the main item and substitute items are summed across the time periods and their totals are used to determine the sales rate ratio.

| TABLE             | SELECT | INSERT | UPDATE | DELETE |
|-------------------|--------|--------|--------|--------|
| REPL_ITEM_LOC     | Yes    | No     | Yes    | no     |
| SUB_ITEMS_HEAD    | yes    | No     | no     | no     |
| SUB_ITEMS_DETAIL  | yes    | No     | no     | no     |
| ITEM_LOC_HIST     | Yes    | No     | No     | No     |
| ITEM_SUPP_COUNTRY | Yes    | No     | No     | No     |

### Stored Procedures / Shared Modules (Maintainability)

REPL\_OLT\_SQL.GET\_OLTS\_AND\_REVIEW\_TIME – to retrieve the number of days until the next review for the given item/location combination.

4\_5\_4.pc – used to calculate the dates used in the sales rate calculations.

### Function Level Description

Process() – this function contains the driving cursor and program control logic. The driving cursor brings back all item/location combinations that are scheduled to be reviewed today and have a replenishment method of 'F' or floating point. It brings back both supplier stocked locations and warehouse stocked locations on floating point replenishment. The replenishment information is found on REPL\_ITEM\_LOC. Additional information that is needed for the program's calculations is brought back from the ITEM\_SUPP\_COUNTRY, REPL\_DAY, SUPS, WH, and STORE tables.

Cal\_info() – Based on the vdate calculates all dates needed to perform the sales rate calculation.

Get\_review\_time() – Gets the number of days between the review cycles for the rate calculations. This function is a wrapper for a call to REPL\_OLT\_SQL.GET\_OLTS\_AND\_REVIEW\_TIME.

F\_issue\_rate() – Calculates the issue rate for an item.

F\_sale\_rate() – Calculates the sales rate for an item.

### Scheduling Considerations

Processing Cycle: PHASE 3

Scheduling Diagram: Must run before rplex/reqext and after rplatupd.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: This program is threaded by dept.

### Restart/Recovery

This program implements restart/recovery. The logical unit of work for this program is a valid item/location combination. Item/location is an unique logical unit of work.

## Item requisition extraction [reqext]

### Design Overview

Reqext (Item Requisition Extraction) handles automatic replenishment of items from warehouses to stores. It cycles through every item-store combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be transferred to the store (if any). In addition, it distributes this *Recommended Order Quantity (ROQ)* over any applicable *alternate items* associated with the item. The program then takes this information and either creates new transfer line items or adds to existing ones.

Alternate items are either *simple packs* or *substitute items*. Simple packs are sellable and orderable packs that contain only a single item, such as a six-pack of cola or twelve-pack of socks. Substitute items are items predefined to be interchangeable with the item being replenished (referred to as the *master item*).

When an item is set up to use simple packs (designated by an indicator on the REPL\_ITEM\_LOC table), the ROQ must be distributed among these packs according to desirability. If a master item has no simple packs associated with it, it will be requested as itself. If there is only one pack associated with the item (referred to as the *primary simple pack*), then there is no distribution needed – the item will be transferred in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. ***Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program!*** Because the pack distribution depends on access to the huge sales history tables (PACKSTORE\_HIST, RAG\_SKUS\_ST\_HIST, WIN\_STORE\_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

If an item is not set up to use simple packs, the program will see if any substitute items are associated with it. If there are no substitute items associated with the master item, it will be transferred alone. If there are substitute items, they will be fetched into a list and the master item placed at either the head or tail end of the list, depending on the *fill priority* (set on the SUB\_ITEMS\_HEAD table). The priority determines which items are transferred first.

No matter what type of alternate items (if any) are used, the program will account for availability when building transfer line items. For simple packs, the share of ROQ allocated to each pack may be decreased or increased if the source warehouse has a shortage of some packs but a surplus of others. For substitute items, transfer quantities are prorated by calculating the ratio of total availability to total need, and items are transferred in order of priority until all need is filled or until no stock is available.

Once the transfer quantity of an item has been calculated, the transfer line item is posted to the database if **1)** the actual quantity to transfer is greater than zero, and **2)** the replenishment order control indicator for the item-store combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL\_RESULTS) for reporting purposes. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL\_RESULTS, *even if the quantity to order is zero*. Whenever a transfer line item is placed, the appropriate item-location table (RAG\_SKUS\_ST, RAG\_SKUS\_WH, WIN\_STORE,

WIN\_WH and/or PACKWH) is updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: Rplatupd, rpladjf and rpladjs need to run before rext so that all replenishment calculation attributes are up to date. Posupld, tsfoupld, tsfiupld, ctniupld, and rcvupld need to run before rext so that all stock information is up to date. Rplext should run after rext, since the ROQ for a warehouse is influenced by any transfers created.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: DEPT

### Restart Recovery

The logical unit of work is item, source warehouse. The driving cursor is ordered by item, source warehouse, order control indicator and simple pack indicator. When any of these values change during the course of processing (i.e., the current value is different than that of the previous record), then a transfer will be created, taking total quantities and availability into consideration (see replenish\_item(), below).

### Program Flow

N/A

### Shared Modules

GET\_REPL\_ORDER\_QTY\_SQL.REPL\_METHOD: Stored PL/SQL procedure for calculating the ROQ of an item at a location.

REPLENISHMENT\_SQL.GET\_STORE\_REVIEW\_TIME: Stored PL/SQL procedure for calculating the time between scheduled shipments to a store from a warehouse. This time is used by GET\_REPL\_ORDER\_QTY\_SQL in its calculations.

ITEMLOC\_QUANTITY\_SQL.GET\_WH\_CURRENT\_AVAIL: Stored PL/SQL procedure for calculating the amount of a given item available at a given warehouse.

NEXT\_TRANSFER\_NUMBER: Stored PL/SQL procedure used for getting the next valid transfer number for use in creating new transfers.

RMS\_ROUND\_TO\_PACKSIZE: Shared C function (see rpl.h) used in rounding an item's quantity up to the size of a simple pack, or for rounding an order quantity up to a receivable pack size.



## Data Structures

*repl\_info\_struct*: Holds information fetched from the driving cursor.

*store\_struct*: Holds information about item-location combinations, used for ROQ and distribution calculations.

*alt\_item\_struct*: Holds information about alternate items associated with a given master item. Used in distribution calculations.

*tsfhead\_struct*: Used to buffer inserts to the TSFHEAD table.

*tsfdetail\_struct*: Used to buffer inserts and updates to the TSFDETAIL table.

*item\_loc\_struct*: Used to buffer updates to the item-location tables (RAG\_SKUS\_ST, RAG\_SKUS\_WH, WIN\_STORE, WIN\_WH, PACKWH).

*repl\_results\_struct*: Used to buffer inserts to the REPL\_RESULTS table.

*domain\_struct*: Used to cache forecasting domain information.

## Function Level Description

### General Controlling Functions

#### *main()*

The standard Retek main function, this calls *init()*, *process()* and *final()*, and posts messages to the daily log files.

#### *init()*

Initializes the Restart-Recovery API and fetches system-level global variables.

#### *driving\_cursor()*

Opens, fetches data from, or closes the driving cursor. This is a support function for *process()*.

#### *process()*

This function fetches records from the driving cursor (*driving\_cursor()*), passes them to *replenish\_item()* to perform all appropriate actions, and commits work when appropriate (*post\_all()*, *restart\_commit()*).

#### *replenish\_item()*

The controlling function for replenishment calculations. This function copies records out of the driving cursor buffer (*copy\_repl\_to\_store()*), and calculates the ROQ for each record (*get\_repl\_order\_qty()*). If a change in item, source warehouse, order control indicator, or simple pack indicator has occurred, the appropriate functions are called to calculate distribution of need over all appropriate alternate items and stores, and to place the transfers. If item's ROQ is zero or negative, no mater simple pack indicator is on the master item will be used for replenishment (*build\_pack\_ratio()*, *calc\_pack\_dist()*, *calc\_sub\_dist()*).

#### *place\_tsf\_line\_item()*

This function takes a item-location combination and a transfer quantity, and actually builds the transfer line item (*handle\_tsf()*). It then updates the item-location tables to reflect the change in

stock (`handle_item_loc()`), and writes a record to the reporting table (`handle_repl_results()`) when appropriate.

#### *final()*

The standard Retek final function, this closes down the Restart-Recovery API.

### Simple Pack Distribution and Transfer

#### *build\_pack\_ratio()*

Calculates distribution of the master item's recommended order quantity (ROQ) over simple packs. Simple packs are sellable and orderable packs containing only a single item (e.g., six-pack of cola). Since the cost per item will always be less in a pack than singularly, the item will only be ordered in terms of simple packs (if any are applicable). This function tries to divide the total ROQ for the item among all applicable simple packs by using the packs' relative sales history to build a distribution 'mask' containing ratios used to calculate each pack's share of the ROQ. This mask is then adjusted to account for availability (shortages of some packs, surpluses of others).

This function performs the following steps to optimally distribute the ROQ among any and all simple packs:

- If a primary simple pack was defined for this item, that pack will be the only one used to supply the item (`add_primary_pack()`).
- If no primary pack was defined, the program will build a list of all simple packs associated with the item (`get_multi_simple_pack()`).
- If no appropriate simple packs are found, the item will be ordered as itself (`add_single_item()`).
- The historical sales for all simple packs and the master item are added up.
- The ROQ is distributed among the simple packs by taking the ratio of each pack's historical sales to the total historical sales (`first_ratio_pass()`).
- If the first pass through the list did not account for the entire ROQ because of lack of availability for some packs, the program must keep cycling through the items until it has either distributed the ROQ among all available packs or there is simply no available stock left to supply the need (`next_ratio_pass()`).

#### *first\_ratio\_pass()*

Performs initial distribution of an item's ROQ among its associated simple packs. Calculates each pack's share as a ratio of its historical sales to the total historical sales (`adjust_pack_ratio()`). The historical sales of the master item are added to those of the simple pack with the lowest cost to give it a greater share of the ROQ. This is a support function for `build_pack_ratio()`.

#### *next\_ratio\_pass()*

This function readjusts the ratios of still-available packs to try and cover the share of ROQ not yet allocated, still distributing the leftover ROQ proportionally by historical sales. This is a support function for `build_pack_ratio()`.

#### *adjust\_pack\_ratio()*

Sets a simple pack's share of the ROQ to reflect its desirability (in terms of historical sales patterns), adjusting for availability. This is a support function for `first_ratio_pass()` and `next_ratio_pass()`.

#### *add\_primary\_pack()*

If an item is flagged to have a primary simple pack, that pack is the only one that will be transferred. This function adds the primary pack to the simple pack distribution array and assigns it the full share of the ROQ. This is a support function for `build_pack_ratio()`.

#### *get\_multi\_simple\_pack()*

Finds all simple packs associated with a given master item and information about them (historical sales, availability, etc). This is a support function for `build_pack_ratio()`.

#### *get\_single\_sales\_hist()*

Gets the historical sales of the master item at all stores supplied by the given warehouse for use in calculating distribution among simple packs. Since the master item will only be transferred as a pack, this sales amount will be added to that of the pack with the lowest cost, increasing its share of the ROQ. This is a support function for `build_pack_ratio()`.

#### *add\_single\_item()*

If an item is flagged to use simple packs, but none are found, it will be ordered as itself. This function adds the master item to the simple pack distribution structure and assigns it the full share of the ROQ. This is a support function for `build_pack_ratio()`.

#### *calc\_pack\_dist()*

Once each simple pack's share of the item's ROQ has been calculated in `build_pack_ratio()`, this function calculates actual transfer quantities and places the transfer line items (`place_tsf_line_item()`). The function loops through each pack in the list, calculating the amount of the pack to transfer to each store (`calc_pack_tsf_qty()`). If the total transfer quantity of the pack exceeds its availability at the warehouse, each store will have its quantity reduced by one receivable pack until a reasonable number has been reached. Finally, a transfer line item is placed for the pack to the store.

#### *calc\_pack\_tsf\_qty()*

Calculate the actual quantity to transfer for a store based on an alternate item's share of the ROQ at a store, adjusted for any applicable simple pack and/or shipping pack sizes. This is a support function for `calc_pack_dist()`.

### Substitute Item Distribution and Transfer

#### *calc\_sub\_dist()*

Calculate distribution of the ROQ over substitute items. Substitute items are items (selected by the user beforehand) that can be requested in place of a given item to cover situations where availability is too low or demand is too high.

After calling `get_sub_items()` to generate a list of appropriate items for transfer, the function loops through every item-location combination and performs the following steps to make sure that both need and availability are accounted for when placing transfers from the warehouse to the stores:

- If the total availability of all items in the substitute list cannot cover the full need over all stores, then the ratio of the total availability to the total ROQ is calculated. If total availability *can* cover total ROQ, the ratio is set to 1.
- The initial transfer quantity for the item at the location is calculated as the store's need adjusted by the availability ratio, and rounded up to a receivable pack size.
- If there is not enough of the item available at the warehouse to fill the calculated transfer quantity, the quantity will be decremented to an orderable amount.

- The transfer line item is placed by calling `place_tsf_line_item()`.
- The store's ROQ, total ROQ, availability of the item, and total availability are all decremented by the amount just transferred to prepare for the next item-location's calculation.

#### *get\_sub\_items()*

Retrieves substitute items for the current master item and information about them from the database (receiving pack size, availability, etc.). If the fill priority for this set of items (`SUB_ITEMS_HEAD.fill_priority`) is set to 'M'aster, the master item will be the first one in the list, and will be used first to fill need. If it is set to 'S'ubstitute, the master item will be placed at the tail end of the list. This is a support function `calc_sub_dist()`.

#### *add\_master()*

Adds the master item to the appropriate position in the substitutes list. This is a support function for `get_sub_items()`.

#### *shift\_subs()*

If the fill priority for the substitutes list is set to 'M'aster, the master item must be placed at the head of the list. This function clears out the first position by moving each substitute item 'back' a slot. This is a support function for `get_sub_items()`.

### Database DML Handling

#### *post\_all()*

The DML handling functions (`handle_tsf()`, `handle_item_loc()`, `handle_repl_results()`) normally only post information to the database tables when their respective buffers are full. When a commit point is reached, however, all buffers must be flushed to ensure restartability. This function forces all the buffers to be posted to the database.

#### *handle\_tsf()*

Controls handling of inserts and updates to the Transfer tables.

#### *add\_tsfhead()*

Deals with transfer header information. Either finds an appropriate transfer to add line items to (matching to/from locations, department and freight code), or creates a new one. passes back the transfer number for use in `add_tsfdetail()`. This is a supporting function for `handle_tsf()`.

#### *add\_tsfdetail()*

Deals with transfer detail information. Either finds an appropriate record on the `TSFDETAIL` table to add quantity to (matching transfer number and item), or creates a new one if none is found. This is a supporting function for `handle_tsf()`.

#### *get\_next\_seq\_no()*

Every line item on a transfer has a unique identifier within that transfer. This function gets the next sequence number for a new line item. This is a supporting function for `add_tsfdetail()`.

#### *post\_tsf()*

Posts transfer information to the database. Inserts to `TSFHEAD`, inserts and updates to `TSFDETAIL`. This is a supporting function for `handle_tsf()`.

#### *handle\_item\_loc()*

Whenever a transfer is created or modified, the source location's reserved quantity and the receiving location's expected quantity must be adjusted to reflect the new stock status. This function controls the handling of updates to the RAG\_SKUS\_ST, RAG\_SKUS\_WH, WIN\_STORE, WIN\_WH and PACKWH tables.

#### *add\_item\_loc()*

Adds records to arrays for update of expected and reserved quantities on the item-location tables (RAG\_SKUS\_ST, RAG\_SKUS\_WH, WIN\_STORE, WIN\_WH, PACKWH) based on the appropriate item types. This is a support function for *handle\_item\_loc()*.

#### *post\_item\_loc()*

Posts item-location stock status changes to the database (RAG\_SKUS\_ST, RAG\_SKUS\_WH, WIN\_STORE, WIN\_WH, PACKWH). This is a support function for *handle\_item\_loc()*.

#### *handle\_repl\_results()*

Controls posting of report information to the REPL\_RESULTS table.

#### *add\_repl\_results()*

Adds records to the replenishment results structure for reporting. This is a supporting function for *handle\_repl\_results()*.

#### *post\_repl\_results()*

Posts replenishment information to the REPL\_RESULTS table. This is a supporting function for *handle\_repl\_results()*.

#### *update\_review\_date()*

Updates the last\_review\_date column on the REPL\_ITEM\_LOC table to reflect the fact that item-location combinations have just been evaluated.

### PL/SQL Stored Procedure Calls

#### *get\_repl\_order\_qty()*

Gets the recommended order quantity (ROQ) and other variables for an item-location-supplier and populates elements of a store information structure with them. This function is a wrapper for the REPLENISHMENT\_SQL.GET\_STORE\_REVIEW\_TIME and GET\_REPL\_ORDER\_QTY\_SQL.REPL\_METHOD stored PL/SQL procedures.

#### *get\_wh\_current\_avail()*

Gets the available quantity of a given item at a given warehouse. This function is a wrapper for the ITEMLOC\_QUANTITY\_SQL.GET\_WH\_CURRENT\_AVAIL stored PL/SQL procedure.

#### *next\_transfer\_number()*

Gets the next transfer number in the Oracle stored sequence for creating new transfer headers. This function is a wrapper for the NEXT\_TRANSFER\_NUMBER stored procedure.

### Domain Validation

#### *get\_domain\_info()*

The GET\_REPL\_ORDER\_QTY\_SQL stored procedure requires a forecasting domain when performing calculations for item-location combinations using a forecasted method of replenishment (Dynamic or Time Supply). For a given department, class or subclass (depending on the system-level Domain Level option), this function finds the associated forecasting domain.

The first time this is called, it will call `build_domain_cache()` to pull all domain information for this thread into an array, and return the desired domain. Subsequent calls will only query the array.

#### *build\_domain\_cache()*

Populates a structure with information on department/class/subclass/domain relationships. Because the program is threaded by department, the function will only pull domains associated with departments, classes, or subclass associated with the current thread.

### Support Functions

#### *copy\_repl\_to\_store()*

Copies a record from the structure holding rows from the driving cursor into a structure holding item-location information for ROQ calculation, distribution, and transfer placement.

#### *reset\_store\_struct()*

Resets summary variables in a store information structure to prepare it for the next set of line items.

#### *reset\_alt\_item\_struct()*

Resets summary variables in an alternate item structure to prepare it for the next set of alternates.

### Array Sizing

#### *size\_repl\_info\_struct()*

Allocates memory to the structure used to buffer fetches from the driving cursor.

#### *size\_store\_struct()*

Allocates memory to the structure used to hold item-location level information.

#### *size\_alt\_item\_struct()*

Allocates memory to the structure used to hold information about alternate items (either simple packs or substitute items).

#### *size\_tsfhead\_struct()*

Allocates memory to the structure used to buffer inserts to the Transfer Header table.

#### *size\_tsfdetail\_struct()*

Allocates memory to structures used to buffer inserts and updates to the Transfer Detail table.

#### *size\_item\_loc\_struct()*

Allocates memory to structures used to buffer updates of the item-location tables (RAG\_SKUS\_ST, RAG\_SKUS\_WH, WIN\_STORE, WIN\_WH, PACKWH).

#### *size\_repl\_results\_struct()*

Allocates memory to the structure used to buffer inserts to the Replenishment Results table.

### Database Interaction

#### Tables Selected From:

DOMAIN\_CLASS  
DOMAIN\_DEPT

DOMAIN\_SUBCLASS  
ITEM\_SUPP\_COUNTRY  
PACKHEAD  
PACKITEM  
PACKSTORE\_HIST  
PERIOD  
RAG\_SKUS\_ST\_HIST  
REPL\_DAY  
REPL\_ITEM\_LOC  
STORE  
SUB\_ITEMS\_HEAD  
SUB\_ITEMS\_DETAIL  
SYSTEM\_OPTIONS  
TSFDETAIL  
TSFHEAD  
V\_RESTART\_DEPT (*View*)  
WH  
WIN\_STORE\_HIST

Tables Inserted To:

REPL\_RESULTS  
TSFDETAIL  
TSFHEAD

Tables Updated:

PACKWH  
RAG\_SKUS\_ST  
RAG\_SKUS\_WH  
REPL\_ITEM\_LOC  
TSFDETAIL  
WIN\_STORE  
WIN\_WH

**I/O Specification**

N/A

**Technical Issues**

N/A

## Replenishment item-location maintenance [rilmaint]

### Design Overview

This program is performance enhancement to replenishment. It works in conjunction with the REPL\_ITEM\_LOC\_UPDATES staging table. REPL\_ITEM\_LOC\_UPDATES is populated when certain attributes effecting replenishment are modified. These attributes are located across the entire system and are monitored for changes by a series of and triggers and modules. Once a change is logged in REPL\_ITEM\_LOC\_UPDATES rilmaint.pc will note the type of change and update REPL\_ITEM\_LOC appropriately.

Before this program existed, the replenishment programs (reqext.pc and rplext.pc) would have to perform table lookups (extra-joins) to get the information that rilmaint.pc provides for them. With rilmaint.pc, the driving cursors of the replenishment programs are simpler and much faster (hopefully).

### Function Level Description

There are five basic replenishment scenarios that are important for this program. They are:

**Error! Not a valid link.**

#### IM –

Input = item

Handles A<sub>1</sub>, A<sub>2</sub>, B, C, D – Update all REPL\_ITEM\_LOC records for **dept, class, subclass, and status** where the item being changed on ITEM\_MASTER matches REPL\_ITEM\_LOC. If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, the ITEM\_MASTER information reflects the primary\_repl\_pack.

#### ILSOM –

Input = item, loc

Handles A<sub>1</sub>, A<sub>2</sub>, B, C, D – Update all REPL\_ITEM\_LOC records for **store\_ord\_mult** where the item/loc being changed on ITEM\_LOC matches REPL\_ITEM\_LOC. If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, the ITEM\_LOC information reflects the primary\_repl\_pack.

#### ILSC –

Input = item, loc

Handles A<sub>2</sub> – Update all REPL\_ITEM\_LOC records for **inner\_pack\_size, supp\_pack\_size, ti, hi, supp\_lead\_time, round\_lvl, round\_to\_inner\_pct, round\_to\_case\_pct, round\_to\_layer\_pct, and round\_to\_pallet\_pct** where the item/loc being changed on ITEM\_LOC matches REPL\_ITEM\_LOC item/source\_wh for stores being replenished from warehouses that themselves are not on replenishment. If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, the ITEM\_LOC information



reflects the `primary_repl_pack`. The pack sizes and rounding information are retrieved from `ITEM_SUPP_COUNTRY` and `ITEM_SUPP_COUNTRY_LOC` given the item, loc, `primary_supp`, and `primary_cntry` on the `ITEM_LOC` record being updated.

**Unit\_cost** is not populated on `REPL_ITEM_LOC` for warehouse stocked store records. Thus it is not updated in the case.

#### ISC –

Input = item, supplier, origin\_country\_id

Handles B, C, D – Update all `REPL_ITEM_LOC` records for **inner\_pack\_size, supp\_pack\_size, ti, hi, and supp\_lead\_time** where the item/supplier/origin\_country\_id being updated on `ITEM_SUPP_COUNTRY` matches the `REPL_ITEM_LOC` item/primary\_repl\_supplier/origin\_country\_id and the `REPL_ITEM_LOC` is using supplier replenishment.

Handles A<sub>1</sub> – Update all `REPL_ITEM_LOC` records for **inner\_pack\_size, supp\_pack\_size, ti, hi, and supp\_lead\_time** where the item being updated on `ITEM_SUPP_COUNTRY` matches the `REPL_ITEM_LOC` item for a warehouse stocked store (`stock_cat = 'W'` and loc type = 'S') where the source\_wh is on replenishment and the item/primary\_repl\_supplier/origin\_country\_id of the source\_wh's `REPL_ITEM_LOC` records matches the item/supplier/origin\_country\_id of the `ITEM_SUPP_COUNTRY` record being updated.

Handles A<sub>2</sub> – Update all `REPL_ITEM_LOC` records for **inner\_pack\_size, supp\_pack\_size, ti, hi, and supp\_lead\_time** where the item being updated on `ITEM_SUPP_COUNTRY` matches the `REPL_ITEM_LOC` item for a warehouse stocked store (`stock_cat = 'W'` and loc type = 'S') where the source\_wh is not replenishment and the item/primary\_supp/primary\_cntry of the source\_wh's `ITEM_LOC` records matches the item/supplier/origin\_country\_id of the `ITEM_SUPP_COUNTRY` record being updated.

If a `primary_repl_pack` exists for the `REPL_ITEM_LOC` record, the `ITEM_SUPP_COUNTRY` information reflects the `primary_repl_pack`.

#### ISCLR –

Input = item, location, supplier, origin\_country\_id

Handles B, C, D – Update the `REPL_ITEM_LOC` records **round\_lvl, round\_to\_inner\_pct, round\_to\_case\_pct, round\_to\_layer\_pct, and round\_to\_pallet\_pct** where the item/loc/supplier/origin\_country being updated on `ITEM_SUPP_COUNTRY_LOC` matches the `REPL_ITEM_LOC` item/location/primary\_repl\_supplier/origin\_country\_id and the `REPL_ITEM_LOC` record is using supplier replenishment.

Handles A<sub>1</sub> – Update all `REPL_ITEM_LOC` records **round\_lvl, round\_to\_inner\_pct, round\_to\_case\_pct, round\_to\_layer\_pct, and round\_to\_pallet\_pct** where the item/location being updated on `ITEM_SUPP_COUNTRY_LOC` matches the item/source\_wh on `REPL_ITEM_LOC` and the source\_wh for the item is on supplier replenishment.

Handles A<sub>2</sub> – Update all `REPL_ITEM_LOC` records **round\_lvl, round\_to\_inner\_pct, round\_to\_case\_pct, round\_to\_layer\_pct, and round\_to\_pallet\_pct** where the

item/location being updated on ITEM\_SUPP\_COUNTRY\_LOC matches the item/source\_wh on REPL\_ITEM\_LOC and the source\_wh for the item is not on supplier replenishment.

If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, the ITEM\_SUPP\_COUNTRY\_LOC information reflects the primary\_repl\_pack.

#### ISCLC –

Input = item, location, supplier, origin\_country\_id

Handles B, C – Update the REPL\_ITEM\_LOC records **unit\_cost** where the item/loc/supplier/origin\_country being updated on ITEM\_SUPP\_COUNTRY\_LOC matches the REPL\_ITEM\_LOC item/location/primary\_repl\_supplier/origin\_country\_id and the REPL\_ITEM\_LOC record is using direct to location supplier replenishment.

Handles D – Update the REPL\_ITEM\_LOC records **unit\_cost** where the item/loc/supplier/origin\_country being updated on ITEM\_SUPP\_COUNTRY\_LOC matches the REPL\_ITEM\_LOC item/source\_wh/primary\_repl\_supplier/origin\_country\_id and the REPL\_ITEM\_LOC record is using direct to xdock or xlink supplier replenishment.

If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, the ITEM\_SUPP\_COUNTRY\_LOC information reflects the primary\_repl\_pack.

#### RILP –

Input = item, location

Perform the IM logic on the item.

Perform the ILSOM logic on the item/location.

Handles B, C, D – Update the REPL\_ITEM\_LOC record's **inner\_pack\_size, supp\_pack\_size, ti, hi, and supp\_lead\_time** where the record being updated on REPL\_ITEM\_LOC matches the ITEM\_SUPP\_COUNTRY item/supplier/origin\_country\_id and the REPL\_ITEM\_LOC record is using direct to location or xdock/xlink supplier replenishment.

Handles A<sub>1</sub> – Update the REPL\_ITEM\_LOC record's **inner\_pack\_size, supp\_pack\_size, ti, hi, and supp\_lead\_time** where the record is a store being sourced by a warehouse that is on replenishment. Use the primary\_repl\_supp/origin\_country\_id on REPL\_ITEM\_LOC for the sourcing warehouse to link to ITEM\_SUPP\_COUNTRY to get the pack size/lead time info. However, if the item is being replenished in the form of a primary simple pack and that primary simple pack does not exist at the primary\_repl\_supp/origin\_country\_id/source\_wh combination on REPL\_ITEM\_LOC, then use the primary\_supp and primary\_cntry for the sourcing warehouse from ITEM\_LOC in order to retrieve the necessary values from ITEM\_SUPP\_COUNTRY and update the records **inner\_pack\_size, supp\_pack\_size, ti, hi, and supp\_lead\_time**.

Handles A<sub>2</sub> – Update the REPL\_ITEM\_LOC record's **inner\_pack\_size**, **supp\_pack\_size**, **ti**, **hi**, and **supp\_lead\_time** where the record is a store being sourced by a warehouse that is not on replenishment. Use the primary\_supp/primary\_chntry for the sourcing warehouse on ITEM\_LOC to link to ITEM\_SUPP\_COUNTRY to get the pack size/lead time info.

Handles B, C, D – Update the REPL\_ITEM\_LOC record's **round\_lvl**, **round\_to\_inner\_pct**, **round\_to\_case\_pct**, **round\_to\_layer\_pct**, and **round\_to\_pallet\_pct** where the record being updated on REPL\_ITEM\_LOC matches the ITEM\_SUPP\_COUNTRY\_LOC item/location/supplier/origin\_country\_id and the REPL\_ITEM\_LOC record is using direct to location or xdock/xlink supplier replenishment.

Handles B, C, D – Update the REPL\_ITEM\_LOC record's **unit\_cost** where the record being updated on REPL\_ITEM\_LOC matches the ITEM\_SUPP\_COUNTRY\_LOC item/location/supplier/origin\_country\_id and the REPL\_ITEM\_LOC record is using direct to location or xdock/xlink supplier replenishment. If direct to location replenishment is being used, use the location to get the cost. If xdock/xlink replenishment is being used, use the source\_wh to get the cost.

Handles A<sub>1</sub> – Update the REPL\_ITEM\_LOC record's **round\_lvl**, **round\_to\_inner\_pct**, **round\_to\_case\_pct**, **round\_to\_layer\_pct**, and **round\_to\_pallet\_pct** where the record is a store being sourced by a warehouse that is on replenishment. Use the primary\_repl\_supp/origin\_country\_id on REPL\_ITEM\_LOC for the sourcing warehouse to link to ITEM\_SUPP\_COUNTRY\_LOC to get the rounding info. However, if the item is being replenished in the form of a primary simple pack and that primary simple pack does not exist at the primary\_repl\_supp/origin\_country\_id/source\_wh combination on REPL\_ITEM\_LOC, then use the primary\_supp and primary\_chntry for the sourcing warehouse from ITEM\_LOC in order to retrieve the necessary values from ITEM\_SUPP\_COUNTRY\_LOC and update the records **round\_lvl**, **round\_to\_inner\_pct**, **round\_to\_case\_pct**, **round\_to\_layer\_pct**, and **round\_to\_pallet\_pct**.

Handles A<sub>2</sub> – Update the REPL\_ITEM\_LOC record's **round\_lvl**, **round\_to\_inner\_pct**, **round\_to\_case\_pct**, **round\_to\_layer\_pct**, and **round\_to\_pallet\_pct** where the record is a store being sourced by a warehouse that is not on replenishment. Use the primary\_repl\_supp/origin\_country\_id on ITEM\_LOC for the sourcing warehouse to link to ITEM\_SUPP\_COUNTRY\_LOC to get the rounding info.

If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, all updated information reflects the primary\_repl\_pack.

#### RILSW –

Input = item, location

B, C – no change – no source wh...

Handles D.

Handles A<sub>1</sub>.

Handles A<sub>2</sub>.

When an RILSW change type occurs, the program will flow through the RILP change type. See the section titled RILP for a description of the possible changes that can occur when the user changes their source\_wh.

#### RILSC –

Input = item, location

Handles B, C, D – Update the REPL\_ITEM\_LOC record's **inner\_pack\_size**, **supp\_pack\_size**, **ti**, **hi**, and **supp\_lead\_time**, **round\_lvl**, **round\_to\_inner\_pct**, **round\_to\_case\_pct**, **round\_to\_layer\_pct**, and **round\_to\_pallet\_pct** where the record being updated on REPL\_ITEM\_LOC matches the ITEM\_SUPP\_COUNTRY item/supplier/origin\_country\_id and it matches the ITEM\_SUPP\_COUNTRY\_LOC item/supplier/origin\_country\_id/loc.

Handles B, C, D – Update the REPL\_ITEM\_LOC record's **unit\_cost** where the record being updated on REPL\_ITEM\_LOC matches the the ITEM\_SUPP\_COUNTRY\_LOC item/supplier/origin\_country\_id/loc.

Handles A<sub>1</sub> – Update the REPL\_ITEM\_LOC record's **inner\_pack\_size**, **supp\_pack\_size**, **ti**, **hi**, and **supp\_lead\_time**, **round\_lvl**, **round\_to\_inner\_pct**, **round\_to\_case\_pct**, **round\_to\_layer\_pct**, and **round\_to\_pallet\_pct** where the record is a store being sourced by a warehouse that is on replenishment. Use the primary\_repl\_supp/origin\_country\_id on REPL\_ITEM\_LOC for the sourcing warehouse to link to ITEM\_SUPP\_COUNTRY and ITEM\_SUPP\_COUNTRY\_LOC to get the rounding info.

If a primary\_repl\_pack exists for the REPL\_ITEM\_LOC record, all updated information reflects the primary\_repl\_pack.

#### RIL –

Input = item, location

Perform IM, PQTY, ILSOM RILRC, and RILP logic.

#### LKITEM –

Input = item, location

Perform RIL logic on every location on replenishment for the item.

#### RILRC –

Input = item, location

Handles A<sub>1</sub>, A<sub>2</sub>, B, C, D – Update the REPL\_ITEM\_LOC record's **next\_review\_date** where the record being updated on REPL\_ITEM\_LOC. Use the review\_cycle and REPL\_DAY records to calculate the next\_review\_date. Assumes that the last\_review\_date is not a factor in setting next\_review\_date.

#### RILD –

Input = item, location

Perform ILSC logic.

**PQTY –**

Input = item

Handles A<sub>1</sub>, A<sub>2</sub>, B, C, D – Update all REPL\_ITEM\_LOC records where the primary\_pack\_no is the item on REPL\_ITEM\_LOC\_UPDATES. Get the new pack qty from PACKITEM.

**ILST –**

Input = item, location

Handles A<sub>1</sub>, A<sub>2</sub>, B, C, D – Delete the REPL\_ITEM\_LOC record where the item/location on REPL\_ITEM\_LOC matches the item/location on REPL\_ITEM\_LOC\_UPDATES OR where the item/source\_wh on REPL\_ITEM\_LOC matches the item/location on REPL\_ITEM\_LOC\_UPDATES and the primary\_repl\_supplier field on REPL\_ITEM\_LOC is not null (item must be active at both wh and store for xlink and xdock).

Handles A<sub>1</sub>, A<sub>2</sub>, B, C, D – Update the REPL\_ITEM\_LOC record where the primary\_pack\_no/location on REPL\_ITEM\_LOC matches the item/location on REPL\_ITEM\_LOC\_UPDATES – set primary\_pack\_no, primary\_pack\_qty equal to NULL.

For each record (the record) on REPL\_ITEM\_LOC where the primary\_pack\_no is updated to NULL, perform RIL logic.

For each record on REPL\_ITEM\_LOC where the item is equal to the item that was deleted and the source\_wh is equal to the location that was deleted, perform RILP logic.

**Scheduling Considerations**

Processing Cycle      *Phase 3*

Scheduling Diagram:

Pre-Processing:      *storeadd.pc, rplatupd.pc*

Post-Processing:      *prepost (rilmaint post), repladj.pc*

Threading Scheme:      *NONE*

**Restart/Recovery**

This program has a unique logical unit of work – item.

**Design Assumptions**

When setting next\_review\_date, program assumes that the last\_review\_date is not a factor in setting next\_review\_date.

## Automatic replenishment order approval [rplapprv]

### Design Overview

This program looks at all replenishment, vendor, and contract orders created during the nightly batch run. These orders are compared with any vendor minimums that may exist. Orders that do not meet the vendor minimums are either deleted or placed in worksheet status. A flag held at the supplier inventory management level (SUP\_INV\_MGMT.ORD\_PURGE\_IND), determines what action is taken on orders that fail minimums. Vendor generated orders are not subject to these minimum checks.

Vendor minimums can be held at the order, item, or location level. Order and location level minimums are held on the SUP\_INV\_MGMT table. There is a flag that determines if they are applied at the order level or at the location level. Vendor minimums at the sku level are held on the ITEM\_SUPP\_COUNTRY table.

When the SUP\_INV\_MGMT.ORD\_PURGE\_IND is 'N', a failure at any level causes the order to be placed in worksheet status. When the SUP\_INV\_MGMT.ORD\_PURGE\_IND is 'Y', a failure at the location level causes the offending location to be deleted, a failure at the sku level caused the offending sku to be deleted, and a failure at the order level caused the entire order to be deleted.

For any orders that fail vendor minimums when the SUP\_INV\_MGMT.ORD\_PURGE\_IND is 'Y', a record is written to the SUPS\_MIN\_FAIL table for reporting purposes. This table is purged at the beginning of the program to remove old records.

After order records are purged, any applicable deals, brackets and allowances are applied to the orders. Open to buy is then updated for any orders built in approved status. If any orders are contract orders, the contract amounts are updated as well to reflect any order record deletions.

This program runs both (multi-channel and non multi-channel) environments. If the order does not pass vendor minimum checks, Since the vendor minimum checks are performed for a physical wh, if the vendor minimum is not met for a physical location, all the virtual whs on the order within the physical wh will need to be removed along with associated allocations.

This program should run directly after the replenishment supcnstr program. It is important that this program runs before any other process affects the generated orders.

| TABLE            | INDEX | SELECT | INSERT | UPDATE | DELETE |
|------------------|-------|--------|--------|--------|--------|
| ORDHEAD          |       | Yes    | No     | Yes    | Yes    |
| ORDLOC           |       | Yes    | No     | No     | Yes    |
| ORDSKU           |       | Yes    | No     | No     | Yes    |
| DESC_LOOK        |       | Yes    | No     | No     | No     |
| ORD_INV_MGMT     |       | Yes    | No     | No     | Yes    |
| DEAL_CALC_QUEUE  |       | No     | Yes    | No     | Yes    |
| ITEM_SUPP_COUNTR |       | Yes    | No     | No     | No     |

|                 |  |     |     |     |     |
|-----------------|--|-----|-----|-----|-----|
| Y               |  |     |     |     |     |
| SUPS_MIN_FAIL   |  | No  | Yes | No  | Yes |
| ALLOC_HEADER    |  | Yes | No  | No  | Yes |
| ALLOC_DETAIL    |  | No  | No  | No  | Yes |
| CONTRACT_HEADER |  | Yes | No  | Yes | No  |
| CONTRACT_DETAIL |  | Yes | No  | Yes | No  |
| OTB             |  | No  | No  | Yes | No  |
| ORDSKU HTS      |  | Yes | No  | No  | Yes |
| REV_ORDERS      |  | No  | Yes | No  | NO  |
| PERIOD          |  | Yes | No  | No  | No  |
| SYSTEM_OPTIONS  |  | Yes | No  | No  | No  |

Re-run:

If this program terminates abnormally, restart without recovery.

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: Run after supcnstr.pc

Pre-Processing: N/A

Post-Processing: prepost.pc (prepost user/password rplprg post)

Threading Scheme:

### Restart Recovery

```
EXEC SQL DECLARE c_order CURSOR FOR
 SELECT oh.order_no,
 oh.purchase_type,
 oh.contract_no,
 oh.payment_method,
 oh.supplier,
 NVL(oim.min_cnstr_lvl,'N'),
 oh.import_order_ind,
 oh.dept,
 oim.ord_purge_ind,
 NVL(oim.ltl_approval_ind,'N'),
 NVL(oim.ltl_ind,'Y'),
 s.sup_status,
 s.bracket_costing_ind,
 DECODE(oh.orig_ind,5,1,0),
 ROWIDTOCHAR(oh.rowid),
 ROWIDTOCHAR(oim.rowid)
 FROM ordhead oh,
 ord_inv_mgmt oim,
 sups s
 WHERE oh.order_no = oim.order_no
 AND oh.supplier = s.supplier
 AND oh.status = 'W'
```

```
AND oim.ord_approve_ind = 'Y'
AND (oh.orig_ind = 0
 OR oh.orig_ind = 5)
AND oh.written_date = TO_DATE(:os_vdate,'YYYYMMDD')
AND oh.order_no > NVL(:os_order_no, -9999);
```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A



## Replenishment attribute update [rplatupd]

### Design Overview

Rplatupd.pc (REPL\_ATTRIB\_UPDATE\_DAILY) will execute the logic necessary to maintain replenishment attributes for an item list. (A user can update replenishment attributes for an individual item online, but processing an item list would take far too long to be acceptable.) When the Replenishment Attribute Maintenance form (replattr.fmb) is used to maintain the replenishment attributes for an item list, the form calls a package (rplattrb/s.pls) to write the changes to a set of temp tables (REPL\_ATTR\_UPDATE\_ITEM and REPL\_ATTR\_UPDATE\_LOC) listing the items, locations, and attributes to be changed. The batch program will read these tables and process the item location relationships using the information written to the table to determine what replenishment attributes for what locations have to be updated. Items are processed in order of sequence number. Each item list maintenance submittal is given a sequence number.

Replenishment attributes for each item/location are recorded in REPL\_ITEM\_LOC. Review cycle information is kept on the REPL\_DAY table.

This program is generally only run sporadically, but when it runs, it often has to process very large volumes (e.g., updating the attributes for all items in an entire group or division at all locations – millions of records). The previous version of rplatupd simply fetched records out of the staging tables and called the same PL/SQL package as the online form for each one. While this did keep online performance from being impaired, the program instead took an inordinate amount of time during the nightly batch run, essentially bringing processing to a complete halt for many hours. The problem is not only that updating many records takes longer than updating one, but that the different situations really require not just more time, but significantly different approaches. Therefore, in order to optimize performance, the validation and update logic has been moved out of PL/SQL and into the Pro\*C code itself.

| TABLE                 | INDEX | SELECT | INSERT | UPADATE | DELETE |
|-----------------------|-------|--------|--------|---------|--------|
| REPL_ATTR_UPDATE_ITEM | yes   | yes    | no     | no      | no     |
| REPL_ATTR_UPDATE_LOC  | yes   | yes    | no     | no      | no     |
| ITEM_LOC              | no    | yes    | no     | no      | no     |
| REPL_ITEM_LOC         |       | yes    | yes    | yes     | yes    |
| REPL_DAY              |       | no     | yes    | no      | yes    |
| ITEM_SEASONS          |       | yes    | yes    | no      | no     |
| SYSTEM_OPTIONS        |       | yes    | no     | no      | no     |
| ITEM_SUPP_COUNTRY     |       | yes    | no     | no      | no     |
| ITEM_MASTER           |       | yes    | no     | no      | no     |
| PACKITEM              |       | yes    | no     | no      | no     |
| DEPS                  |       | yes    | no     | no      | no     |
| REPL_ITEM_LOC_UPDATE  |       | no     | yes    | no      | no     |
| SUB_ITEMS_DETAIL      |       | yes    | no     | no      | no     |

**Scheduling Constraints**

Processing Cycle: Daily, Phase 3

Scheduling Diagram: This program should run in Phase 3 and be run before the replenishment batch programs, rpladj, rplext, reqext.

Pre-Processing: Truncate the MC\_REJECTIONS table (rplatupd\_pre)

Post-Processing: Truncate the REPL\_ATTR\_UPDATE\_ITEM and REPL\_ATTR\_UPDATE\_LOC tables (rplatupd\_post) Delete sub\_items from SUB\_ITEMS\_HEAD and SUB\_ITEMS\_DETAIL for item/locations that were deactivated. (sub\_items post)

Threading Scheme: Thread by store and warehouse (use v\_restart\_store\_wh view)

**Restart Recovery**

Driving cursor:

```
EXEC SQL DECLARE c_repl_items CURSOR FOR
SELECT rau_item.seq_no,
 im.item,
 rau_item.action,
 NVL(im.item_parent, ''),
 NVL(im.item_grandparent, ''),
 rau_item.repl_method_ind,
 rau_item.stock_cat,
 rau_item.repl_order_ctrl,
 rau_item.sourcing_wh,
 TO_CHAR(rau_item.activate_date, 'YYYYMMDD'),
 TO_CHAR(rau_item.deactivate_date, 'YYYYMMDD'),
 rau_item.pres_stock,
 rau_item.demo_stock,
 rau_item.repl_method,
 rau_item.min_stock,
 rau_item.max_stock,
 rau_item.incr_pct,
 rau_item.min_supply_days,
 rau_item.max_supply_days,
 rau_item.time_supply_horizon,
 rau_item.inv_selling_days,
 rau_item.service_level,
 rau_item.lost_sales_factor,
 rau_item.reject_store_ord_ind,
 rau_item.non_scaling_ind,
 rau_item.max_scale_value,
 rau_item.pickup_lead_time,
 rau_item.wh_lead_time,
 rau_item.terminal_stock_qty,
 rau_item.season_id,
 rau_item.phase_id,
 rau_item.supplier,
```

```

 rau_item.origin_country_id,
 rau_item.review_cycle,
 rau_item.monday_ind,
 rau_item.tuesday_ind,
 rau_item.wednesday_ind,
 rau_item.thursday_ind,
 rau_item.friday_ind,
 rau_item.saturday_ind,
 rau_item.sunday_ind,
 rau_item.unit_tolerance,
 rau_item.pct_tolerance,
 rau_item.default_pack_ind,
 rau_item.use_tolerance_ind,
 rau_loc.loc,
 rau_loc.loc_type
FROM v_restart_store_wh vrswh,
 repl_attr_update_loc rau_loc,
 repl_attr_update_item rau_item,
 item_master im
WHERE rau_loc.seq_no = rau_item.seq_no
AND im.item_level = im.tran_level
AND (rau_item.item = im.item
 OR rau_item.item = im.item_parent
 OR rau_item.item = im.item_grandparent)
AND vrswh.driver_name = :os_restart_driver_name
AND vrswh.num_threads = :oi_restart_num_threads
AND vrswh.thread_val = :oi_restart_thread_val
AND vrswh.driver_value = rau_loc.loc
AND (rau_item.seq_no > NVL(:os_restart_sequence, -999) OR
 (rau_item.seq_no = :os_restart_sequence AND
 (rau_item.item > :os_restart_item OR
 (rau_item.item = :os_restart_item AND
 (rau_loc.loc > :os_restart_loc)))))
ORDER BY rau_item.seq_no, rau_item.item, rau_loc.loc; /* order by seq,item,loc */

```

### Shared Modules

ITEMLIST\_MC\_REJECTS\_SQL.INSERT\_REJECTS –write rejected records to mc\_rejections table for later reporting.

### REPL\_ATTRIBUTE\_MAINTENANCE\_SQL.GET\_ITEM\_TYPE

This package will no longer be called from the program, but the validation logic of these two modules must be kept in sync.

FETCH\_STRUCT – s\_dc\_fetch

Will hold item, location, and all replenishment attributes fetched from  
REPL\_ATTR\_UPDATE\_ITEM and REPL\_ATTR\_UPDATE\_LOC

ATTRIBUTE\_STRUCT – s\_update, s\_activate

Will hold an item, location, and all of its replenishment attributes. In addition, it should hold a rowid

(to speed up updates). The structure will be used to define the arrays used for both inserts and updates to REPL\_ITEM\_LOC.

REPL\_DAY\_STRUCT – s\_insert\_days, s\_delete\_days

This structure will be used for inserts to the REPL\_DAY table. It should contain the item, location,

location type, and weekday. This array should be sized to seven times the commit max counter, since each

item-location combination being fetched could potentially be replenished every day of the week. *Because of the need for this larger array, the commit\_max\_ctr setting on restart\_control for this program should be set with caution, since Oracle has a 32k limit on bind arrays.*

The same structure type can be used to define the arrays for deletes from REPL\_DAY, but that array should not hold the weekday, as any delete from this table will clear out all records for an item-location combination.

REPL\_ITEM\_LOC\_UPDATES\_ARRAY – s\_ril\_updates

This structure will be used for inserts into REPL\_ITEM\_LOC\_UPDATES.

### Function Level Description

#### init()

- initialize restart/recovery
- check if contracting is in use
- find what day of the week Oracle thinks Sunday is

#### process()

The controller for most of the processing in the program, this function opens and fetches from the driving cursor according to Retek standards, using array processing. For each item-location coming through (i.e., each record fetched), this function will call the appropriate validation function based on the requested action. If validation passes, information should be passed into the function(s) to fill insert, update, and/or delete arrays. A note on updating attributes: since records on REPL\_DAY cannot be updated (weekday is part of the primary key), the old days need to be cleared off and new ones inserted when doing an update. This should only happen, of course, when the user actually wants to change the days of replenishment (update\_repl\_days\_ind = 'Y').

When the fetch array has been fully processed, post functions are called and the restart commit logic is called.

#### activate\_item()

Calls validate\_activate. If the item was rejected, call insert\_rejection; otherwise call fill\_activate\_array and fill\_insert\_days\_array to populate arrays. Calls fill\_ril\_updates\_array to insert into REPL\_ITEM\_LOC\_UPDATES.

It also gets the item\_season\_seq\_no for the record's item/season/phase combination.

#### post activations

Insert records into repl\_item\_loc.

#### update\_item

Call `validate_update`. If the record fails validation, call `insert_rejection` ; otherwise put it into the update arrays with `fill_update_array` and `fill_insert_days_array`.

It also gets the `item_season_seq_no` for the record's item/season/phase combination. Calls `fill_ril_updates_array` if the supplier or origin\_country is being updated.

#### deactivate\_item

Fill the deactivation array, after checking that the record exists on `repl_item_loc`.

#### post\_deactivations

Delete given rows from `repl_item_loc` table.

#### post\_day\_changes

Insert changes into the `repl_day` table

#### get\_item\_info

Queries the item tables to get the forecast indicator for the item (which will be used during validation for activation and update). This function also queries DEPS to get the purchase type of the item's department to determine whether or not the item is on consignment or not. If an item is on consignment, it cannot be replenished, since the supplier determines what goes on the shelves. The function also checks to make sure that the sourcing warehouse is valid.

`Get_item_info()` also queries `ITEM_SUPPLIER` and `ITEM_SUPP_COUNTRY` to verify that the given supplier and origin country (if any) are valid. `Get_item_info` checks the `default_pack_ind`. If the `default_pack_ind` = 'Y', then it gets the `primary_cost_pack` from `ITEM_LOC` and checks that it is active at the location.

#### get\_itemloc\_info

Queries the `REPL_ITEM_LOC` table to find the current replenishment attributes for the current item-location combination. If no record is found, the item is not on replenishment at that location (this is not necessarily an error). The rowid should also be fetched to facilitate updates and deletes. Queries the `ITEM_LOC` table to get the clearance indicator. Also checks the `SUB_ITEMS_DETAIL` table to check if the item is a substitute item.

#### validate\_activate

If an item-location is already on replenishment, the function should return a non-fatal error so that the record is skipped. If the item-location meets any of the following criteria, `insert_rejection()` should be called with the appropriate rejection reason code (see table `MC_REJECTION_REASONS`), and a non-fatal error should be returned so the record is not written to the activate array:

- The item is not supplied by the specified replenishment supplier/origin country.
- The item is on consignment.
- The item is a substitute item.
- The replenishment method requires forecast information (Time Supply or Dynamic), but the item is not forecastable.
- It is on clearance and contracting is disabled (this check only applies to stores, since an item cannot be on clearance at a warehouse).
- The item is not stocked at the specified sourcing warehouse (this check only applies to stores, a warehouse is never sourced by another warehouse).

If a seasonal method of replenishment is being used, this function also calls `validate_item_seasons()` to ensure that the item-season-phase relationship exists, or creates it if it doesn't.

If the item-location combination passes all the validation criteria, the function will return successfully.

#### validate\_update

When the user updates replenishment attributes, she may not want to change all of them, and so will leave certain fields blank. These will get written to REPL\_ATTR\_UPDATE\_ITEM as NULL values. Therefore, any column coming out of the driving cursor as NULL is seen as representing a “no-change”. This requires careful monitoring of indicator variables during validation and assignment.

If an item-location is not already on replenishment, the function should return a non-fatal error so that the record is skipped. If the item-location meets any of the following criteria, insert\_rejection() should be called with the appropriate rejection reason code (see table MC\_REJECTION\_REASONS), and a non-fatal error should be returned so the record is not written to the update array:

- The item is not supplied by the specified replenishment supplier/origin country.
- The replenishment method requires forecast information (Time Supply or Dynamic), but the item is not forecastable.
- The item is not stocked at the specified sourcing warehouse (this check only applies to stores, a warehouse is never sourced by another warehouse).
- If the activation date is being changed but the deactivation date is not, and the new activation date is later than the old deactivation date.
- If the deactivation date is being changed but the activation date is not, and the new deactivation date is earlier than the old activation date.
- If the minimum stock quantity is being changed but the maximum is not, and the new minimum is greater than the old maximum.
- If the maximum stock quantity is being changed but the minimum is not, and the new maximum is less than the old minimum.
- If the minimum number of supply days is being changed but the maximum is not, and the new minimum is greater than the old maximum.
- If the maximum number of supply days is being changed but the minimum is not, and the new maximum is less than the old minimum.
- The order control Buyer worksheet is allowed with the stock category and location type.

If a seasonal method of replenishment is being used, this function also calls validate\_item\_seasons() to ensure that the item-season-phase relationship exists, or creates it if it doesn't.

If the item-location combination passes all the validation criteria, the function will return successfully.

#### fill\_activate\_array

Assigns all replenishment attributes for the current record to the array to be used in inserts to REPL\_ITEM\_LOC.

#### fill\_update\_array

Assigns the replenishment attributes for the current record to the array to be used in updating REPL\_ITEM\_LOC. If the replenishment method is changing (REPL\_ATTR\_UPDATE\_ITEM.repl\_method\_ind = 'Y'), then all replenishment-method-specific attributes must be written as-is to the arrays, whether they are NULL or not. These

attributes are: `repl_method`, `min_stock`, `max_stock`, `incr_pct`, `min_supply_days`, `max_supply_days`, `time_supply_horizon`, `inv_selling_days`, `service_level`, `lost_sales_factor`, `reject_store_ord`, `terminal_stock_quantity`, `season_id`, `phase_id`, and `primary_pack_no`. If the replenishment method is not changing, these attributes should be left unchanged if they are NULL on `REPL_ATTR_UPDATE_ITEM`; that is, the old value from `REPL_ITEM_LOC` should be written to the array rather than the new NULL value to keep the actual record on the table from being inadvertently overwritten. All other attributes should be left unchanged if they are NULL on `REPL_ATTR_UPDATE_ITEM`.

#### fill\_insert\_days\_array

Assigns the item, location, location type, and weekday to the array to be used in inserting to `REPL_DAY`. A record needs to be added for each weekday that has been marked as a review day for the current item-location combination.

#### fill\_delete\_days\_array

Assigns the item and location to the array to be used in deleting from `REPL_DAY`. The weekday does not need to be added to this array, since the program will simply delete all weekdays for a given item-location combination.

#### fill\_ril\_updates\_array

Assigns the item, supplier, origin\_country, location, and loc\_type to the array to be used in inserting into `REPL_ITEM_LOC_UPDATES`. A record will be added for each item/location that is activated or item/location where the supplier or origin\_country is being updated.

#### open\_dc

Open the driving cursor.

#### fetch\_dc

Fetch the driving cursor.

#### post\_updates

Performs arrayed update of `REPL_ITEM_LOC`.

#### post\_repl\_item\_loc\_updates

Performs arrayed insert of `REPL_ITEM_LOC_UPDATES`

#### validate\_item\_seasons

If a Seasonal method of replenishment (either Dynamic or Time-Supply) is being used, a record on the `ITEM_SEASONS` table must exist to maintain the foreign key dependency from `REPL_ITEM_LOC`. This function queries `ITEM_SEASONS` to ensure that a relationship exists. If it doesn't, a record should be inserted. No error should be raised on a `DUP_VAL_FOUND` occurrence, since it is possible that another thread of the program could be trying to write the same record, as threading is done by location rather than item. It is not possible to perform this insert as an arrayed SQL statement, since a `DUP_VAL_FOUND` error would stop the rest of the array from being posted.

#### insert\_rejection

This function calls the stored PL/SQL procedure `ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS`, passing in the error information passed into this function by the caller. It should take in a rejection reason and up to three other strings (for additional error message text) as parameters.

size\_arrays

Allocates memory for all processing arrays.

final()

call restart\_close to finish restart/recovery

#### **Technical Issues**

N/A



# Replenishment build [rplbld]

## Design Overview

Rplbld builds RMS orders from Recommended Order Quantities (*ROQ*) generated by rplext.pc and ibcalc.pc. Cntrprss.pc associates contracts with the ROQs created by rplext.pc. These ROQs are placed on a temporary table (ORD\_TEMP or IB\_RESULTS) by rplext.pc and ibcalc.pc. All records on ORD\_TEMP/IB\_RESULTS are processed by rplbld each night. These ORD\_TEMP/IB\_RESULTS records are placed into logical groups, and a RMS order is created for each logical group.

In order to be placed in the same order group, the item/location ROQs from ORD\_TEMP/IB\_RESULTS must share a common supplier, have the same order\_status ('W'orksheet or 'A'pproved), and be on the same contract (or not be associated with a contract). Depending on flags on the ORD\_INV\_MGMT table, two other criteria can be used for splitting order groups. First, if the INV\_MGMT\_LVL is set to 'D'ept, only items in a single department are allowed in an ordering group. Second, the SINGLE\_LOC\_IND can be set to 'Y'es. If this is the case, only one location is allowed per ordering group. Finally, sku may only exist in an ordering group with a single origin country. When an item/loc ROQ ORD\_TEMP/IB\_RESULTS record is encountered with a different origin country than the one it exists with in the current ordering group, it is placed in a different ordering group.

To assist the recalculation and order scaling processes of replenishment ROQs, the REPL\_RESULTS record, associated with the ORD\_TEMP being processed, is updated with the ORDER\_NO and ALLOC\_NO that the ORD\_TEMP record was placed with. IB\_RESULTS is also updated with the ORDER\_NO.

## Stored Procedures / Shared Modules (Maintainability)

### ORDLIB.H

- Data structure and global variable definition
- Set\_golbals()
- Get\_sup\_info()
- Add\_to\_order()
- Do\_inserts()
- Kill\_order()

## Function Level Description

### Driving cursor:

```
EXEC SQL DECLARE c_driver CURSOR FOR
 SELECT NVL(rr.source_type,'R'),
 ot.supplier,
 ot.dept,
 ot.order_status,
 ot.item,
 ot.pack_ind,
 ot.location,
 wh.physical_wh,
 NVL(wh.repl_wh_link, ot.location),
```

```

ot.loc_type,
ot.due_ind,
ot.crossdock_ind,
NVL(ot.crossdock_store, -1),
NVL(ot.contract_no, -1),
NVL(ot.contract_type, 'N'),
ot.qty_ordered,
NVL(rr.last_rounded_qty, 0),
NVL(rr.last_grp_rounded_qty, 0),
ot.origin_country_id,
ot.unit_cost,
NVL(rr.suppl_lead_time, -1),
NVL(rr.pickup_lead_time, -1),
NVL(ot.suppl_pack_size, 1),
rr.estimated_stock_out,
rr.accepted_stock_out,
NVL(rr.non_scaling_ind, 'N'),
rr.tsfc_po_link_no,
0 , /* l_ib_days_to_event */
ROWIDTOCHAR(rr.rowid),
NULL,
NULL
FROM ord_temp ot,
repl_results rr,
wh,
v_restart_supplier rv
/* need the outer join to repl_results because a
record may not be there for the WH total
portion of a xdock */
WHERE ot.ord_temp_seq_no = rr.ord_temp_seq_no(+)
AND ot.location = wh.wh(+)
AND rv.driver_value = ot.supplier
AND rv.num_threads = :pi_restart_num_threads
AND rv.thread_val = :pi_restart_thread_val
AND (ot.supplier > NVL(:ps_restart_supplier, -999) OR
(ot.supplier = :ps_restart_supplier AND
(NVL(ot.contract_no, -1) > :ps_restart_contract OR
(NVL(ot.contract_no, -1) = :ps_restart_contract AND
(ot.order_status > :ps_restart_status))))))
UNION ALL
SELECT ir.source_type,
ir.supplier,
ir.dept,
DECODE(ir.ib_order_ctrl, 'S', 'W', ir.ib_order_ctrl),
ir.item,
ir.item_type,
ir.location,
ir.physical_wh,
NVL(ir.repl_wh_link, ir.location),
ir.loc_type,
'Y', /* due ind */

```

```

 'N', /* xdock ind */
 -1, /* xdock store */
 -1, /* contract_no */
 'N', /* contract_type */
 ir.order_roq,
 NVL(ir.last_rounded_qty, 0),
 NVL(ir.last_grp_rounded_qty, 0),
 ir.origin_country_id,
 ir.unit_cost,
 NVL(ir.supp_lead_time, -1),
 NVL(ir.pickup_lead_time, -1),
 ir.case_size,
 0, /* eso */
 0, /* aso */
 'Y', /* non-scaling ind */
 -1, /* tsf_po_link */
 ir.days_to_event,
 NULL,
 ROWIDTOCHAR(ir.rowid),
 NULL
FROM ib_results ir,
 v_restart_supplier rv
WHERE ir.ib_order_ctrl in ('A', 'S')
AND ir.status = 'W'
AND ir.create_date = TO_DATE(:ps_vdate, 'YYYYMMDD')
AND rv.driver_value = ir.supplier
AND rv.num_threads = :pi_restart_num_threads
AND rv.thread_val = :pi_restart_thread_val
AND (ir.supplier > NVL(:ps_restart_supplier, -999) OR
 (ir.supplier = :ps_restart_supplier AND
 (-1 > :ps_restart_contract OR
 (-1 = :ps_restart_contract AND
 (ir.ib_order_ctrl > :ps_restart_status))))))
/* See the comments before the do_inserts() function before
 changing the order by clauses. The first three elements and
 for rplbld order splitting and restart recovery. The last
 five elements in this clause are to ensure do_inserts works. */
ORDER BY 2, /* supplier */
 14, /* contract */
 4, /* order_status */
 1 desc, /* source_type */
 5, /* item */
 7, /* location */
 12, /* xdock ind */
 13; /* crossdock_store */

```

### Functions:

#### *main()*

The standard Retek main function, this calls *init()*, *process()*, and posts to the daily log files.

#### *init()*

This function initializes the restart/recovery API and fetches global options and variables.

*Process()*

This is the main control function of the program. It contains the driving cursor. The driving cursor is fetched using array processing, it is fetched in groups of the commit max counter. The records brought back by the driving cursor are looped through individually. For each record, the `get_sup_info()` and `get_contract_info()` functions are fetched to fill out additional information that can not be supplied by the driving cursor. Once all the need information is gathered, a decision is made whether to add the current record to the linked lists or to process the linked lists and start over. This decision is made by calling the `switching_variable_change()` function. This function will return one when the conditions dictate a flush of the linked lists. The *flushing triggers* are a change in supplier, order status, contract number, dept (depends on dept level order flag), and location (depends on single loc flag). If the verdict is a flush of the linked lists, the `ORDLIB.DO_INSERTS()` function is called to flush the linked lists. In addition, `ORDLIB.ROUND_PO()` is called for each order created. It then clears out the linked lists and frees their memory. For each order created a counter is incremented. When this counter becomes greater than the commit max counter, a commit is done by calling `retek_force_commit()`. As the logical unit or work is supplier, order status, and contract no, The commit can only be done when one of these changes. Finally, `ORDLIB.ADD_TO_ORDER()` is called to add the current record to the linked lists. If the linked lists were just flushed, or the first record is being processed, the linked lists could be empty.

*Switching\_variables\_change()*

This function determines if the supplier, contract, or order status has changed since the last call to it. If no change has occurred, zero is returned. If a change has taken place one is returned.

*Size\_driv\_cur\_struct()*

Allocates memory for the structure used in the array fetch of the driving cursor. The structure is sized to the commit max counter.

*free\_driv\_cur\_struct()*

Give back memory allocated for the array fetch of the driving cursor.

*Copy\_driv\_cur\_rec()*

This function copies a single record of the driving cursor fetch structure to a single element holder structure (process structure). The single element structure is then used to gather additional needed information, and is passed to the linked list building functions. The single element structure allows the rest of the program to not use any array subscripts pertaining to the driving cursor for loop.

*Get\_contract\_info()*

This function retrieves contract information for the passed in process structure. It uses local static variables to avoid unnecessary database hits. If a database hit is required, the contract terms, approval indicator, and currency code are brought from `CONTRACT_HEADER` for the passed in contract number (in the process structure).

**Input Specifications****'Table-To-Table'**

Pulls ROQs from `ORD_TEMP`, `REPL_RESULTS`, and `IB_RESULTS`.

**Output Specifications****'Table-To-Table'**

See createordlib.h create\_po.pc design.

**Scheduling Considerations**

Processing Cycle      Phase III

Scheduling Diagram:

Pre-Processing:      rplext.pc, cntrprss.pc(if contracting is being used),  
prepost.pc (prepost user/password rplbld pre)

Post-Processing:      prepost (prepost user/password rplbld post) then supcnstr.pc

Threading Scheme:      SUPPLIER

**Restart/Recovery**

The logical unit of work is supplier, contract, and order status. The driving cursor is ordered by supplier, contract, order status, source\_type, sku, location, crossdock indicator, and crossdock store. The first three elements are for restart recovery and order splitting. The next five are for performance and to ensure crossdock posting performs correctly. When the supplier, contract, or order status changes, there is an opportunity to commit. At that point the number of orders created since the last commit is compared with the commit max counter. If the number of orders is greater than the commit\_max\_counter retek\_force\_commit is called.

## Vendor replenishment extraction [rplex]

### Design Overview

Rplex (Vendor Replenishment Extraction) is the driving program for the replenishment process. It cycles through every item-location combination that is ready to be reviewed on the current day, and calculates the quantity of the item that needs to be ordered to the location. The program then writes these order line items to ORD\_TEMP and REPL\_RESULTS for processing by cntrprss and rplbld.

The logic of this program is determined mainly by the stock category of the item-location combination. If the stock category is *Direct to Store*, the process is fairly straightforward. The ROQ is calculated, and this quantity is then written to the aforementioned tables for ordering.

If it is *Crossdocked*, the ROQ calculations are handled similarly to the Direct to Store category. However, when crossdocking occurs, the item must be ordered to a warehouse before being allocated to the store. This means that the total number of items being ordered must be tracked across all stores sharing the same source warehouse, so that a *summary record* can be written to ORD\_TEMP to connect all the stores and create one order to the warehouse to be allocated from there.

Similarly, if the stock category is *WH/Cross Link*, the ROQ and distribution calculations are handled much like the Direct to Store category. However, when processing WH/Cross Link item-store records that have been generated in reqext, the item must also be ordered to a warehouse before being allocated to the store in order to fulfill the WH/Cross Link transfer quantity generated in reqext. Much like crossdocking, this means that the total number of items being ordered must be tracked across all stores sharing the same source warehouse, PO-Linked Transfer Number and replenishment order control, so that a *summary record* can be written to ORD\_TEMP to connect all the item-stores and create one order to the warehouse to be allocated from there.

No item-store combinations with a stock category of *Warehouse Stocked* are processed by this program. Since these stores are supplied via transfers from the source warehouse rather than purchase orders, they are processed in a separate program (reqext). However, orders must still be built so that the source warehouse has stock to supply the stores with. If the replenishment method for the item-warehouse combination is non-forecasted (*Constant*, *Floating-Point*, or *Min-Max*), the process is the same as Direct to Store. However, if the method does involve forecasting (*Dynamic* or *Time-Supply*), then the ROQ for the warehouse is based on the predicted ROQ (or *need*) of the item (or its associated simple pack) at all stores that are supplied by the warehouse.

Once the order quantity for an item (or simple pack) has been calculated, an order line item is written to the ORD\_TEMP table if **1)** the actual quantity to order is greater than zero, and **2)** the replenishment order control indicator for the item-location combination is either Automatic or Semi-Automatic. If it is Manual or Buyer Worksheet, a record will be written to another table (REPL\_RESULTS) for reporting purposes. If the system-level Replenishment Results indicator is set to "Yes", then all order line items being written to ORD\_TEMP will also be written to REPL\_RESULTS. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL\_RESULTS, *even if the quantity to order is zero*. Crossdock summary records are never written to REPL\_RESULTS, since they are only used internally within the replenishment batch process.

### Scheduling Constraints

|                     |                                                                                                                                                                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Processing Cycle    | Phase III                                                                                                                                                                                                                                                                    |
| Scheduling Diagram: | Pccext, rplatupd, rilmaint, rpladj, reqext and entrordb need to run before rplext to make sure that replenishment attributes and stock information are all up to date. If contracting is being used, cntrprss should run after rplext; otherwise, run ibcxpl, ibcalc rplbld. |
| Pre-Processing:     | The ORD_TEMP table is truncated in the pre-processing.                                                                                                                                                                                                                       |
| Post-Processing:    | N/A                                                                                                                                                                                                                                                                          |
| Threading Scheme:   | DEPT                                                                                                                                                                                                                                                                         |

### Restart Recovery

The logical unit of work is item, supplier. The driving cursor is ordered by item, supplier, origin country, sourcing warehouse, and order control indicator. The first two elements are for restart recovery. The second two have to do with crossdocked orders. Because a summary record has to be written for a crossdock, totals must be kept across all locations an item is being ordered to. When the sourcing warehouse changes, we know we have reached the end of this particular crossdock and can post the summary record. The order control indicator is needed since if some crossdocks are written to be Semi-automatically approved (written in Worksheet status) and others to be Automatically approved, two separate orders (and thus two separate summaries) need to be written.

### Program Flow

N/A

### Shared Modules

GET\_REPL\_ORDER\_QTY\_SQL.REPL\_METHOD: Stored PL/SQL procedure for calculating the ROQ of an item at a location.

REPL\_OLT\_SQL.GET\_ITEM\_LOC\_REVIEW\_TIME: Stored PL/SQL procedure for determining the time between reviews for an item-location combination. This information is used in GET\_REPL\_ORDER\_QTY and is posted to ORD\_TEMP and REPL\_RESULTS.

RMS\_ROUND\_TO\_PACKSIZE: Shared C function (see rpl.h) used in rounding an item's quantity up to the size of a simple pack, or for rounding an order quantity up to a receivable pack size. This function is called when ordering a primary simple pack in the build\_pack\_dist\_struct() function, or when generating crossdock order quantities for individual stores.

### Data Structures

*repl\_info\_struct*: Holds information fetched in from the driving cursor, as well as storing variables to be used in other calculations.

*ord\_temp\_struct*: Used to buffer inserts to the ORD\_TEMP table.

*repl\_results\_struct*: Used to buffer inserts to the REPL\_RESULTS table.

*pack\_dist\_struct*: Used for holding information about the simple pack that an item belongs to, or if an item has no simple packs associated with it, this structure will simply hold information for the singular item itself.

*xdock\_info\_struct*: Holds information to be used in placing the summary record of a crossdocked order.

*repl\_update\_struct*: Holds information to be used in updating the REPL\_ITEM\_LOC table for implicit restart-recovery purposes and for storing the last ROQ generated for investment buy functionality.

*domain\_struct*: Used to cache forecasting domain information.

### Function Level Descriptions

#### *main()*

The standard Retek main function, this calls *init()*, *process()*, *final()* and posts to the daily log files.

#### *init()*

This function initializes the restart/recovery API and fetches global options and variables.

#### *driving\_cursor()*

Opens, fetches or closes the driving cursor. This is a support function for *process()*.

#### *process()*

Controlling function for processing. Sizes most of the data structures, fetches data from the driving cursor (by calling *driving\_cursor()*), copies each record returned by the driving cursor into a structure for processing (by calling *copy\_repl\_info()*), calls *replenish\_item()* to do most of the work for each record returned, posts orders when necessary and handles restart/recovery.

#### *replenish\_item()*

The main controlling function for replenishment calculations.

This function first calls *get\_vendor\_line\_info* in order to return all relevant supplier inventory management records for the item-location being processed.

If the location being processed is a warehouse *and* the warehouse is using a forecasted method of replenishment (either Dynamic or Time Supply), the calculations for determining ROQ are based on the stores that this warehouse is a source for, and *replenish\_to\_wh()* is called to find this information and place the orders.

Otherwise, this function calls *get\_olts\_and\_review\_time* to get the applicable review and lead times for the item-location in order to pass them into *get\_repl\_order\_qty()*, which calculates the raw ROQ for the item-location combination based upon the replenishment method and lead and review times being used. The function then calls *build\_pack\_dist\_struct()* in order to associate the raw ROQ with either a single item or a simple pack, depending upon what the location is set up to order. If the current record is part of a crossdock order, the totals are updated, and when all



the locations for the order have been processed, a summary record is posted. Finally, the order line item is placed by calling `place_item_orders()`.

#### *build\_pack\_dist\_struct()*

If the item is not associated with a simple pack (determined by the simple pack number on the `REPL_ITEM_LOC` table), the item and its ROQ will simply be written to the distribution structure by itself. If a primary simple pack is defined on the `REPL_ITEM_LOC` table, that pack will be written to the distribution structure after rounding the quantity up to an orderable pack size.

#### *replenish\_to\_wh()*

Replenishes an item to a warehouse using a forecasted method of replenishment that will in turn be used as a replenishment source for stores (see `retext`).

This function fetches replenishment information for the current item at all stores that are supplied by the current warehouse. It then finds the forecasted ROQ for each item-store, calls `build_pack_dist_struct()` in order to write associate that ROQ with either a single item or primary simple pack depending upon what the store has been set up to order. In this manner, it behaves similar to the main processing loop, but rather than calculating quantity to order, it is calculating the future 'need' for the item at the store. As each store is processed, a running total of the need for each item or simple pack is kept (by the `update_wh_need()` function). When all the stores have been processed, then the total need for each item or simple pack is used in calculating the warehouse's actual ROQ of that pack (in `get_repl_order_qty()`).

#### *update\_wh\_need()*

Updates the total need of an item or simple pack at a warehouse over all stores supplied by that warehouse. This is a support function for `replenish_to_wh()`.

#### *update\_xdock\_totals()*

Writes the initial warehouse record for the first store being processed and updates the total quantity to order for all stores on a crossdock order if more than one store is being crossdocked. This information is used in placing the summary record of a crossdock order. It should be noted that the order quantities to the stores will be rounded to the nearest pack size, whereas the summary quantity for the warehouse will not be rounded.

#### *place\_xdock\_wh\_order()*

If a crossdock order is being created, a record must be written to `ORD_TEMP` to order the item or simple pack to the warehouse for allocation to stores. This function adds this summary record to the `ORD_TEMP` buffer (but not to the `REPL_RESULTS` buffer).

#### *place\_item\_orders()*

Adds a record to the appropriate structure(s) for insert to `ORD_TEMP` and/or `REPL_RESULTS`. The conditions for writing records on the `ORD_TEMP` is when the following were met: the order control must be 'A'utomatic or 'S'emi-Automatic, the `SUP_INV_MGMT.due_order_process_ind` is 'Y', or the `SUP_INV_MGMT.due_order_process_ind` is 'N' and ROQ package `due_ind` is 'Y'. Records will be written to the `REPL_RESULTS` buffer if the system-level Replenishment Results option is set to "Yes", or `SUP_INV_MGMT.due_order_process_ind` is 'Y' or ROQ package `due_ind` is 'Y'.

If the order control is 'Manual' or 'Buyer Worksheet', records will not be written to the ORD\_TEMP buffer, however records will be written to the REPL\_RESULTS buffer if the system-level Replenishment Results option is set to "Yes", or SUP\_INV\_MGMT.due\_order\_process\_ind is 'Y' or ROQ package due\_ind is 'Y'.

If one or both structures fill up, they will be posted to the database and reset to be refilled.

#### *post\_orders()*

Actually writes order information to the database, inserting to ORD\_TEMP and REPL\_RESULTS.

#### *update\_review\_and\_delivery\_dates()*

Updates the replenishment information table (REPL\_ITEM\_LOC) to reflect the fact that item-location combinations have just been evaluated.

#### *get\_repl\_order\_qty()*

Gets the recommended order quantity (ROQ) and other information for an item-location combination. Essentially just a wrapper for the REPLENISHMENT\_SQL.GET\_ITEM\_LOC\_REVIEW\_TIME and GET\_REPL\_ORDER\_QTY\_SQL.REPL\_METHOD stored PL/SQL procedures.

#### *get\_domain\_info()*

The GET\_REPL\_ORDER\_QTY\_SQL stored procedure requires a forecasting domain when performing calculations for item-location combinations using a forecasted method of replenishment (Dynamic or Time Supply). For a given department, class or subclass (depending on the system-level Domain Level option), this function finds the associated forecasting domain. The first time this is called, it will call build\_domain\_cache() to pull all domain information for this thread into an array, and return the desired domain. Subsequent calls will only query the array.

#### *build\_domain\_cache()*

Populates a structure with information on department/class/subclass/domain relationships. Because the program is threaded by department, the function will only pull domains associated with departments, classes, or subclass associated with the current thread.

#### *copy\_repl\_info()*

Copies replenishment information from the fetch array into a more convenient structure for evaluation and processing.

#### *copy\_pack\_to\_repl()*

Copies replenishment and simple pack information into a repl\_info\_struct and a pack distribution structure for calculating ROQ and building orders for a warehouse. This is a support function for replenish\_to\_wh().

#### *Get\_vendor\_line\_info()*

Get vendor line constraint information from SUPS\_INV\_MGMT table. Copy them to repl\_info\_struct. They are needed by GET\_REPL\_ORDER\_QTY\_SQL for getting an item-locations ROQ.

#### *size\_repl\_info\_struct()*

Allocates memory for the elements of a replenishment information structure.

#### *size\_ord\_temp\_struct()*

Allocates memory for the elements of an ORD\_TEMP insert structure.

*size\_repl\_results\_struct()*

Allocates memory for the elements of a REPL\_RESULTS insert structure.

*size\_pack\_dist\_struct()*

Allocates memory for the elements of a simple pack distribution structure.

*size\_xdock\_info\_struct()*

Allocates memory for the elements of a crossdock totals structure.

*final()*

The normal Retek final function, this closes down the restart/recovery API.

### Database Interaction

#### Tables Selected From:

DOMAIN\_CLASS  
DOMAIN\_DEPT  
DOMAIN\_SUBCLASS  
REPL\_DAY  
REPL\_ITEM\_LOC  
STORE  
SYSTEM\_OPTIONS  
WH  
SUPS  
V\_RESTART\_DEPT (*View*)

#### Tables Inserted To:

ORD\_TEMP  
REPL\_RESULTS

#### Tables Updated:

REPL\_ITEM\_LOC

### I/O Specification

N/A

### Technical Issues

N/A

# Replenishment purge [rplprg]

## Design Overview

The replenishment extraction programs (rplext, reqext) write a number of records to REPL\_RESULTS, the store order upload program (soupld.pc) populates STORE\_ORDERS, the investment buy process writes records to IB\_RESULTS and the Buyer Worksheet Form populates BUYER\_WKSHT\_MANUAL. These tables hold information that is relevant to replenishment processes. Over time, records on these tables become unneeded and should be cleared out. The Replenishment Purge Program goes through these tables and clears out those records that are older than a predetermined number of days.

## Scheduling Constraints

Processing Cycle: No constraints; Phase V or ad-hoc.

Scheduling Diagram: N/A.

Pre-Processing: N/A.

Post-Processing: N/A.

Threading Scheme: N/A.

## Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Because this program performs only deletes, there is no need for restart/recovery or multithreading, and there is no driving cursor. However, this program still needs an entry on RESTART\_CONTROL to determine the number of records to be deleted between commits.

## Program Flow

Structure Chart

N/A.

## Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage  
N/A.

## Function Level Description

All database interactions required and error handling considerations

main

Standard Retek main.

init

This function fetches various threshold dates. These dates are the current processing date (PERIOD.vdate) minus SYSTEM\_OPTIONS.repl\_results\_purge\_days, PERIOD.vdate minus

SYSTEM\_OPTIONS.store\_orders\_purge\_days and SYSTEM\_OPTIONS.ib\_results\_purge\_days. This function should also fetch the commit\_max\_ctr from RESTART\_CONTROL to determine how many records can be deleted between commits.

process

This program will, in four different loops, delete from REPL\_RESULTS, BUYER\_WKSHT\_MANUAL, STORE\_ORDERS and IB\_RESULTS where the repl\_date, create\_date, need\_date and create\_date are less than their associated threshold date and the rownum is less than the commit counter. The BUYER\_WKSHT\_MANUAL table will be purged using the replenishment results purge days threshold date.

If this statement processes no rows, then the table has been fully purged and the function should return successfully. Otherwise, it should commit and return to the top of the loop. The loop with intermittent commits is necessary in order to ensure that rollback segments are not in danger of overflowing.

### **I/O Specification**

All files layouts input and output  
N/A.

### **Technical Issues**

Think about using Parallel Query in the delete.

If the client doesn't want to hold on to their results, it's probably better to remove this program from the schedule and modify rpl\_pre to truncate REPL\_RESULTS nightly (similar to the current treatment of ORD\_SUGG/F).

### **Testing Scenarios**

Insert records onto REPL\_RESULTS, BUYER\_WKSHT\_MANUAL, STORE\_ORDERS and IB\_RESULTS some with a repl\_date, create\_date or need\_date earlier than the threshold date, some later. Run rplprg. Those records earlier than the threshold should be deleted, while those later should remain.

Process enough records (or set the commit\_max\_ctr low enough) so that the program must perform multiple deletes in process ().

## Replenishment order [rplsplt]

### Design Overview

This new batch program will process all orders eligible for truck slitting during the nightly replenishment run. This program will run nightly after the replenishment-scaling program (supcnstr.pc) and before the replenishment purge program (rplapprv.pc). The purpose of this program will be to select all of the orders created by the replenishment programs which are eligible for splitting. The orders that are eligible will be sent into the truck splitting logic and the resulting orders will be created.

The orders which will be eligible for splitting are as follows:

- The order must have been created today by replenishment with  
ord\_inv\_mgmt.ord\_approve\_ind = 'Y'..
- The order must not have been already split.
- The order must be a single location order and the location must be a warehouse.
- The order must not have any allocations.

### Stored Procedures / Shared Modules (Maintainability)

#### ORDLIB.H

- Data structure and global variable definition
- Get\_inv\_mgmt()
- Add\_to\_order()
- Split\_po()
- Do\_inserts()
- Update\_group\_rounded\_qty()
- Kill\_order()

### Program Flow

Open driving cursor

While (records found)

Fetch driving cursor

If (first time)

Get ord\_inv\_mgmt info – get\_inv\_mgmt()

Set prev\_order = current\_order

If (prev\_order != current\_order)

CLEAN UP THE ORDER THAT HAS BEEN LOADED INTO THE LINKED LIST

Split the order (call split\_po())

If (the order was split)

Create the newly split orders – call do\_inserts()

– call update\_group\_rounded\_qty()

Give back the memory used by the ordering linked list – kill\_order()

Commit logic

INITIALIZE THE NEW ORDER

Get ord\_inv\_mgmt\_info – get\_inv\_mgmt()

Set prev\_order = current\_order

End (prev\_order != current\_order)

Add the current line item to the ordering linked list – add\_to\_order()

End loop

CLEAN UP LAST ORDER IN THE LINK LIST

Split the order (call split\_po())

If (the order was split)

    Create the newly split orders – call do\_inserts()

        – call update\_group\_rounded\_qty()

Give back the memory used by the ordering linked list – kill\_order()

### Function Level Description

#### Driving cursor:

```
EXEC SQL DECLARE c_driver CURSOR FOR
SELECT oh.order_no,
 'O', /* order as source type */
 oh.supplier,
 NVL(oh.dept,-1),
 oh.status,
 ol.item,
 DECODE(im.pack_ind,'Y', 'P', 'M'),
 ol.location,
 NVL(wh.physical_wh, ol.location),
 NVL(wh.repl_wh_link, ol.location),
 ol.loc_type,
 'Y', /*ot.due_ind,*/
 'N', /*t.crossdock_ind,*/
 '-1', /*NVL(ot.crossdock_store, '-1'),*/
 '-1', /*NVL(ot.contract_no, '-1'),*/
 '-1', /*NVL(ot.contract_type, '-1'),*/
 ol.qty_ordered,
 ol.last_rounded_qty,
 ol.last_grp_rounded_qty,
 os.origin_country_id,
 ol.unit_cost,
 NVL(ol.unit_cost_init, ol.unit_cost),
 1, /*supp_lead_time,*/
```

```

1, /*pickup_lead_time,*/
ol.non_scale_ind,
os.suppack_size,
1, /*eso*/
0, /*aso*/
NVL(ol.tsfpo_link_no, -1),
'-1', /* contract terms */
'-1' /* contract approval ind */
FROM ordhead oh,
ordsku os,
ordloc ol,
ord_inv_mgmt oim,
item_master im,
wh,
v_restart_supplier v
WHERE oh.written_date = TO_DATE(:ps_vdate, 'YYYYMMDD')
AND oh.contract_no IS NULL
AND oh.orig_ind = 0
AND oh.split_ref_ordno IS NULL
AND oh.order_no = oim.order_no
AND oim.ord_approve_ind = 'Y'
AND oim.truck_split_ind = 'Y'
AND oim.single_loc_ind = 'Y'
AND os.order_no = oh.order_no
AND ol.order_no = os.order_no
AND ol.item = os.item
AND im.item = ol.item
AND ol.location = wh.wh(+)
AND NOT EXISTS (select 'x'
 from alloc_header ah
 where ah.order_no = oh.order_no)
AND v.driver_value = oh.supplier
AND v.num_threads = TO_NUMBER(:ps_restart_num_threads)
AND v.thread_val = TO_NUMBER(:ps_restart_thread_val)
ORDER BY oh.order_no,
ol.item,
ol.location;
```

**Functions:***main()*

The standard Retek main function, this calls init(), process(), and posts to the daily log files.

*init()*

This function initializes the restart/recovery API and fetches global options and variables.

*Process()*

This is the main control function of the program. It contains the driving cursor. The driving cursor brings back each line item from legible orders. Each line item is sent to add\_line\_item() which places the line item into the ordering linked list. Once all the line items are added to the linked list, split\_po() is called. If split\_po() actually splits the order, do\_inserts() is called to create the new orders in the database. Kill\_order() is called after each call to split\_po() regardless



if the order is actually split or not, this function clears out the memory allocated for the ordering linked list.

### **Input Specifications**

#### **'Table-To-Table'**

Driving cursor select orders to be split using:

ORDHEAD, ORDSKU, ORDLOC, ORD\_INV\_MGMT, ITEM\_MASTER, and WH.

### **Output Specifications**

#### **'Table-To-Table'**

Created new orders see ordlib.h create\_po.pc design.

### **Scheduling Considerations**

Processing Cycle      Phase III

Scheduling Diagram:    NA

Pre-Processing:        supcnstr.pc

Post-Processing:       rplapprv.pc

Threading Scheme:     SUPPLIER

### **Restart/Recovery**

The program uses standard restart recovery. The logical unit of work is an order. This is a non-unique logical unit of work.

## RPM moving average [rpmmovavg]

### Design Overview

This new module will take the number of units sold as saved in the tran\_data table for all items designated for the store within a specified store/day, and maintain a smoothed average in the if\_rpm\_smoothed\_avg table. The average will only include days of information for when sales have a sales type of regular. If the item is on promotion or clearance, then no updating is required.

In order to populate the daily smoothed average table in RMS we will perform the following calculation. When sales are received from the POS upload we will receive the number of units sold at regular, promotion, and clearance price. Our intention is only to consider units sold at regular price for this calculation. In order to determine if we had no sales of the product for the day or if we just had no regular sales for the day we will assume the following in the program. If any promotion or clearance sales are received in POS upload, we will not update the smoothed average table. If no promotion or clearance sales are received and no regular sales are received we will update using zero as the unadjusted units amount. If no promotion or clearance sales are received and regular sales are received we will update using the received amount as the unadjusted units amount.

There will be a value at the department level that represents a threshold percent. This percent will be compared to the existing smoothed average value and used to limit the upper and lower boundaries for regular sales received. If the unadjusted units amount is outside of the boundaries, then the appropriate boundary amount will be substituted and become the adjusted units amount. If no threshold percent is defined for the department, it will be defaulted to 50%.

There will be a counter attached to each smoothed average value on the table. This counter will be increased by one each time the table value is updated until it reaches a maximum amount that is set at the department level. This value will be used to help determine the new average value. For example, if the user wants to always represent a 52 week (or table update instance) average, they would have the counter stop at 52. If they want the value in the table to represent 10 weeks of sales, the counter could be set to stop at 10. This will serve to create a type of smoothing because the weight of the existing value can be greater based on the number of instances recorded. If no counter is defined for the department, it will be defaulted to 52.

| TABLE               | INDEX | SELECT | INSERT | UPDATE | DELETE |
|---------------------|-------|--------|--------|--------|--------|
| item_loc            | No    | Yes    | No     | No     | No     |
| item_master         | No    | Yes    | No     | No     | No     |
| tran_data           | No    | Yes    | No     | No     | No     |
| deps                | No    | Yes    | No     | No     | No     |
| if_rpm_smoothed_avg | No    | Yes    | Yes    | Yes    | No     |

### Scheduling Constraints

This program has to be run after all of the data for the store/day has been uploaded into RMS and before the days data is removed from the tran\_data table

**Restart Recovery**

This program's logical unit of work (LUW) is store/day combination.

**Program Flow**

N/A

**Shared Modules**

N/A

**Driving Cursor:**

```
SELECT il.item,
 il.loc
 FROM item_loc il,
 item_master im
 WHERE il.item = im.item
 AND im.item_level = im.tran_level
 AND il.status in ('A','C')
 AND (il.loc = :ps_store
 OR :ps_store IS NULL)
 AND il.loc NOT IN (SELECT location
 FROM location_closed
 WHERE (location = :ps_store
 OR :ps_store IS NULL)
 AND close_date = :ps_business_date)
 ORDER BY il.loc, il.item;
```

**Function Level Description**

Main

**Init()**

- Call retek\_init() to get restart-recover variables.
- Limit the commit max counter so that it does not exceed the MAX\_ORACLE\_ARRAY\_SIZE.

**Process()**

- Allocate memory for arrays.
- Open the driving cursor. Fetch the data into the array within a while loop.
- Inside a for loop, which is inside the while loop, open and fetch another cursor to return item information. End of for loop
- Call calculate\_avg()
- Call insert\_update()
  - End of while loop.

**Calculate\_avg()**

- Set the sum of the units values. If values are beyond threshold limits, set them to equal the threshold limits.
- If the current counter is less than the max counter, increment counter by 1.
- If the item/store/day is being ran for the first time, ( no records exist on table if\_rpm\_smoothed\_avg), set the smoothed average value equal to the units sold. Otherwise perform calculation to get smoothed average.

**Insert\_update()**

- Within for loop, execute switch statement which will update the if\_rpm\_smoothed\_avg table for the corresponding store/item. If there is no record on the table for the item/store, insert record with the calculated values, and insert zero's for the rest of the values.

**final():**

restart/recovery close

**I/O Specification**

N/A

**Technical Issues**

N/A

## Purge RTV [rtvpgrg]

### Design Overview

The RTV Purge (rtvpgrg) module deletes out-dated RTV records from the system. RTV transactions that have been completed shipped or cancelled and those for which the completion date exceeds the RTV retention period are deleted. RTVs with a debit memo not in 'P'osted status (found on INVC\_HEAD) will not be purged. The RTV retention period is maintained on-line as a system option.

| TABLE        | INDEX | SELECT | INSERT | UPDATE | DELETE            |
|--------------|-------|--------|--------|--------|-------------------|
| PERIOD       | No    | Yes    | No     | No     | No                |
| RTV_HEAD     | Yes   | Yes    | No     | No     | Yes               |
| RTV_DETAIL   | Yes   | No     | No     | No     | Yes               |
| UNIT_OPTIONS | No    | Yes    | No     | No     | No                |
| INVC_HEAD    | No    | Yes    | No     | No     | Yes (via package) |

### Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

### Restart Recovery

```
EXEC SQL DECLARE c_rtv CURSOR FOR
 SELECT r.rtv_order_no
 FROM rtv_head r
 WHERE 0 < (SELECT NVL(MONTHS_BETWEEN(p.vdate, I
 r.completed_date), 0)
 - u.rtv_order_history_months
 FROM period p,
 unit_options u)
 AND NOT EXISTS (SELECT 'x'
 FROM invc_head i
 WHERE r.rtv_order_no = i.ref_rtv_order_no
 AND i.invc_type = 'D'
 AND i.status != 'P');
```

### Program Flow

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

# Sales audit escheated vouchers [saescheat]

## Functional Area

Escheatment of vouchers

## Module Affected

Saescheat.pc - new batch program that will set the status of vouchers that have met certain state's escheat rules, or have expired, to the proper status and produce a total for later export to Invoice Matching.

Saescheat\_nextesn.pc – new batch program that should include just a single function, nextEscheatSeqNo (), which selects a block of available sequence numbers. This will be very similar to saimptlog\_nexttsn.pc

## Design Overview

### Stored Procedures / Shared Modules (Maintainability)

Libretek.a library functions:

Retek\_init/retek\_close – to initialize and close Retek's restart/recovery module

FetchSysdate – to get the current date and time

FetchSaSystemOptions – to get ReSA system options

Libresa.a library functions:

Get\_lock – to establish a read lock on a store-day

Release\_lock – to release a store-day lock

## Input Specifications

### 'Table-To-Table'

Foundation tables used by this program for reference include:

Sa\_escheat\_options

Sa\_voucher\_options

Sa\_system\_options

Transaction data table used to drive the processing includes:

Sa\_voucher

## Output Specifications

### 'Table-To-Table'

Updates made to:

Sa\_voucher

Records inserted into:

Sa\_escheat\_voucher

Sa\_escheat\_total

**Function Level Description**

The logical unit of work in this program is a store, which will be used to drive the restart/recovery. Voucher entries will be selected based on the store that issued them. Totals will be produced using this store and the current vdate when the document was escheated.

Driving cursor:

```
SELECT store_day_seq_no,
 store
FROM sa_store_day,
 period
WHERE business_date = vdate
ORDER BY store
```

Init()

1. Call **retex\_init()** to initialize restart recovery.
2. Get system date/time from **fetchSysdate()**
3. Get the escheat\_ind from **fetchSaSystemOptions()**
4. Call **OraNumInit()** to initialize string numbers arithmetic operations.
5. Determine if escheatment is performed at the company level by calling **IsCompanyStateEscheat()**.
6. Call **FetchEscheatOptions()** and populate an "Escheat Options" array with all records returned.
7. Allocate memory for any dynamic structures by calling **SizeSDInfoArray()**, **SizeVoucherInfoArray()**, and **SizeUpdateSaVoucherArray()**.

Process()

1. Select a store-day to be processed by calling the first driving cursor.
2. Loop through this cursor
  - a. Attempt to lock the store-day by calling **get\_lock()**.
  - b. If escheat\_ind is Yes, call **ProcessEscheat()**
  - c. Call **DoUpdateSaVoucher()** to force update of the database.
  - d. Else call **ProcessIncomeAdjust()**
  - e. Call **WriteEscheatTotal ()**
3. After pl\_commit\_max\_ctr records have been fetched,
  - a. Call **DoUpdateSaVoucher()**
  - b. Call **DoInsertEscheatVoucher()**
  - c. Call **retex\_force\_commit()** to commit the database changes. This also releases the lock.

ProcessEscheat ()

```
Select tender_type_id,
 voucher sequence number,
 issued customer state,
 issued customer country,
 recipient state,
 recipient country,
 store state,
 store country
```

from the sa\_voucher, store, comphead and sa\_voucher\_options tables  
where the issued store equals the store in the current driving cursor record



and the status is issued (I)  
 and the expiration date is null  
 and the issued date is not null  
 and the (vdate – issued date) is greater than the escheat\_no\_days for that tender\_type\_id

1. Call the above cursor.
2. Loop through this cursor
  - a. If the issued state/country are not null, call **PerformEscheat()** passing these values and a 'state indicator' as parameters.
    1. If successful, break
  - b. If the issued customer state/country are not null, call **PerformEscheat()** passing these values and a 'customer indicator' as parameters.
    1. If successful, break
  - c. If the recipient state/country are not null, call **PerformEscheat()** passing these values and a 'recipient indicator' as parameters.
    1. If successful, break
  - d. If the company state/country are not null, call **PerformEscheat()** passing these values and a 'company indicator' as parameters.
    1. If successful, break
  - e. Call **AdjustVoucher ()**

**PerformEscheat ()**

1. Check the indicator passed to this function to determine which level to process at
  - a. Case 'state indicator'  
 Get value of the store\_escheat\_ind in the "Escheat Options" array for this state/country.
  - b. Case 'customer indicator'  
 Get value of the iss\_cust\_escheat\_ind in the "Escheat Options" array for this state/country.
  - c. Case 'recipient indicator'  
 Get value of the recipient\_escheat\_ind in the "Escheat Options" array for this state/country.
  - d. Case 'company indicator'  
 Get value of the hq\_escheat\_ind in the "Escheat Options" array for this state/country.
2. If value is Yes, call **EscheatVoucher()**, passing the store\_day\_seq\_no, voucher\_seq\_no, partner\_type, partner\_id and amount.
3. Else return 1

**EscheatVoucher (store\_day\_seq\_no, voucher\_seq\_no, partner\_type, partner\_id, amount)**

1. Call **UpdateSaVoucher()**, passing voucher\_seq\_no and status of E.
2. Call **InsertEscheatTotal()**, passing E as the escheat\_type, and the store\_day\_seq\_no, partner\_type, partner\_id and amount.
3. Call **WriteEscheatVoucher()**, passing the escheat\_seq\_no and voucher\_seq\_no.

**InsertEscheatTotal (store\_day\_seq\_no, partner\_type, partner\_id, amount, escheat\_type (status))**

1. Search "Escheat Total" array for store\_day\_seq\_no, partner
2. If found, add to escheat amount.

3. Else select the next available `escheat_seq_no` by calling **nextEscheatSeqNo()** and insert the value into this array with the `escheat_type` passed in as a parameter, by calling **InsertTotalNode()**.
4. Return `escheat_seq_no`.

**WriteEscheatVoucher** (`escheat_seq_no`, `voucher_seq_no`)

1. Insert `escheat_seq_no`, `voucher_seq_no` into `sa_escheat_voucher` array.

**DoInsertEscheatVoucher()**

1. This function does the actual insert to the `sa_escheat_voucher` table on the database.

**UpdateSaVoucher**(`voucher_seq_no`, `status`)

1. Insert `voucher_seq_no`, `status` and `vdate` into the `sa_update_voucher` array.

**DoUpdateSaVoucher()**

1. This function does the actual update of the `sa_voucher` table on the database. For each `voucher_seq_no` on the `sa_update_voucher` array, the `status` and the `escheat_date` get updated to the values in the array.

**WriteEscheatTotal ()**

1. Insert values from “Escheat Total” array into `sa_escheat_total`

**ProcessIncomeAdjust ()**

Select `tender_type_id`,  
       voucher sequence number,  
       issued customer state,  
       issued customer country,  
       recipient state,  
       recipient country,  
       store state,  
       store country

from the `sa_voucher`, `store`, `comphead` and `sa_voucher_options` tables  
 where the issued store equals the store in the current driving cursor record  
 and the status is issued (I)  
 and ((the expiration date is null  
 and the issued date is not null  
 and the (`vdate` – issued date) is greater than the `escheat_no_days` for that `tender_type_id`)  
 or the expiration date is less than the `vdate`)

1. Call the above cursor.
2. Loop through this cursor
  - a. Call **AdjustVoucher ()** for each voucher returned from the cursor

**AdjustVoucher ()**

1. Call **UpdateSaVoucher()**, passing the `voucher_seq_no` and `astatus` of I
2. Call **InsertEscheatTotal()**, passing `store_day_seq_no`, `partner_type`, `partner_id`, `amount`, `N` as the `escheat_type`
3. Call **WriteEscheatVoucher()**, passing the `escheat_seq_no` and `voucher_seq_no`.

**FetchEscheatOptions ()**

1. Select all columns from the `sa_escheat_options` table into the “Escheat Options” array.

`escheat_options_lookup (state, country)`

This function returns a pointer to the element of the `escheat_options` array that contains the state and country passed in.

`nodecmp ()`

This function compares two elements in the `escheat_options` array.

`InsertTotalNode ()`

This function inserts a new element in the `escheat_total` array, using insertion sort.

`escheat_total_lookup (store_day_seq_no, partner_type, partner_id)`

This function returns a pointer to the element of the `escheat_total` array that contains the information passed in.

`totalcmp ()`

This function compares two elements in the `escheat_total` array.

**Size\*Array and Free\*Array functions**

These functions allocate memory for global arrays, such as the `escheat_options` and `escheat_total` arrays and free the memory allocated to these arrays.

`IsCompanyStateEscheat()`

This function populates a `company_level_data` array that holds information about whether the state where the HQ of the company is incorporated, has escheatment laws.

`Final ()`

1. Clean up any memory used.
2. Call `retek_close` to close restart recovery.
3. Free memory for any dynamic structures by calling **`FreeSDInfoArray()`**, **`FreeVoucherInfoArray()`**, **`FreeUpdateSaVoucherArray()`**, **`FreeInsertEscheatVoucherArray()`**, **`FreeEscheatTotalArray()`**, and **`FreeInsertEscheatVoucherArray()`**.

### Scheduling Considerations

This program should be run as part of a monthly schedule. It should follow ReSA Totaling and Auditing process (`satotals.pc` and `sarules.pc`), and precede the export to Invoice Matching (`saexpim.pc?`) and the db purge (`sapurge.pc`).

### Locking Strategy

Locking will be performed via the `get_lock` and `release_lock` library functions to lock the store-day during processing.

### Restart/Recovery

A commit should be performed after processing is complete for each store-day.

## Sales audit ACH download [saexpach]

### Design Overview

This module will post Store/day deposit totals to the SA\_STORE\_ACH table and bank deposit totals for a given day to a standard ACH format file. The ACH export deviates from the typical Sales Audit export in that store/days must be exported even though errors may have occurred for a given day or store (depending on the unit of work defined) and also the store/day does not need to be closed for the export to occur. The nature of the ACH process is such that as much money as possible must be sent as soon as possible to the consolidating bank. Any adjustments to the amount sent can be made via the sabnkach form.

Also, we are assuming that there is only one total to be exported for ACH per store/day.

Deposits for store/days that have not been 'F'ully loaded will not be transferred to the consolidating bank. After they are fully loaded, their deposits will be picked up by the next run of the program.

Furthermore, the program estimates a 0 for a store/day that is closed, for example due to a holiday. An example is shown below (Wednesday is a holiday):

|                                              | Mon | Tues     | Wed | Thu       | Fri      |
|----------------------------------------------|-----|----------|-----|-----------|----------|
| Estimated deposit for next day               | 5   | 0        | /   | 10        |          |
| Adjustment to estimated deposit for this day | ... | 5        |     | 15        | 0        |
| Exported at close                            | ... | <b>5</b> |     | <b>25</b> | <b>0</b> |
| Actual deposit                               | ... | 10       |     | 15        | 10       |

In this example, we export only 5 (the adjustment) at close of Tuesday. The program is not run at close on Wednesday because it does not have a store\_day\_seq\_no. Thus, on Thursday, the estimate for that day is 0 and the adjustment equals the actual. Also, on Thursday, we estimate that the total is going to be 10 and we export 25 at close of Thursday. Thus, the bank account should return to the minimum balance at this point.

| Table                          | <i>Operations Performed</i> |        |        |        |
|--------------------------------|-----------------------------|--------|--------|--------|
|                                | Select                      | Insert | Update | Delete |
| <b>Period</b>                  | Yes                         | No     | No     | No     |
| <b>Sa_store_day</b>            | Yes                         | No     | No     | No     |
| <b>Sa_export_log</b>           | Yes                         | No     | Yes    | No     |
| <b>Sa_exported</b>             | No                          | Yes    | No     | No     |
| <b>Sa_store_ach</b>            | Yes                         | Yes    | Yes    | No     |
| <b>Sa_bank_ach</b>             | Yes                         | Yes    | Yes    | No     |
| <b>Sa_total</b>                | Yes                         | No     | No     | No     |
| <b>Sa_bank_store</b>           | Yes                         | No     | No     | No     |
| <b>Sa_store_day_write_lock</b> | Yes                         | No     | Yes    | No     |
| <b>Sa_store_day_read_lock</b>  | Yes                         | No     | No     | No     |
| <b>Store</b>                   | Yes                         | No     | No     | No     |
| <b>Partner</b>                 | Yes                         | No     | No     | No     |

### Background information – Quick Overview of the ACH process

ACH stands for Automated Clearing House and is a process by which funds can be transferred electronically from one account to another, possibly at a different financial institution. Instructions for each transaction are stored in a file, called an ACH file, which is then transferred across the ACH Network to be processed. This document provides only an overview of the process and will only describe points of interest for the saexpach program. It is beyond the scope of this document to provide the details of this process. Readers interested in knowing more about ACH should consult the 2000 ACH Rules published by the National ACH Association (NACHA).

There are 5 participants in an ACH transaction:

1. The originating company (called the Originator). The Originator is the entity requesting the transaction (i.e. this is where the transaction originates from).
2. The Originating Depository Financial Institution (ODFI).
3. The ACH Operator.
4. The Receiving Depository Financial Institution (RDFI).
5. The receiving company (called the Receiver).

\*It is important to note that the above description refers to direction of file transfers and not to direction of money flow.

Since the ReSA client has control over both the stores and the headquarters, the Originator can be either the former or the latter. To simplify the process, the headquarters will be the Originator, as this would require only one file to be produced, requesting money from each individual store.

Figure 1 gives a pictorial overview.

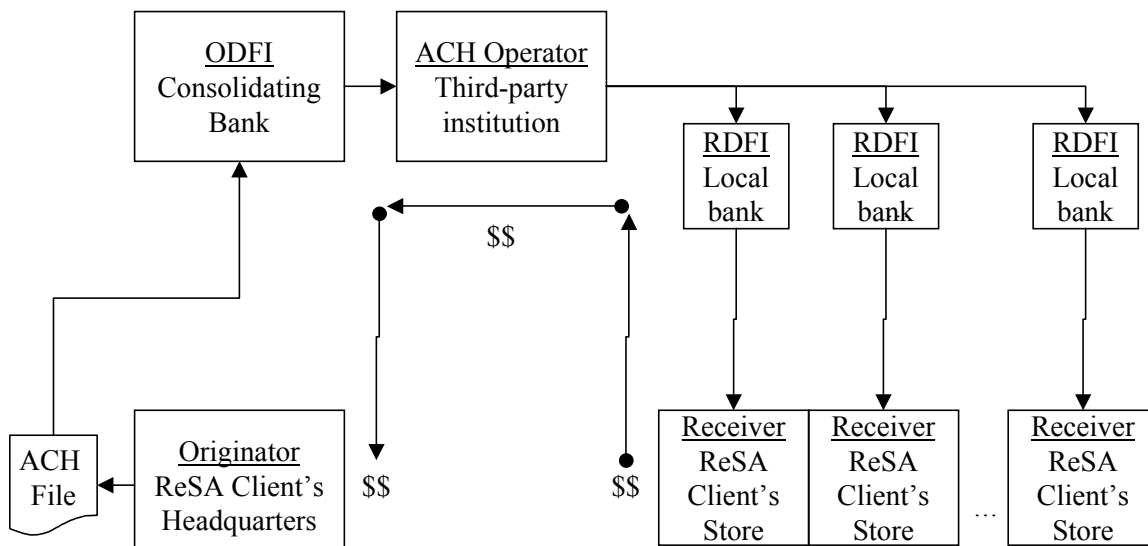


Figure 1: Overview of an ACH Network

The file that is produced at the Originator is sent to the ODFI which then routes it to the appropriate ACH operator(s). The latter will then contact the RDFI to request the money transfer.

In ACH jargon, the type of transaction that is being requested is a Cash Concentration and Disbursement (CCD). As of September 2000, however, transactions between institutions in different countries require a Corporate Cross-Border (CBR) Transaction. This program will meet this new requirement.

ACH is a US network of banks and therefore, this program should not be used for ACH look-alike networks outside the US, such as in Europe, as the file formats may be different. In other words, throughout this program, it is assumed that the country in which the consolidating bank is based is the United States.

Furthermore, all amounts in the ACH file are expected to be in US dollars (USD). Amounts for CBR transactions will have to be converted to USD.

Custom modifications can be made to this program such that output files that meet the requirements of other networks can be created. It is expected that the general structure of the program can be left unchanged and that only the functions that actually write the data out would have to change.

### Data Security

The fact that this program automates the transfer of funds on behalf of the user makes it a likely target for electronic theft. It must be made clear that the responsibility of electronic protection lies with the users themselves. Retek does not provide any kind of encryption or authentication beyond what is provided by the operating system and the database management system. Retek does provide some tips and recommendation to users:

1. A specific user should probably be used to run the program. This user would be the only one (or one of a few) who has access to this program.

2. The `umask` for this user should be setup so as to prevent other users to read/write its files. This would ensure that when the output file is created, it will not be accessible to other users.
3. The appropriate permissions should be setup on the directory which holds the ACH files. The most restrictive decision would be to not allow any other user to view the contents of the directory.
4. The password to this user should be kept confidential.
5. A secure means of communication should be implemented for transferring the file from where it has been created to the ACH network. This may be done via encryption, or by copying the file to a disk and trusting the courier to deliver the files intact.
6. Retek assumes that the ACH network is secure.

### **Scheduling Constraints**

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA Totaling process: `satotals` and `sarules`.

Threading Scheme: N/A

### **Restart Recovery**

Logical Unit of Work (recommended Commit checkpoints )

Driving Cursor

This module is in two distinct parts, with two different logical units of work. Thus restart/recovery has to be implemented so that the first part does not get reprocessed in case the program is being restarted. Details on the implementation follow.

The first driving cursor in this module retrieves a store/day to generate ACH totals. Once the first cursor is complete, the second retrieves bank locations by account numbers.

The first Logical Unit of Work (LUW) is defined as a unique store/day combination. Records will be fetched, using the first driving cursor, in batches of `commit_max_ctr`, but processed one store/day at a time.

The first driving cursor will fetch all store/days that have been 'F'ully Loaded, whose audit status is 'A'udited, 'H'Q Errors Pending or 'S'tore Errors Pending and that are ready to be exported to ACH. Before processing starts, a write lock is obtained using `get_lock ()`. This driving cursor only fetches store/days with a `sa_export_log.status` of `SAES_R`. After a store/day is processed, `sa_export_log.status` is set to `SAES_P` so that this store/day will not be selected again if the program is restarted. We commit using `retek_force_commit` after each store/day has been processed and `sa_export_log` updated, so as to release the lock.

In case a store/day could not be processed due to locking, then the store/day information is placed on a list (called locked store/day list) and the next store/day is processed. This list is kept in memory and is available only during processing. If the store for a store/day obtained from the first driving cursor, is on the locked store/day list, then this store/day cannot be processed. This is the case because there is a data dependency such that data from a particular store/day is

dependent on data for the same store but at an earlier date. Thus, if a store/day cannot be processed, then subsequent store/days for the same store cannot be processed either. After the driving cursor returns no more data, the program attempts to process each store/day on the list two more times. If the store/day is still locked, then it is skipped entirely and a message is printed to the error log.

The second LUW is a bank account number. Again, records will be fetched in batches of `commit_max_ctr`. The second driving cursor cannot retrieve information by the LUW because it is possible for the store's currency to be different from the local bank's currency. In that case, a currency conversion is needed.

For each store/day, the query should retrieve the required ACH transfer. The latter is determined by adding the estimated deposit for the next day, the adjustment to the estimate for the current day and any manual adjustment to the estimate.

Since a store can be associated with different accounts at different banks, only accounts that are consolidated should be retrieved. Since it is possible for the local bank to be in a different country than the consolidating bank, the currency of the partner should also be fetched.

Since processing is dependent on the type of account at the RDFI, the account type should be fetched by this cursor.

Due to differences in transaction processing in cases when the bank is outside the US, the partner's country should also be fetched. The results of the query should be sorted by partner country.

The results of the query should also be ordered by accounts.

## Program Flow

### *Structure Chart*

Please see the following document for the complete structure chart of the standard export for ReSA.

Functional Design – SA export.doc

## Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

rettek library functions:

- `rettek_init()` – This function initializes restart/recovery.
- `rettek_close()` – This function cleans up restart/recovery.
- `rettek_force_commit()` – This function commits any change to the database.

Sales/Audit library functions (libresa):

- `fetchVdate()` – This function is used to get the vdate.
- `fetchSysdate()` – This function is used to get system date and time
- `fetchStoreDayToBeExported()` – This function contains the first driving cursor.
- `get_lock()` – This function is used to lock the store/day being processed.
- `OraNumInit()` – Initialize OraNum functions.
- `OraNumAdd()` – Add two large numbers passed in as strings.



- OraNumSub() –Subtract two large numbers passed in as strings.
- OraNumDiv() –Divide two large numbers passed in as strings.

### Function Level Description

All database interactions required and error handling considerations

#### Init ()

- Initialize restart/recovery by calling `restart_init()`.
- Get the vdate from the period table and the system time.
- Get the system level information: the sender id, the company id, the consolidating bank name, the consolidating routing number and the consolidating account number. These are on the `sa_ach_info` table.

#### Process ()

1. Get the next store/day to be processed (exported) by fetching from the first driving cursor.
2. Attempt to lock the store/day with a call to **get\_lock()**. If this fails, write the store to a linked list (which contains all unprocessed store/days).
3. Skip to step 7 if the store of the store/day to be processed is for a store which is on the linked list.
4. Call the function **postStoreACH()** for the current store/day.
5. Set `sa_export_log.status` to `SAES_P` by calling **setProcessed()** for the current store/day, so that it will not be processed again in case of a restart.
6. Call **rettek\_force\_commit()** to commit changes to the database and to release write lock.
7. Loop from beginning until the driving cursor returns no more data.
8. Call the function **postBankSummaryTotals()**.

#### Final ()

- Clean up restart/recovery by calling **rettek\_close()**.
- If the program has successfully processed the data, call **rettek\_refresh\_thread()**.

#### PostStoreACH ()

- This function will generate and post an estimate and adjustment to the `SA_STORE_ACH` table for a given store/day. The function `postStoreACH` will accomplish the following processes in the following order:
- Get the following pieces of data for the system code `SYSE_ACH`:
- The total for the current business date,
- Get the total for the following business date if it exists (by calling `GetTomorrowTotal` ),
- Call the function `GetPastData()` to get the totals for the past 4 weeks and for yesterday (that is, if the current store/day is for a Tuesday, then we want to get the totals for the past 4 Wednesdays and for yesterday). The latter pieces of data are obtained from the `sa_store_ach` table, by summing the estimate for a day with the adjustment for the same day.
- Call the function `GetPartnerInfo()` to get partner type and partner id information.
- If there are more than one total for `SYSE_ACH` for a particular day, then this should be noted in the error log. We expect only one total per store/day. Only the first total returned by the function will be used, the rest will be ignored.
- Call the function `CalculateData()` to compute the estimate for the next business day and adjustment for the current store/day.
- Call the function `PostStoreACHTable()`.

**GetTomorrowTotal ()**

- This function attempts to get the total for the next business day to be used as the estimate. It returns a -1 if a fatal error occurred, a 0 if it was able to get the total. If a total was not found, the estimate is assigned to -1. If a store/day is never opened (i.e. a holiday), then a 0 is estimated for that store/day. Also, if a total is found, it should not be marked as exported.

**GetPastData ()**

- This function retrieves totals for the same day of the week over the past 4 weeks and for the previous business day.

**GetPartnerInfo ()**

- This function retrieves the bank partner (partner type and partner id) for the given store whose account is consolidated.

**CalculateData ()**

- This function calculates the estimate for the next business day and adjustment for the current store/day.
- Find the estimate for the following business date using the following rules:
- If the total for the following business date exists, then this is the estimate.
- Otherwise, the estimate is the average for the data for the past 4 weeks. If we obtain data for fewer than 4 weeks, then we use the available data, but if we do not obtain any data, then we use the current day's total as the estimate.
- If the estimate is a 0, then we use the current day's total as the estimate.
- Calculate the adjustment, which is the current date's total minus the estimate for the current date (which lies on the row for the previous day on the sa\_store\_ach table) and minus the manual adjustment for the current date (which lies on the row for the previous day on the sa\_store\_ach table).

**ProcessLockedSD ()**

This function processes any store/days that were not in the **process()** function due to locking. The list of such store/days is stored on the linked list.

1. Try to process the store/days that were not processed, that is, those that are on the linked list. Thus, for each store/day on the linked list, we try to obtain a lock. If one is not obtained, then we skip this store/day. If a lock is obtained, then we remove the store/day from the list.
2. Skip to step 5 if the store of the store/day to be processed is for a store which is on the linked list.
3. Call the function **postStoreACH** for the current store/day.
4. Set sa\_export\_log.status to SAES\_P by calling **setProcessed()** for the current store/day, so that it will not be processed again in case of a restart.
5. Loop through steps 1 to 3, until each store/day in the list has been looked at.
6. Loop through steps 1 to 5 NUM\_LOCK\_RETRIES times. NUM\_LOCK\_RETRIES is by default 2. Thus, we try to attempt to process store/days that are locked two more times before giving up and skipping all locked store/days entirely.
7. For each store/day that was not processed, we write an error to the log.

**PostStoreACHTable ()**

This function inserts data into the sa\_store\_ach table. It updates if there is already an entry for the store, business date and partner.

- If there is no entry in the sa\_store\_ach table for the current store/day.
- Create an entry in the SA\_STORE\_ACH table with the current store\_day\_seq\_no and the new estimate and adjustment deposits for the current store\_day\_seq\_no.

- If there is an entry in the sa\_store\_ach table for the current store/day.
- Update the entry in sa\_store\_ach with the estimated deposit, and estimated deposit adjustment.

### postBankSummaryTotals ()

This procedure will summarize the bank transaction totals to the ACH output file. Please see the section on I/O specifications for more information about the format of this file.

1. Open and fetch from the second driving cursor.
2. If any entries are to be made (i.e. there are results from the cursor), create ACH file and write file header by calling **WriteACHFileHeader()**.
3. If the country of the bank just retrieved is different from the country of the previous bank, write a Batch Control Record by calling **WriteACHBatchControl()**, unless no Batch Header records have been written yet.
4. If the country of the bank just retrieved is different from the country of the previous bank, a new Batch Header record needs to be written. If the bank's country is the US, the **WriteACHCCDBatchHeader()** function should be called to write a Batch Header for CCD transactions. For all other countries, the **WriteACHCBRBatchHeader()** function should be called to write a Batch Header for CBR transactions.
5. If the store's currency is different from the bank's currency, do a conversion. Sum all the deposits for each bank account.
6. For each account at a bank in the US, create a CCD record in the file by calling **WriteACHCCDEntry()**.
7. For each account at a bank outside the US, create a CBR record by calling **WriteACHCBREntry()**.
  - If the amount to be transferred is negative, the record should be skipped.
  - If the account is a checking account, the transaction code to use is '27'.
  - If the account is a savings account, the transaction code to use is '37'.
8. If the amount to be transferred is positive, call the function **PostBankACHTable()** to record the amount of the ACH entry, else do nothing.
9. Keep running totals for the current batch's total amount and the total ACH amounts.
10. Commit after pl\_commit\_max\_ctr LUW have been processed. Redefine the SAVEPOINT after the commit because savepoints are lost after a commit.
11. Loop to step 3 until the cursor returns no data.
12. Write the ACH Batch Control record and the ACH File Control record
13. The ACH file format requires that the file size meet certain "block" requirements. See the section on the ACH file format for more details. Write the required number of "completion records" to meet the blocking requirements.
14. Mark all store/days *that were not locked* (i.e. those with a sa\_export\_log.status of SAES\_P) as completed (SAES\_E) in the sa\_export\_log.

### postBankACHTable ()

This function inserts into the table sa\_bank\_ach. It updates if there already exist a record for the same partner and business date.

1. If an entry does not exist for the current bank and date in the sa\_bank\_ach table:
  - Make an entry in the sa\_bank\_ach table for the current bank and account placing the sums of the store ACH amounts and adjustments in the ACH amount field (sa\_bank\_ach.ach\_amt).
2. If an entry exists for the current bank and date in the sa\_bank\_ach table:
  - Add the manual adjustment to the bank ACH deposit amount.

- Update the sa\_bank\_ach table with the bank ACH deposit amount (sa\_bank\_ach.ach\_amt).

#### File Output functions

The functions **WriteACHFileHeader()**, **WriteACHFileControl()**, **WriteACHCCDBatchHeader()**, **WriteACHCBRBatchHeader()**, **WriteACHBatchControl()**, **WriteACHCCDEntry()**, **WriteACHCBREntry()**, **WriteACHCBRAddendum()** and **WriteACHCompleteBlock()** write the File Header Record, the File Control Record, the Batch Header Record for CCD transactions, the Batch Header Record for CBR transactions, the Batch Control Record, the CCD Entry Record, the CBR Entry Record, the CBR Addendum Record and the Completion Blocks, respectively. The **WriteACHCBREntry()** function should call the **WriteACHCBRAddendum()** function after writing to the file.

#### Linked list functions

The functions **AddToList()**, **DeleteList()**, **GetNext()** and **RemoveFromList()** provide means to manipulate and to retrieve data from the linked list which contains the store/days which were not processed due to locking issues.

#### MarkAllStoreDaysCompleted ()

This function sets the sa\_export\_log.status to SAES\_E for store/days whose status is SAES\_P. These are the store/days that have been exported. If a store/day was not exported, it will be picked up in the next run after it has met the conditions for export.

#### SetCurrencyDecimals ()

Given a currency code and an amount with 4 implicit decimals, this function will give out an amount with the appropriate number of decimals for the currency. For more details, see the BAI file format documentation. For example, there are two implicit decimals for the US Dollar, but none for the Japanese Yen. This function may need to be expanded because only a select few currencies are being processed. The last two decimal places are dropped for currencies that are not explicitly defined.

#### TruncateDec ()

This function truncates a number at the decimal point, i.e. “1234.56” becomes “1234”.

#### I/O Specification

All files layouts input and output

#### ACH File Structure

This section describes the structure of the output file of the saexpach.pc program. The output file conforms to the requirements imposed by the National Automated Clearing House Association (NACHA) and only the subset of records used by this program is outlined here. For more information on the other types of records and more information about the rules and regulations governing the ACH network, please refer to the “2000 ACH Rules” book published by NACHA.

The ACH file format is similar in many ways to Retek’s flat file formats. The most distinctive differences are:

- The record type is a one-digit number rather than a five-digit character field.
- All records are 94 characters in length.
- Records are organized in blocks, where 1 block = 940 characters = 10 records.

- The File Control Record (similar to an FTAIL) contains a “Block Count” field which gives the total number of blocks in the file, including the File Header Record and the File Trailer Record. Records containing 9’s must be used to complete the last block. For example, a file with 15 records will need 5 such records to give it a Block Count of 2. These “completion records” go at the end of the file.
- Transactions are organized in batches. Similar transactions make up one batch. In ReSA’s case, the transactions are organized by the country of origin of the funds.

### File Header Record

This record contains information about the characteristics of the file, such as sender and receiver, creation datetime, and so on.

| Field Name                 | Field Description                                                                                                                                                                                                                                           | Value                                                  | Length | Jstf/<br>Pad* |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|--------|---------------|
| Record Type Code           | The type of record.                                                                                                                                                                                                                                         | ‘1’                                                    | 1      | None          |
| Priority Code              | Reserved for future scheme for priority handling of files. ‘01’ should be used.                                                                                                                                                                             | ‘01’                                                   | 2      | None          |
| Immediate Destination      | Routing number of the consolidating bank. The field begins with a blank, followed by the 4-digit Federal Reserve Routing Symbol, the 4-digit ABA Institution Identifier, and the Check Digit.                                                               | SA_BANK_ST<br>ORE.<br>CONSOLIDATI<br>NG_ROUTING<br>_NO | 10     | None          |
| Immediate Origin           | A unique identification to determine the Originator. The ID and the format are supplied by the consolidating bank. Note that the user is responsible for the padding. That is, it is assumed that the data in the field will be exactly 10 characters wide. | SA_SYSTEM_<br>OPTIONS.<br>ACH_SENDER<br>_ID            | 10     | None          |
| File Creation Date         | Date when the file was created.                                                                                                                                                                                                                             | YYMMDD                                                 | 6      | None          |
| File Creation Time         | Time when the file was created.                                                                                                                                                                                                                             | HH24MM                                                 | 4      | None          |
| File ID Modifier           | This is used to differentiate files created on the same date and between the same Origin/Destination. Valid values are A through Z and 0 through 9. It is expected that only one file will be created per day, so a ‘0’ should be used.                     | ‘0’                                                    | 1      | None          |
| Record Size                | Number of characters per record.                                                                                                                                                                                                                            | ‘094’                                                  | 3      | None          |
| Blocking Factor            | Number of physical records within a block.                                                                                                                                                                                                                  | ‘10’                                                   | 2      | None          |
| Format Code                | Reserved for future format variations. A ‘1’ should be used.                                                                                                                                                                                                | ‘1’                                                    | 1      | None          |
| Immediate Destination Name | The name of the consolidating bank.                                                                                                                                                                                                                         | SA_SYSTEM_<br>OPTIONS.<br>CONSOL_BAN<br>K_NAME         | 23     | L/B           |
| Immediate Origin Name      | The name of the company.                                                                                                                                                                                                                                    | COMPHEAD.<br>CO_NAME                                   | 23     | L/B           |
| Reference Code             | Any reference code. This is an optional field. ReSA will not populate this field as the create datetime should be enough to reference the                                                                                                                   | blanks                                                 | 8      | None          |

|  |                                                                           |  |  |  |
|--|---------------------------------------------------------------------------|--|--|--|
|  | data that was exported by comparing with<br>SA_EXPORTED.<br>EXP_DATETIME. |  |  |  |
|--|---------------------------------------------------------------------------|--|--|--|

\* Note: This column described the justification and padding involved in the field being described. 'L' stands for left; 'R' stands for Right; 'B' stands for blank padding and '0' stands for 0 padding. None means that the field should be completely filled.

#### Batch Header Record for CCD transactions

| Field Name                 | Field Description                                                                                                                                                                                                                                                                                       | Value                                          | Length | Jstf/<br>Pad* |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|--------|---------------|
| Record Type Code           | The type of record.                                                                                                                                                                                                                                                                                     | '5'                                            | 1      | None          |
| Service Class Code         | This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used.                                                                                   | '225'                                          | 3      | None          |
| Company Name               | The name of the company.                                                                                                                                                                                                                                                                                | First 16 characters of<br>COMPHEAD.<br>CO_NAME | 16     | L/B           |
| Company Discretionary Data | Any kind of data specific to the company. ReSA will not use this field                                                                                                                                                                                                                                  | blanks                                         | 20     | None          |
| Company Identification     | An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example, "1" IRS Employer ID Number "9" User Assigned Number. ReSA assumes that the company_id field on the sa_system_options table will contain the correct id. | SA_SYSTEM_<br>OPTIONS.<br>COMPANY_ID           | 10     | L/B           |
| Standard Entry Class Code  | This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CCD entries, this field should hold the value 'CCD'.                                                                                                                                                | 'CCD'                                          | 3      | None          |
| Company Entry Description  | A short description from the Originator about the purpose of the entry.                                                                                                                                                                                                                                 | 'CONSOL.'                                      | 10     | L/B           |
| Company Descriptive Date   | Optional field providing a date to the Receiver for descriptive purposes. ReSA will populate it with the next day's date in the YYMMDD format.                                                                                                                                                          | YYMMDD<br>format of<br>PERIOD.VDATE + 1        | 6      | None          |
| Effective Entry Date       | The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date).                                       | YYMMDD<br>format of<br>PERIOD.VDATE + 1        | 6      | None          |
| Settlement Date            | This is inserted by receiving ACH Operator. ReSA will leave this blank.                                                                                                                                                                                                                                 | blanks                                         | 3      | None          |

|                        |                                                                                                                                                   |                                                           |   |      |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|---|------|
| Originator Status Code | This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution. | '1'                                                       | 1 | None |
| ODFI Identification    | 8-digit routing number of the ODFI.                                                                                                               | First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO | 8 | None |
| Batch Number           | The batch number.                                                                                                                                 |                                                           | 7 | R/0  |

### Batch Header Record for CBR transactions

| Field Name                           | Field Description                                                                                                                                                                                                               | Value                                   | Length | Jstf/Pad* |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|--------|-----------|
| Record Type Code                     | The type of record.                                                                                                                                                                                                             | '5'                                     | 1      | None      |
| Service Class Code                   | This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used.           | '225'                                   | 3      | None      |
| Company Name                         | The name of the company.                                                                                                                                                                                                        | First 16 characters of COMPHEAD.CO_NAME | 16     | L/B       |
| Foreign Exchange Indicator           | Code used to indicate the foreign exchange conversion methodology applied to a CBR entry. Retek uses the "Fixed-to-Variable" method to convert from the foreign currency into US dollars. Therefore, this field should be 'FV'. | 'FV'                                    | 2      | None      |
| Foreign Exchange Reference Indicator | Code used to indicate the contents of the Foreign Exchange Reference field. The latter will contain the conversion rate used by Retek which means that the value should be '1'.                                                 | '1'                                     | 1      | None      |
| Foreign Exchange Reference           | This should contain the foreign exchange rate used to compute the amounts in the CBR Entry Record. No decimal places are implied, that is, this field should contain the exact rate used.                                       |                                         | 15     | L/B       |
| ISO Destination Country Code         | The country where the money is to be transferred to. Since ReSA assumes that the consolidating bank will be in the US, this should be 'US' – NOTE: verify that "US" is the correct ISO code for United States of America.       | 'US'                                    | 2      | None      |
| Company Identification               | An alphanumeric code identifying the company. The first character may be the                                                                                                                                                    | SA_SYSTEM_OPTIONS.                      | 10     | L/B       |



|                               |                                                                                                                                                                                                                                                                   |                                                                 |    |      |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|----|------|
|                               | ANSI one-digit Identification Code Designators (ICD). For example, “1” IRS Employer ID Number “9” User Assigned Number.<br>ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.                                     | COMPANY_ID                                                      |    |      |
| Standard Entry Class Code     | This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CBR entries, this field should hold the value ‘CBR’.                                                                                                          | ‘CBR’                                                           | 3  | None |
| Company Entry Description     | A short description from the Originator about the purpose of the entry.                                                                                                                                                                                           | ‘CONSOL.’                                                       | 10 | L/B  |
| ISO Originating Currency Code | Currency code in which the funds are originating from. This must be the ISO code of the currency.                                                                                                                                                                 | PARTNER.<br>CURRENCY_CODE                                       | 3  | None |
| ISO Destination Currency Code | Currency code in which the funds are to be received. This must be “USD”.                                                                                                                                                                                          | ‘USD’                                                           | 3  | None |
| Effective Entry Date          | The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date). | YYMMDD<br>format of<br>PERIOD.VDATE + 1                         | 6  | None |
| Settlement Date               | This is inserted by receiving ACH Operator. ReSA will leave this blank.                                                                                                                                                                                           | blanks                                                          | 3  | None |
| Originator Status Code        | This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution.                                                                                                                 | ‘1’                                                             | 1  | None |
| ODFI Identification           | 8-digit routing number of the ODFI.                                                                                                                                                                                                                               | First 8 digits of<br>SA_BANK_STORE.<br>CONSOLIDATING_ROUTING_NO | 8  | None |
| Batch Number                  | The batch number. It is not expected that the file will have more than two batches.                                                                                                                                                                               | ‘1’ or ‘2’                                                      | 7  | R/0  |

**CCD Entry Detail Record**

| Field Name       | Field Description                                                                                                                                                                                                                   | Value        | Length | Jstf/<br>Pad* |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|--------|---------------|
| Record Type Code | The type of record.                                                                                                                                                                                                                 | ‘6’          | 1      | None          |
| Transaction Code | Code used to identify the type of debit and credit. This is dependent on the type of account and on the direction of funds transfer.<br>‘27’ – if the account is a checking account,<br>‘37’ – if the account is a savings account. | ‘27’ or ‘37’ | 2      | None          |



|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                   |    |      |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|----|------|
| RDFI Identification      | 8-digit routing number of the RDFI.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | First 8 digits of SA_BANK_STORE.ROUTING_NO        | 8  | None |
| Check Digit              | This is the 9 <sup>th</sup> digit from the routing number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 9 <sup>th</sup> digit of SA_BANK_STORE.ROUTING_NO | 1  | None |
| DFI Account Number       | The account number at the local bank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | SA_BANK_STORE.BANK_ACCT_NO                        | 17 | L/B  |
| Amount                   | The amount involved in the transaction. This field is numeric only and the last two digits are automatically assumed to be decimals. ReSA amounts are stored as 20 digit numbers, with 4 for decimals. ReSA will truncate the last two digits of the amount and should the resulting amount be greater than 10 digits, this program will abort with an error. It is not expected that a client will send an ACH amount greater than US\$100 million. The values for this are taken from the sa_store_ach table. The values from the columns today_adj_deposit_est, next_day_man_adj_deposit, and next_day_deposit_est are added up by business_date and then multiplied by 10000 and later divided by 100 to obtain a dollar amount. |                                                   | 10 | R/0  |
| Identification Number    | Optional field containing a number used by Originator to insert its own number for tracing purposes. ReSA will not populate this field.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | blanks                                            | 15 | None |
| Receiving Company Name   | Name of the local store.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | STORE. STORE_NAME                                 | 22 | L/B  |
| Discretionary Data       | Any kind of data specific to the transaction. ReSA will not use this field                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | blanks                                            | 2  | None |
| Addenda Record Indicator | This field identifies whether this entry record contains addenda records. ReSA has no use for such records in CCD and will use the value of '0'                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | '0'                                               | 1  | None |
| Trace Number             | Used to uniquely identify each entry within a batch. The first 8 digits contain the routing number of the ODFI and the other 7 contains a sequence number. This sequence number should be ascending. Although the ACH specification does not require the numbers to be consecutive, ReSA will use consecutive numbers. Trace numbers should not be duplicated between batches.                                                                                                                                                                                                                                                                                                                                                       |                                                   | 15 | None |

## CBR Entry Detail Record

| Field Name               | Field Description                                                                                                                                                                                                                                                                                                                                                              | Value                                                        | Length | Jstf/<br>Pad* |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|--------|---------------|
| Record Type Code         | The type of record.                                                                                                                                                                                                                                                                                                                                                            | '6'                                                          | 1      | None          |
| Transaction Code         | Code used to identify the type of debit and credit. This is dependent on the type of account and on the direction of funds transfer.<br>'27' – if the account is a checking account,<br>'37' – if the account is a savings account.                                                                                                                                            | '27' or '37'                                                 | 2      | None          |
| RDFI Identification      | 8-digit routing number of the RDFI.                                                                                                                                                                                                                                                                                                                                            | First 8 digits of<br>SA_BANK_ST<br>ORE.<br>ROUTING_NO        | 8      | None          |
| Check Digit              | This is the 9 <sup>th</sup> digit from the routing number.                                                                                                                                                                                                                                                                                                                     | 9 <sup>th</sup> digit of<br>SA_BANK_ST<br>ORE.<br>ROUTING_NO | 1      | None          |
| DFI Account Number       | The account number at the local bank.                                                                                                                                                                                                                                                                                                                                          | SA_BANK_ST<br>ORE.<br>BANK_ACCT_<br>NO                       | 17     | L/B           |
| Amount                   | The amount involved in the transaction. This field is numeric only and the last two digits are automatically assumed to be decimals. This amount is in US dollars.                                                                                                                                                                                                             |                                                              | 10     | R/0           |
| Identification Number    | Optional field containing a number used by Originator to insert its own number for tracing purposes. ReSA will not populate this field.                                                                                                                                                                                                                                        | blanks                                                       | 15     | None          |
| Receiving Company Name   | Name of the local store.                                                                                                                                                                                                                                                                                                                                                       | STORE.<br>STORE_NAME                                         | 22     | L/B           |
| Discretionary Data       | Any kind of data specific to the transaction. ReSA will not use this field                                                                                                                                                                                                                                                                                                     | blanks                                                       | 2      | None          |
| Addenda Record Indicator | This field identifies whether this entry record contains addenda records. Since CBR records must be followed by an addendum record, this value should be '1'.                                                                                                                                                                                                                  | '1'                                                          | 1      | None          |
| Trace Number             | Used to uniquely identify each entry within a batch. The first 8 digits contain the routing number of the ODFI and the other 7 contains a sequence number. This sequence number should be ascending. Although the ACH specification does not require the numbers to be consecutive, ReSA will use consecutive numbers. Trace numbers should not be duplicated between batches. |                                                              | 15     | None          |

**CBR Addendum Record**

| Field Name                   | Field Description                                                                                                                                                                                                                                                 | Value | Length | Jstf/<br>Pad* |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------|---------------|
| Record Type Code             | The type of record.                                                                                                                                                                                                                                               | '7'   | 1      | None          |
| Addenda Type Code            | This code identifies the type of addendum record. CBR has only one type of Addenda Type Code: '01'.                                                                                                                                                               | '01'  | 2      | None          |
| Payment Related Information  |                                                                                                                                                                                                                                                                   |       | 80     | L/B           |
| Addenda Sequence Number      | This is a sequence number denoting the position of each addendum record. The first record should always have a sequence number of 1 and subsequent records must be increasing and consecutive. ReSA will create only one addendum record for the CBR transaction. | '1'   | 4      | R/0           |
| Entry Detail Sequence Number | This is the sequence number part of the Trace Number of the entry record to which this addendum is referring.                                                                                                                                                     |       | 7      | R/0           |

**Batch Control Record**

| Field Name                                | Field Description                                                                                                                                                  | Value                            | Length | Jstf/<br>Pad* |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|--------|---------------|
| Record Type Code                          | The type of record                                                                                                                                                 | '8'                              | 1      | None          |
| Service Class Code                        | This field identifies the general classification of dollar entries to be exchanged. Since money is being requested, this code should be 225 for "ACH Debits only". | '225'                            | 3      | None          |
| Entry/Addenda Count                       | The number of entries and addenda in the batch. Basically, this is the number of records between the Batch Header Record and the Batch Control Record.             |                                  | 6      | R/0           |
| Entry Hash                                | This is the sum of the RDFI IDs in the detail records. It is the arithmetic sum of the 8-digit routing number. Overflow on the high order bits is ignored.         |                                  | 10     | R/0           |
| Total Debit Entry Dollar Amount in batch  | These fields contain the accumulated debit and credit for the batch. This field is numeric only and the last two digits are automatically assumed to be decimals.  |                                  | 12     | R/0           |
| Total Credit Entry Dollar Amount in batch |                                                                                                                                                                    |                                  | 12     | R/0           |
| Company Identification                    | An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example,                    | SA_SYSTEM_OPTIONS.<br>COMPANY_ID | 10     | L/B           |

|                                   |                                                                                                                                                                           |                                                           |    |      |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|----|------|
|                                   | “1” IRS Employer ID Number<br>“9” User Assigned Number.<br>ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.             |                                                           |    |      |
| Message Authentication Code (MAC) | The first 8 characters represent a code from the DES (Data Encryption Standard) algorithm. The remaining eleven characters are blanks. ReSA will not populate this field. | blanks                                                    | 19 | None |
| Reserved                          | Reserved                                                                                                                                                                  | blanks                                                    | 6  | None |
| ODFI Identification               | 8-digit routing number of the ODFI.                                                                                                                                       | First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO | 8  | None |
| Batch Number                      | The batch number.                                                                                                                                                         |                                                           | 7  | R/0  |

### File Control Record

This record contains summary information about the file to verify its integrity.

| Field Name                               | Field Description                                                                                                                                                                         | Value                                         | Length | Jstf/Pad* |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|--------|-----------|
| Record Type Code                         | The type of record.                                                                                                                                                                       | ‘9’                                           | 1      | None      |
| Batch Count                              | The number of batches sent in the file.                                                                                                                                                   |                                               | 6      | R/0       |
| Block Count                              | The number of physical blocks in the file, including both File Header and File Control Records. This is the ceiling of the number of records divided by the blocking factor, which is 10. | $\lceil (\text{Number of records})/10 \rceil$ | 6      | R/0       |
| Entry/Addenda Count                      | The number of entries and addenda in the file. Basically, this is the number of records between the Batch Header Record and the Batch Control Record.                                     |                                               | 8      | R/0       |
| Entry Hash                               | This is the sum of the Entry Hash fields on the Batch Control Records.                                                                                                                    |                                               | 10     | R/0       |
| Total Debit Entry Dollar Amount in File  | These fields contain the accumulated debit and credit for the file. This field is numeric only and the last two digits are automatically assumed to be decimals.                          |                                               | 12     | R/0       |
| Total Credit Entry Dollar Amount in File |                                                                                                                                                                                           |                                               | 12     | R/0       |
| Reserved                                 | This field should be filled with blanks. It is used to ensure that each record is of length 94.                                                                                           | blank                                         | 39     | None      |

## Sales audit export to ReIM [saexpim]

### Design Overview

The purpose of this program is to support invoices from Direct Store Delivery and Escheatment sales audit transactions. Direct Store Delivery invoices refer to products or services that are delivered to the store and paid out at the store. This program will take DSD invoices that have been staged to the SA\_TRAN\_HEAD table by the saimptlog.pc program and move them into the INVC\_HEAD table. All DSD transactions will be assumed paid. They can be assumed received if there is a proof of delivery number listed on them. Transactions with a vendor invoice ID or a proof of delivery number should be matched to any existing invoice in INVC\_HEAD, and that invoice updated with the new information being interfaced. Invoices that do not match an existing invoice in INVC\_HEAD will need to be inserted. Each transaction will be exported to INVC\_HEAD table only once.

The Sales Audit Transaction type used to identify invoices for Direct Store Delivery transactions will be "Paid Out". Transaction types are stored on the codes tables with a code\_type = 'TRAT'. The Paid Out transaction has a code of 'PAIDOU'. The Sales Audit sub-transaction types will be used to identify whether the invoice is an "Expense Vendor Payout" or a "Merchandise Vendor Payout". These types are stored on the codes table with a code\_type = 'TRAS'. The codes will be 'EV' for Expense Vendor Payout and 'MV' for Merchandise Vendor Payout. Any Paid Out transaction with a sub transaction type of Expense Vendor will create a non-merchandise invoice and cause a record to be written to the INVC\_NON\_MERCH table. ReSA will store non-merchandise codes in the reason\_code field on sa\_tran\_head. Valid values for these reason codes should correspond to the codes stored on the non\_merch\_code\_head table.

In addition to DSD invoices, this program will also interface Escheatment totals to Invoice Matching. Escheatment is the process where an unredeemed gift certificate/voucher or credit voucher will, after a set period of time, be paid out as income to the issuing Retailer or in some states, the State receives this escheatment income. ReSA will be the governing system that determines who receives this income, but Invoice Matching will send the totals, with the related Partner, to Accounts Payable. Escheatment information will be stored on the ReSA SA\_TOTALS table and will be used to create non-merchandise invoices in Invoice Matching. These invoices will be assumed not paid.

To accommodate Escheatment, a new calculation should be added to Sales Audit to create escheatment totals. ReSA automatically totals sales transactions based on calculation definitions that the customer's users have previously created using the online wizard. Whenever users create new calculation definitions or edit existing ones, they become part of the automated totaling process the next time that satotals.pc runs.

| Table                  | Operations Performed |        |        |        |
|------------------------|----------------------|--------|--------|--------|
|                        | Select               | Insert | Update | Delete |
| Period                 | Yes                  | No     | No     | No     |
| Sa_system_options      | Yes                  | No     | No     | No     |
| Sa_export_options      | Yes                  | No     | No     | No     |
| Sa_store_day           | Yes                  | No     | No     | No     |
| Sa_store_day_read_lock | Yes                  | Yes    | No     | Yes    |

|                        |     |     |     |    |
|------------------------|-----|-----|-----|----|
| <b>Sa_export_log</b>   | Yes | No  | Yes | No |
| <b>Sa_tran_head</b>    | Yes | No  | No  | No |
| <b>Sa_tran_tender</b>  | Yes | No  | No  | No |
| <b>Sa_exported</b>     | No  | Yes | No  | No |
| <b>Sa_exported_rev</b> | No  | Yes | No  | No |
| <b>Sa_total</b>        | Yes | No  | No  | No |
| <b>Sa_error</b>        | Yes | No  | No  | No |
| <b>Invc_head</b>       | Yes | Yes | Yes | No |
| <b>Invc_non_merch</b>  | Yes | Yes | Yes | No |
| <b>Terms</b>           | Yes | No  | No  | No |
| <b>Currency_rates</b>  | Yes | No  | No  | No |
| <b>Addr</b>            | Yes | No  | No  | No |

### Stored Procedures / Shared Modules (Maintainability)

libretek library functions:

- retek\_init/retek\_close/retek\_refresh\_thread - to initialize and close Retek's restart/recovery module

libresa library functions:

- fetchSysdate – to get the current date and time
- fetchSaSystemOptions – to get ReSA system options
- fetchStoreDayErrorCount – to determine whether there are errors for the current store\_day
- markStoreDayExported – to mark a store day as updated
- updateStoreDayExported – to force commit of status 'E' on sa\_export\_log for store days that have been exported
- getTotal – to fetch totals for escheatment invoices
- get\_lock – to establish a read lock on a store\_day
- release\_lock – to release a store\_day lock

### Packages:

- **INVC\_SQL.NEXT\_INVC\_ID**—generates a new invoice id
- **DIRECT\_STORE\_INVC\_SQL.CHECK\_INVC\_DUPS** – Checks if invoice exists.

### Input Specifications

#### 'Table-To-Table'

Foundation tables used by this program for reference include:

Sa\_store\_data  
 Sa\_system\_options  
 Sa\_export\_options  
 Sa\_error\_codes  
 Sa\_error\_impact

Transaction data tables used to drive the processing includes:

Sa\_tran\_head  
 Sa\_total  
 Sa\_tran\_tender  
 Sa\_exported  
 Sa\_error

**Sa\_total\_usage** – Add new Invoice Matching code ('IM') to the code\_detail table where code\_type = 'SAUT' (Sales Audit Usage Type). This code will be used as the invoice matching usage\_type. Using the ReSA on-line wizard, add a new 'Total' calculation for escheatment transactions.

| Total_id | Usage_type |
|----------|------------|
| ESCHEAT  | IM         |

**Sa\_store\_data** – add invoice matching code to code\_detail table where code\_type = 'SYSE'

Code\_detail

| Code_type | Code | Code_desc | Code_seq |
|-----------|------|-----------|----------|
| SYSE      | IM   | IM Export | 7        |

**Sa\_store\_data** – For each store that will require the Invoice Matching Export, add a record to the sa\_store\_data table with a system\_code of 'IM' (Invoice Matching) and imp\_exp of 'E' (Export).

| Store     | System_code | Imp_exp |
|-----------|-------------|---------|
| e.g. 1009 | IM          | E       |
|           |             |         |

**Sa\_export options** – add IM code to the export options table to identify this new export to Invoice Matching. Since this program will not follow full disclosure logic, the multiple\_export\_ind should be 'N' (No).

| System_code | Multiple_export_ind | Exp_detail_ind |
|-------------|---------------------|----------------|
| SYSE        | N                   | IM Export      |
|             |                     |                |

## Output Specifications

### 'Table-To-Table'

Updates/Inserts made to Invoice Matching tables:

Invc\_head  
 Invc\_non\_merch

## Function Level Description

### Init ()

1. Call retek\_init to determine the max\_commit\_ctr.
2. Get the system date and time using **fetchSysdate()**.
3. Call the function **fetchSaSystemOptions()** to retrieve the unit\_of\_work (UOW).

**Process ()**

1. Process() gets the store\_days to be exported using the first driving cursor.
2. For each store\_day to be exported, process() checks whether there are errors associated with this store\_day (**fetchStoreDayErrorCount()**). If so, processing for this store\_day is skipped and step 2 is repeated for the next store\_day.
3. Process() then attempts to obtain a read lock for this store day. If unsuccessful, processing for this store\_day is skipped and step 2 is repeated for the next store\_day.
4. Call **processStoreDay()** to process the store day transactions for DSD invoices.
5. Call **postInvoices()** to create the appropriate invc\_head and/or invc\_non\_merch records.
6. Call **processStoreDayTotals()** to process the escheatment totals
7. Call **markStoreDayExported()** – updates sa\_export\_log, sets status to exported
8. Call **release\_lock()** to release the lock on the store/day.
9. Steps 2-7 are repeated until there are no more store\_days.

**Final ()**

1. Clean up any memory used.
2. Call **updateStoreDayExported()** to update sa\_export\_log for the last store day and system code (SYSE\_IM).
3. Call **rettek\_close()**
4. Call **rettek\_refresh\_thread()**

**processStoreDay()**

1. This function uses the second driving cursor to retrieve all transactions associated with direct store delivery invoice transactions. This function should fetch the appropriate data for inserts into invc\_head and invc\_non\_merch tables.
2. Calls **PostInvoices()** posts the invoices to invc\_head and invc\_non\_merch tables.
3. Calls the library function **markTransactionExported()** to insert a record into sa\_exported.

**processStoreDayTotals()**

This function will loop through the library function **getTotals()** for the current store day and SYSE\_IM usage type.

The getTotals() function is being modified to include pass back ref\_no1, ref\_no2 and ref\_no3. Ref\_no1 will contain the partner\_id, ref\_no2 the partner\_type and ref\_no3 the escheat amount.

```
int getTotal(
 char *is_store_day_seq_no,
 const char *is_usage_type,
 char *os_total_seq_no,
 char *os_total_value,
 char *os_rev_no,
 char *os_total_type
 char *os_total_type_id,
 char os_ref_no1,
 char os_ref_no2,
 char os_ref_no3
 char *os_status,
 long il_max_counter,
```



long     il\_multiplier);

1. Call **post\_escheat\_invoice()** to create the appropriate invc\_head and/or invc\_non\_merch records.
2. Mark the total exported by calling the library function **markTotalExported()**.

#### **calcDueDate()**

Select duedays from the terms table for the terms associated with the supplier or partner on sa\_tran\_head.vendor\_no. Due\_date = invoice date + duedays. Invoice dates should be sa\_tran\_head.tran\_datetime that was fetched in the first cursor. The terms discount percent should be taken from the terms table.

```
SELECT TO_CHAR((TO_DATE(invoice date, 'YYYYMMDDHH24MISS') +
duedays),'YYYYMMDD'),
 percent
FROM terms
WHERE terms = terms of the supplier or partner
```

#### **getNewInvcId()**

This function should call the package INVC\_SQL.NEXT\_INVC\_ID to retrieve the next invoice id in the sequence. Include standard package error handling around the package call.

#### **postInvoices()**

Note \* Partner, supplier, and date information will all be validated in ReSA so there is no need to validate in the export. ReSA validates to ensure that merchandise invoices are only associated with a supplier, not a partner. Validates partner, date, and numbers as well.

Calls **check\_invc\_exists()** to check if the invoice exists on the invc\_head tables. If a record is found, update it, otherwise call **GetNewInvcId()** and insert a new record to invc\_head. For non-merchandise invoices, be sure to update/insert the invc\_non\_merch record as well. A record should only be written to invc\_non\_merch if the invoice is a non-merchandise invoice. This can be determined by looking at the sub\_tran\_type on sa\_tran\_head. If the sub\_tran\_type is 'EV', then it's a non-merchandise invoice. If it's 'MV', then it's a merchandise invoice. If the invoice does not exist on the invc\_head table call **post\_new\_invoice()**. If the invoice already exists on the invc\_head table call **update\_invoice()**.

1. Call **calcDueDate()** to determine what the due date is for each invoice.
2. Call **get\_invc\_head\_data()** to get other columns :  
terms,payment\_method,freight\_terms,currency\_code,exchange\_rate,addr\_key  
for the invoice associated with a supplier or partner.
3. Use the mapping below to determine what values to use when inserting into invc\_head and/or invc\_non\_merch.

#### **Field Mapping between ReSA and Invoice Matching**

| INVC_HEAD | Required? | Datatype   | Sales Audit Value                                                                      |
|-----------|-----------|------------|----------------------------------------------------------------------------------------|
| INVC_ID   | NOT NULL  | NUMBER(10) | If the invoice is not matched with an existing one, invc_id will be a system generated |

|                  |          |              |                                                                                                                                                                                                                                                                                                                                                  |
|------------------|----------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |          |              | number<br>(invc_sql.next_invc<br>id).                                                                                                                                                                                                                                                                                                            |
| INVC_TYPE        | NOT NULL | VARCHAR2(1)  | 'I' for Merchandise<br>Invoice,<br>'N' for Non-<br>merchandise Invoice                                                                                                                                                                                                                                                                           |
| SUPPLIER         |          | NUMBER(10)   | DSD -<br>Sa_tran_head.vendo<br>r_no<br>Escheat – NULL                                                                                                                                                                                                                                                                                            |
| EXT_REF_NO       |          | VARCHAR2(30) | DSD -<br>Sa_tran_head.vendo<br>r_invc_no<br>Escheat – 'E'<br>concatinated with<br>State or Partner ID<br>(Which will be<br>state)                                                                                                                                                                                                                |
| STATUS           | NOT NULL | VARCHAR2(1)  | Code for the status<br>of the invoice.<br>Valid values are U<br>for unmatched, R<br>for partially<br>matched, M for<br>matched, A for<br>approved and P for<br>posted. Invoice<br>statuses are held on<br>the codes table<br>under the code type<br>'IMST'.<br>DSD - Default to<br>'U' using IMST_U.<br>Escheat – Default to<br>'A' using IMST_A |
| EDI_INVC_IND     | NOT NULL | VARCHAR2(1)  | 'N'                                                                                                                                                                                                                                                                                                                                              |
| EDI_SENT_IND     | NOT NULL | VARCHAR2(1)  | 'N'                                                                                                                                                                                                                                                                                                                                              |
| MATCH_FAIL_IND   | NOT NULL | VARCHAR2(1)  | Indicates whether or<br>not an invoice has<br>failed a match<br>attempt. Valid<br>values are 'Y' or 'N'.<br>Default to 'N' using<br>YSNO_N.                                                                                                                                                                                                      |
| REF_INVC_ID      |          | NUMBER(10)   | N/A – Used for<br>types 'C', 'D', 'R'.<br>Default to NULL.                                                                                                                                                                                                                                                                                       |
| REF_RTV_ORDER_NO |          | NUMBER(6)    | N/A – Used for<br>types 'C', 'D', 'R'.                                                                                                                                                                                                                                                                                                           |

|                  |          |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------|----------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |          |              | Default to NULL.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| REF_PRICE_CHANGE |          | NUMBER(8)    | N/A – Used for types 'C', 'D', 'R'. Default to NULL.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| REF_RSN_CODE     |          | VARCHAR2(6)  | N/A – Used for types 'C', 'D', 'R'. Default to NULL.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| TERMS            |          | VARCHAR2(15) | Defaulted from sups or partner table if DSD transaction or partner table if escheatment.                                                                                                                                                                                                                                                                                                                                                                                  |
| DUE_DATE         | NOT NULL | DATE         | Defaulted based on terms of the supplier or partner.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| PAYMENT_METHOD   |          | VARCHAR2(6)  | Code identifying the payment method for the invoice, indicating how the invoice will be paid. Valid values include 'LC' for letter of credit, 'WT' for wire transfer, and 'OA' for open account. Other values maybe added by the client as desired. Payment methods will be held on the codes table under a code type of 'PAYM'. Defaulted based on the payment_method of the vendor on the invoice for Merchandise Invoice. Default to NULL for Non merchandise Invoice. |
| TERMS_DSCNT_PCT  |          | NUMBER(12,4) | Discount that will be applied to the invoice if the invoice is paid by the due date.                                                                                                                                                                                                                                                                                                                                                                                      |

|                               |          |              |                                                                                                                                                                 |
|-------------------------------|----------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                               |          |              | Default to the terms.pct of the terms associated with the vendor on the invoice.                                                                                |
| TERMS_DSCNT_APPL_IND          | NOT NULL | VARCHAR2(1)  | Indicates whether or not the terms discount has been applied to the total cost of the invoice or not. Valid values are 'Y' or 'N'. Default to 'N' using YSNO_N. |
| TERMS_DSCNT_APPL_NON_MRCH_IND | NOT NULL | VARCHAR2(1)  | This field will indicate if the specified terms discount should be applied to non-merchandise costs. Default to 'N' using YSNO_N.                               |
| FREIGHT_TERMS                 |          | VARCHAR2(2)  | Indicator that references the freight terms associated with the invoice. Default from sups table for merchandise invoice. Non Merchandise Invoice: NULL.        |
| CREATE_ID                     | NOT NULL | VARCHAR2(30) | 'ReSA'                                                                                                                                                          |
| CREATE_DATE                   | NOT NULL | DATE         | The data the invoice was entered in the system – vdate                                                                                                          |
| INVC_DATE                     | NOT NULL | DATE         | Date the invoice was issued by the supplier – sa_tran_head.tran_d atetime for existing invoice. For New Invoice default to vdate.                               |
| MATCH_ID                      |          | VARCHAR2(30) | Oracle user ID of the user that matched the invoice. Default to NULL.                                                                                           |
| MATCH_DATE                    |          | DATE         | Date the invoice was matched.                                                                                                                                   |

|                  |          |               |                                                                                                                                                         |
|------------------|----------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |          |               | Default to NULL.                                                                                                                                        |
| APPROVAL_ID      |          | VARCHAR2(30)  | Oracle user ID of the user that approved the invoice match. Default to NULL.                                                                            |
| APPROVAL_DATE    |          | DATE          | Date the invoice match was approved. Default to NULL.                                                                                                   |
| FORCE_PAY_IND    | NOT NULL | VARCHAR2(1)   | Indicates whether or not the invoice is to be force paid (paid before being matched to receipts). Valid values are Y or N. Default to 'N' using YSNO_N. |
| FORCE_PAY_ID     |          | VARCHAR2(30)  | Oracle ID of the user that marked the invoice for force payment. This field will only have a value if the force_pay_ind = 'Y'. Default to NULL.         |
| POST_DATE        |          | DATE          | Date the invoice was posted to the AP staging tables. Default to NULL.                                                                                  |
| CURRENCY_CODE    | NOT NULL | VARCHAR2(3)   | Code identifying the currency in which the invoice is held. Default the supplier's or partner's currency.                                               |
| EXCHANGE_RATE    |          | NUMBER(20,10) | Exchange rate at which the invoice is held. Default from currency_rates table based on the currency of the supplier/partner.                            |
| TOTAL_MERCH_COST |          | NUMBER(20,4)  | Total merchandise cost for the invoice. This field will be held in the invoice currency. For DSD merchandise                                            |

|              |          |              |                                                                                                                                                                                                                                 |
|--------------|----------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              |          |              | invoices, this field should hold the total cost of the invoice (sa_tran_tender.tender_amt). Default to NULL for DSD non-merchandise and escheatment invoices.                                                                   |
| TOTAL_QTY    |          | NUMBER(12,4) | Total quantity of items on the invoice. This field is optional, and only needs to be entered if total quantity matching will be performed on the invoice. Quantity will not be captured in ReSA. Default to NULL.               |
| DIRECT_IND   | NOT NULL | VARCHAR2(1)  | Indicates whether the invoice was created for a direct store delivery order via the Quick Order Entry form in which the invoice was already paid. Valid values are 'Y' –Yes and 'N' -No.<br><b>Default to 'Y' using YSNO Y.</b> |
| PARTNER_TYPE |          | VARCHAR2(6)  | Type of partner assigned to the invoice. This field will always be 'EV' for Expense Vendor. Default using PTAL_EV.                                                                                                              |
| PARTNER_ID   |          | VARCHAR2(10) | DSD -<br>Sa_tran_head.vendor_no<br>Escheatment –<br>ref_no1 (partner_id)<br>Partner assigned to the invoice.<br>Partners can be assigned to any invoice type except                                                             |

|                      |          |               |                                                                                                                                                                                                         |
|----------------------|----------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                      |          |               | merchandise invoices.                                                                                                                                                                                   |
| ADDR_KEY             | NOT NULL | NUMBER(6)     | Indicates which vendor invoice address should be associated with the invoice.<br>Default to the primary address for the invoice address type on ADDR table (Addr_type = 05 and primary_addr_ind = 'Y'). |
| PAID_IND             | NOT NULL | VARCHAR2(1)   | Default to 'Y' (YSNO_Y) for all DSD transactions – they will be assumed paid. Set to 'N' (YSNO_N) for Escheatment invoices – they will be assumed not paid.                                             |
| PAYMENT_REF_NO       |          | VARCHAR2(13)  | DSD -<br>Sa_tran_head.payment_ref_no<br>Escheat – NULL                                                                                                                                                  |
| PAYMENT_DATE         |          | DATE          | Date that the invoice was paid from the POS system –<br>sa_tran_head.transaction_time.<br>Escheat – NULL                                                                                                |
| PROOF_OF_DELIVERY_NO |          | VARCHAR2(30)  | DSD –<br>Sa_tran_head.proof_of_delivery_no<br>Escheat – NULL                                                                                                                                            |
| CE_ID                |          | NUMBER(10)    | NULL                                                                                                                                                                                                    |
| OBLIGATION_KEY       |          | NUMBER(10)    | NULL                                                                                                                                                                                                    |
| COMMENTS             |          | VARCHAR2(255) | NULL                                                                                                                                                                                                    |

| INVC_NON_MERCH | Required? | Datatype   | Sales Audit Value                                                        |
|----------------|-----------|------------|--------------------------------------------------------------------------|
| INVC_ID        | NOT NULL  | NUMBER(10) | The invc_id on invc_non_merch should correspond to the one on invc_head. |

|                  |          |              |                                                                                                                                                                                                                                                                                                                                      |
|------------------|----------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NON_MERCH_CODE   | NOT NULL | VARCHAR2(6)  | Code identifying the non-merchandise cost being added to the invoice. These codes will be held on the non_merch_code_head table .<br>For DSD transactions, this field should be set to the reason_code from sa_tran_head. Should be 'E' for Escheatment invoices.                                                                    |
| NON_MERCH_AMT    |          | NUMBER(20,4) | Amount of the non-merchandise cost, specified by the non-merchandise code that has been invoiced for. This field will be held in the invoice currency.<br>DSD – sa_tran_tender.tender_amt<br>Escheat – total_value from <b>getTotals()</b>                                                                                           |
| VAT_CODE         |          | VARCHAR2(6)  | Default to NULL.                                                                                                                                                                                                                                                                                                                     |
| SERVICE_PERF_IND | NOT NULL | VARCHAR2(1)  | Indicates if a service non-merchandise cost has actually been performed. Valid values are 'Y' (service has been performed) or 'N' (service has not been performed or non-merchandise cost is not a service cost).<br>For DSD, if proof of delivery is provided, this field should be 'Y' (YSNO_Y). Should be set to 'N' (YSNO_N) for |



|       |  |           |                                                                                                                                      |
|-------|--|-----------|--------------------------------------------------------------------------------------------------------------------------------------|
|       |  |           | escheatment invoices.                                                                                                                |
| STORE |  | NUMBER(4) | Indicates the store at which the service was performed. Should be populated with the DSD invoice store number. Null for escheatment. |

### Scheduling Considerations

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: This module should be executed after the ReSA transaction import process.

Threading Scheme: N/A

### Locking Strategy

Locking will be performed via the `get_lock` and `release_lock` library functions to lock the store-day during processing.

### Restart/Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted based on the `commit_max_ctr` specified on the `restart_control` table. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed.

In case of failure, we rollback all work done to the point right after the call to `get_lock()` and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into `sa_exported` for one store\_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to `get_lock()`.

Restart recovery is implicit in the program, as only store\_days with a `sa_export_log.status` of 'R'eady (SAES\_R) will be selected for processing. Since we set this status to 'E'xported (SAES\_R) after a store\_day is processed, then on restart, store\_days that have been processed will be skipped.

### Driving cursors:

The program has three driving cursors: one to fetch store\_days to be exported, another to fetch invoice matching transactions to be exported for the store\_days fetched in the first cursor, and the last to fetch escheatment totals to be exported to Invoice Matching.

The following cursor will be used to retrieve the valid store/day identifiers that must be processed:

```
SELECT sd.store_day_seq_no,
 TO_CHAR(sd.business_date, 'YYYYMMDD'),
```

```

 el.seq_no,
 sd.store
 FROM sa_store_day sd, sa_export_log el
 WHERE sd.store_day_seq_no = el.store_day_seq_no
 AND sd.store_status = :SASS_C /* Closed */
 AND sd.data_status = :SADS_F /* Fully Loaded */
 AND sd.audit_status IN (:SAAS_A, :SAAS_S, :SAAS_H) / *Audit Pending, Store Errors
 Pending, HQ Errors Pending */
 AND el.system_code = :SYSE_IM
 AND el.status = :SAES_R /* Ready to be exported */
 ORDER BY sd.store, sd.business_date;

```

The second driving cursor selects DSD invoice transactions from transaction tables ('PADIOU').

```

SELECT h.tran_seq_no,
 h.rev_no,
 TO_CHAR(h.tran_datetime,'YYYYMMDDHH24MISS'),
 h.tran_no,
 h.tran_type,
 h.sub_tran_type,
 h.reason_code,
 h.vendor_no,
 h.vendor_inv_no,
 h.payment_ref_no,
 h.proof_of_delivery_no,
 h.status,
 t.tender_amt
 :SAFD_P /* Positive Transaction/Total */
 FROM sa_tran_head h
 WHERE h.store_day_seq_no = :is_store_day_seq_no
 AND h.tran_seq_no = t.tran_seq_no
 AND h.tran_type IN (:TRAT_PAIDOU)
 AND h.status != :SAST_D
 AND h.sub_tran_type IN ('EV','MV')
 AND NOT EXISTS
 (SELECT ex.tran_seq_no
 FROM sa_exported ex
 WHERE ex.tran_seq_no = h.tran_seq_no
 AND ex.store_day_seq_no = :is_store_day_seq_no)
 AND NOT EXISTS /* and no errors for the transaction. */
 (SELECT er.tran_seq_no
 FROM sa_error er, sa_error_impact ei, sa_tran_head h
 WHERE h.tran_seq_no = er.tran_seq_no
 AND er.error_code = ei.error_code
 AND ei.system_code = :SYSE_IM
 AND er.hq_override_ind != :YSNO_Y))
 ORDER BY h.tran_datetime;

```

The last driving cursor is embedded in the **getTotals()** function. This function is called with a usage type of SYSE\_IM. For each escheatment transaction that is processed, write to the invc\_head and invc\_non\_march tables.

**Design Assumptions**

- All DSD invoices will be assumed paid.
- All Escheatment invoices will be assumed not paid.

## Sales audit export to RDW [saexprdw]

### Purpose

The Batch Detailed Design is a thorough definition of a single batch program / module within one functional area. The documented information is derived from this functional area's Technical Design.

### Objectives

This Batch Detailed Design must:

- Document specific functions for a single batch program,
- Enable project team review, validation and consensus regarding the individual batch program's scope,
- Document the batch program in preparation for and in response to prototyping, and
- Prepare for and provide a defined and documented framework in which to perform Development Phase activities.

A Batch Detailed Design should **not** include code (SQL).

### Functional Area

### Modules Affected

### Design Overview

The purpose of this batch module is to fetch all corrected sale and return transactions that do not have RDW errors from the Retek Sales Audit (ReSA) database tables for transmission to the Retek Data Warehouse (RDW). The data will be sent at the store day level. If the transaction has a status of Deleted and it has previously been transmitted, a reversal of the transaction will be sent.

Four files of type RDWT, RDWF, RDWS and RDWC will be created for each store\_day. See the file Interface File – SA to RDW.doc for more information.

RDW requires that the employee id be sent. saexprdw is expected to do this by mapping a cashier ID to an employee ID using the sa\_store\_emp table. However, the latter may not always be populated and thus, we send a blank field to RDW in this case.

| Table           | <i>Operations Performed</i> |        |        |        |
|-----------------|-----------------------------|--------|--------|--------|
|                 | Select                      | Insert | Update | Delete |
| sa_store_day    | Yes                         | No     | No     | No     |
| sa_export_log   | Yes                         | No     | Yes    | No     |
| sa_error        | Yes                         | No     | No     | No     |
| sa_error_impact | Yes                         | No     | No     | No     |

|                    |     |     |    |    |
|--------------------|-----|-----|----|----|
| sa_tran_head       | Yes | No  | No | No |
| sa_tran_item       | Yes | No  | No | No |
| sa_tran_disc       | Yes | No  | No | No |
| sa_tran_tender     | Yes | No  | No | No |
| sa_customer        | Yes | No  | No | No |
| sa_tran_head_rev   | Yes | No  | No | No |
| sa_tran_item_rev   | Yes | No  | No | No |
| sa_tran_disc_rev   | Yes | No  | No | No |
| sa_tran_tender_rev | Yes | No  | No | No |
| sa_store_emp       | Yes | No  | No | No |
| sa_total           | Yes | No  | No | No |
| sa_exported        | Yes | Yes | No | No |
| sa_exported_rev    | Yes | No  | No | No |

### Program Flow

This is a detailed diagram (structure chart type or function level text) that will define all functions performed in the module. This section will require the designer to specify the program flow and activities performed in each code segment without detailing SQL.

Thoroughly analyze the module with the sole objective of achieving the most efficient implementation. Uncover parallelism inherent in the application. Identify functional modules executed concurrently and the dependencies between these modules. In addition, identify standard computational modules, scope, performance, scheduling constraints, common functions, maintainability, and overall module integrity.

This should provide an overall picture of the LUW performed by the batch module, derived from the Technical Design.

### Global Variable Descriptions

| Gobal Variable     | Description                                                 |
|--------------------|-------------------------------------------------------------|
| pl_commit_max_ctr  | Commit max counter used for array fetches.                  |
|                    |                                                             |
| ps_sysdate         | Current sysdate value from the database.                    |
| ps_store           | Store ID from store/day driving cursor.                     |
| ps_business_date   | Business date from store/day driving cursor.                |
| ps_temp_rdwtfile   | Temporary file name to be used for the RDWT file.           |
| ps_temp_rdwffile   | Temporary file name to be used for the RDWF file.           |
| ps_temp_rdwsfile   | Temporary file name to be used for the RDWS file.           |
| ps_temp_rdwcf file | Temporary file name to be used for the RDWC file.           |
|                    |                                                             |
|                    |                                                             |
|                    |                                                             |
|                    |                                                             |
|                    |                                                             |
| pi_curtrat         | Current transactions transaction type converted to an enum. |
|                    |                                                             |
|                    |                                                             |
|                    |                                                             |

|                           |                                                                                                                                                                                                                                                                                                                                            |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                           |                                                                                                                                                                                                                                                                                                                                            |
|                           |                                                                                                                                                                                                                                                                                                                                            |
|                           |                                                                                                                                                                                                                                                                                                                                            |
|                           |                                                                                                                                                                                                                                                                                                                                            |
|                           |                                                                                                                                                                                                                                                                                                                                            |
|                           |                                                                                                                                                                                                                                                                                                                                            |
|                           |                                                                                                                                                                                                                                                                                                                                            |
| pi_tdetl_count            | TDETL record count for TTAIL record in the RDWT file.                                                                                                                                                                                                                                                                                      |
| -                         | -                                                                                                                                                                                                                                                                                                                                          |
| ps_total_sales_value      | Total sales value of a TITEM record minus any discounts from associated IDISC records.                                                                                                                                                                                                                                                     |
| pl_rdwc_line_ctr          | Line counter for the RDWC file.                                                                                                                                                                                                                                                                                                            |
| pl_rdwf_line_ctr          | Line counter for the RDWF file.                                                                                                                                                                                                                                                                                                            |
| pl_rdws_line_ctr          | Line counter for the RDWS file.                                                                                                                                                                                                                                                                                                            |
| pl_rdwt_line_ctr          | Line counter for the RDWT file.                                                                                                                                                                                                                                                                                                            |
| RDWFFile                  | File pointer for the RDWF file.                                                                                                                                                                                                                                                                                                            |
| RDWTFile                  | File pointer for the RDWT file.                                                                                                                                                                                                                                                                                                            |
| RDWSFile                  | File pointer for the RDWS file.                                                                                                                                                                                                                                                                                                            |
| RDWCFile                  | File pointer for the RDWC file.                                                                                                                                                                                                                                                                                                            |
| pi_num_locks_not_released | Counter for the number of store/day locks that could not be released.                                                                                                                                                                                                                                                                      |
| pi_num_non_fatal_errors   | Counter for the number of non-fatal errors encountered: <ul style="list-style-type: none"> <li>• Store/day lock could not be release.</li> <li>• An unexpected total was encountered.</li> <li>• Could not translate a cashier POS ID to an employee ID.</li> <li>• Could not translate a salesperson POS ID to an employee ID.</li> </ul> |

### Function Level Description

#### main()

**int argc**  
**char \*argv[]**

Check command line for required arguments.  
Call **LOGON** to connect to the database.  
Call **Init** to initialize the program.  
Call **process** to export the available RDW data.  
Report unlocking errors.  
Report non-fatal errors.  
Call **final** to cleanup.

#### init()

**No arguments**

This function initializes Restart recovery.  
Get the value of sa\_system\_options.unit\_of\_work by calling the library function **fetchSaSystemOptions**.  
Initialize Oracle Number functions by calling **OraNumInit**.  
Get temporary filenames to use for generating the output files. Store these names in ps\_temp\_rdwtfile, ps\_temp\_rdwffile, ps\_temp\_rdwsfile, and ps\_temp\_rdwcf file.

**process()****No arguments**

Picks a store/day to be processed by fetching using the first driving cursor. Save the store ID in `ps_store` and the date in `ps_business_date`.

Attempt to lock the store/day with a call to **get\_lock**. If this fails, go on to the next store/day.

Open RDWTFile, RDWSFile, RDWCFile and RDWFile, using temporary names generated in **init**.

Set `pl_rdwc_line_ctr`, `pl_rdwf_line_ctr`, `pl_rdws_line_ctr` and `pl_rdwt_line_ctr` to 0.

Call **fetchSysDate** to get the current date/time. Store it in `ps_sysdate`.

Call **WrRDWFHead** to write a RDWT FHEAD record to the RDWT file.

Call **WrRDWFHead** to write a RDWF FHEAD record to the RDWF file.

Call **processStoreDay** to process the store/days transactions.

Call **WrOutputData** to write the data in memory to the appropriate file.

Increment `pl_rdwt_line_ctr`.

Call **WrRDWFTail** to write a RDWT FTAIL record to the RDWT file.

Call **WrRDWFTail** to write a RDWF FTAIL record to the RDWF file.

Call **processStoreDayTotals** to process all totals for a given store day.

Update the status in `sa_export_log` to Complete by calling the library function

**markStoreDayExported**.

Close the RDWTFile, RDWFFile, RDWSFile and RDWCFile and rename them appropriately (*file-type\_store\_business-date\_current-datetime*).

Call to **release\_lock** and go on to the next store/day. This function commits as a side effect, thus committing the changes to the database.

**final()****int ii\_process\_ret**

Remove the temporary file, if we failed to finish (`ii_proces_ret` is not OK).

.

Call **retek\_close**.

Call **retek\_refresh\_thread**.

**processStoreDay()****char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

For each transaction from the store/day being processed, get the following information from the second driving cursor and call **processTransHead** with the information.

| Table        | Column        | Description                                                  |
|--------------|---------------|--------------------------------------------------------------|
| Sa_tran_head | Tran_seq_no   |                                                              |
| Sa_tran_head | Rev_no        |                                                              |
| Sa_tran_head | Tran_datetime | Format YYYYMMDDHH24MISS                                      |
| Sa_tran_head | Tran_no       |                                                              |
| Sa_tran_head | Register      |                                                              |
| Sa_store_emp | Emp_id        | Pos_id = cashier via an outer join separate from salesperson |
| Sa_store_emp | Emp_id        | Pos_id = salesperson via an outer join separate from cashier |
| Sa_customer  | Cust_id_type  | via an outer join                                            |
| Sa_customer  | Cust_id       | via an outer join                                            |
| Sa_tran_head | Reason_code   |                                                              |
| Sa_tran_head | Tran_type     |                                                              |
| Sa_tran_head | Sub_tran_type |                                                              |

|              |                  |                                                                                                                                                                                                                                                |
|--------------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sa_tran_head | Orig_tran_no     |                                                                                                                                                                                                                                                |
| Sa_tran_head | Orig_reg_no      |                                                                                                                                                                                                                                                |
| Sa_tran_head | Ref_no1          |                                                                                                                                                                                                                                                |
| Sa_tran_head | Ref_no2          |                                                                                                                                                                                                                                                |
| Sa_tran_head | Ref_no3          |                                                                                                                                                                                                                                                |
| Sa_tran_head | Ref_no4          |                                                                                                                                                                                                                                                |
| Sa_tran_head | Vendor_no        |                                                                                                                                                                                                                                                |
| Sa_tran_head | Status           |                                                                                                                                                                                                                                                |
| Sa_tran_head | Value            | 'SIGN_N' or 'SIGN_P' depending on the sign of value.                                                                                                                                                                                           |
| Sa_tran_head | Value            | Absolute value multiplied by 10000.                                                                                                                                                                                                            |
|              | Transaction Sign | 'SAFD_P' if the transaction has not been deleted (status != 'SAST_D') and there are no errors and it has not been exported.<br>'SAFD_N' if the transaction has been deleted (status = 'SAST_D') and it has been exported after being exported. |
| Sa_exported  | Exp_datetime     | Only for transactions with a Transaction Sign of 'SAFD_N'.<br>Format YYYYMMDDHH24MISS                                                                                                                                                          |
|              |                  |                                                                                                                                                                                                                                                |
|              |                  |                                                                                                                                                                                                                                                |
|              |                  |                                                                                                                                                                                                                                                |

Calls the library function **markTransactionExported** to insert a record into sa\_exported for each transaction.

#### **processTransHead()**

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**struct pt\_sa\_tran\_head ir\_sa\_tran\_head**

If the transaction status is deleted (SAST\_D) and it has been previously exported, then call **retrieveTransHeadRev**. Also, if the revision number of the transaction is not 1, then a previous revision may have been exported; call **retrieveTransHeadRev** to get the exported revision (for full disclosure purposes).

Call **retrieveTransItem**, **retrieveTransDisc** and **retrieveTransTender** to obtain the items, discounts and tenders for the transaction, both Positive transactions and Negative ones.

Call **saveData** for both the Positive and Negative transactions to write the information into the RDW files.

#### **retrieveTransHeadRev()**

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**struct pt\_sa\_tran\_head \*or\_sa\_tran\_head\_rev**

This function gets the sa\_tran\_head\_rev record that needs to be processed. A record needs to be processed if it has been previously exported.

| Table            | Column        | Description                                                  |
|------------------|---------------|--------------------------------------------------------------|
| Sa_tran_head_rev | Tran_seq_no   |                                                              |
| Sa_tran_head_rev | Rev_no        |                                                              |
| Sa_tran_head_rev | Tran_datetime | Format YYYYMMDDHH24MISS                                      |
| Sa_tran_head_rev | Tran_no       |                                                              |
| Sa_tran_head_rev | Register      |                                                              |
| Sa_store_emp     | Emp_id        | Pos_id = cashier via an outer join separate from salesperson |
| Sa_store_emp     | Emp_id        | Pos_id = salesperson via an outer join separate from cashier |



|                  |                  |                                                                                       |
|------------------|------------------|---------------------------------------------------------------------------------------|
| Sa_customer      | Cust_id_type     | via an outer join                                                                     |
| Sa_customer      | Cust_id          | via an outer join                                                                     |
| Sa_tran_head_rev | Reason_code      |                                                                                       |
| Sa_tran_head_rev | Tran_type        |                                                                                       |
| Sa_tran_head_rev | Sub_tran_type    |                                                                                       |
| Sa_tran_head_rev | Orig_tran_no     |                                                                                       |
| Sa_tran_head_rev | Orig_reg_no      |                                                                                       |
| Sa_tran_head_rev | Ref_no1          |                                                                                       |
| Sa_tran_head_rev | Ref_no2          |                                                                                       |
| Sa_tran_head_rev | Ref_no3          |                                                                                       |
| Sa_tran_head_rev | Ref_no4          |                                                                                       |
| Sa_tran_head_rev | Vendor_no        |                                                                                       |
| Sa_tran_head_rev | Status           |                                                                                       |
| Sa_tran_head_rev | Value            | 'SIGN_N' or 'SIGN_P' depending on the sign of value.                                  |
| Sa_tran_head_rev | Value            | Absolute value multiplied by 10000.                                                   |
|                  | Transaction Sign | 'SAFD_N'                                                                              |
| Sa_exported_rev  | Exp_datetime     | Only for transactions with a Transaction Sign of 'SAFD_N'.<br>Format YYYYMMDDHH24MISS |
|                  |                  |                                                                                       |

If no data is found, than set or\_sa\_tran\_head\_rev->s\_rev\_no to -1.

#### retrieveTransItem()

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**char is\_rev\_no[NULL\_SA\_REV\_NO]**

**long \*ol\_num\_sa\_tran\_item**

**struct pt\_sa\_tran\_item \*\*or\_sa\_tran\_item**

This function gets all sa\_tran\_item records or sa\_tran\_item\_rev (if is\_rev\_no is not -1) that need to be processed for a tran\_seq\_no.

| Table        | Column               | Description                                                |
|--------------|----------------------|------------------------------------------------------------|
| Sa_tran_item | Tran_seq_no          |                                                            |
| Sa_tran_item | Item_seq_no          |                                                            |
| Sa_tran_item | Item_status          |                                                            |
| Sa_tran_item | Item                 |                                                            |
| Sa_tran_item | Ref_item             |                                                            |
| Sa_tran_item | Non_merch_item       |                                                            |
| Sa_tran_item | Voucher_no           |                                                            |
| Sa_tran_item | Dept                 |                                                            |
| Sa_tran_item | Class                |                                                            |
| Sa_tran_item | Subclass             |                                                            |
|              |                      |                                                            |
|              |                      |                                                            |
|              |                      |                                                            |
| Sa_tran_item | Standard_qty         | 'SIGN_N' or 'SIGN_P' depending on the sign of qty.         |
| Sa_tran_item | Standard_qty         | Absolute value multiplied by 10000.                        |
| Sa_tran_item | Standard_unit_retail | 'SIGN_N' or 'SIGN_P' depending on the sign of unit_retail. |

|              |                               |                                                                    |
|--------------|-------------------------------|--------------------------------------------------------------------|
| Sa_tran_item | Standard_unit_retail          | Absolute value multiplied by 10000.                                |
| Sa_tran_item | Tax_ind                       |                                                                    |
| Sa_tran_item | Item_swiped_ind               |                                                                    |
| Sa_tran_item | Standard_orig_unit_r<br>etail | 'SIGN_N' or 'SIGN_P' depending on the sign of<br>orig_unit_retail. |
| Sa_tran_item | Standard_orig_unit_r<br>etail | Absolute value multiplied by 10000.                                |
| Sa_tran_item | Item_type                     |                                                                    |
|              |                               |                                                                    |
|              |                               |                                                                    |
|              |                               |                                                                    |
| Sa_tran_item | Override_reason               |                                                                    |
| Sa_store_emp | Emp_id                        |                                                                    |
| Sa_tran_item | Return_reason_code            |                                                                    |
| Sa_tran_item | Drop_ship_ind                 |                                                                    |

The same columns as above are select from the sa\_tran\_item\_rev table if the rev\_no passed in is not -1.

Set \*ol\_num\_sa\_tran\_item to the total number of records fetched.

#### retrieveTransDisc()

char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]

char is\_rev\_no[NULL\_SA\_REV\_NO]

long \*ol\_num\_sa\_tran\_disc

struct pt\_sa\_tran\_disc \*\*or\_sa\_tran\_disc

This function gets all sa\_tran\_disc or sa\_tran\_disc\_rev records (if is\_rev\_no is not -1) for a tran\_seq\_no that needs to be processed.

| Table                        | Column                                                            | Description                                                         |
|------------------------------|-------------------------------------------------------------------|---------------------------------------------------------------------|
| Sa_tran_disc                 | Tran_seq_no                                                       |                                                                     |
| Sa_tran_disc                 | Item_seq_no                                                       |                                                                     |
| Sa_tran_disc                 | Discount_seq_no                                                   |                                                                     |
| Sa_tran_disc                 | Rms_promo_type                                                    |                                                                     |
| Sa_tran_disc                 | Promotion                                                         |                                                                     |
| Sa_tran_disc                 | Discount_type                                                     |                                                                     |
| Sa_tran_disc                 | Coupon_no                                                         |                                                                     |
| Sa_tran_disc                 | Coupon_ref_no                                                     |                                                                     |
| Sa_tran_disc                 | Standard_qty                                                      | 'SIGN_N' or 'SIGN_P' depending on the sign of<br>qty.               |
| Sa_tran_disc                 | Standard_qty                                                      | Absolute value multiplied by 10000.                                 |
| Sa_tran_disc<br>Sa_tran_item | (Unit_retail *<br>standard_qty) –<br>(unit_discount_amt *<br>qty) | Absolute value multiplied by 10000.                                 |
| Sa_tran_disc<br>Sa_tran_item | (Unit_retail *<br>standard_qty) –<br>(unit_discount_amt *<br>qty) | 'SIGN_N' or 'SIGN_P' depending on the sign of the<br>expression.    |
| Sa_tran_disc                 | Standard_unit_discount_<br>amt                                    | 'SIGN_N' or 'SIGN_P' depending on the sign of<br>unit_discount_amt. |

|              |                            |                                     |
|--------------|----------------------------|-------------------------------------|
| Sa_tran_disc | Standard_unit_discount_amt | Absolute value multiplied by 10000. |
| Sa_tran_disc |                            |                                     |

The same columns as above are select from the sa\_tran\_disc\_rev table if the rev\_no passed in is not -1.

Set \*ol\_num\_sa\_tran\_disc to the total number of records fetched.

#### retrieveTransTender()

char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]

char is\_rev\_no[NULL\_SA\_REV\_NO]

long \*ol\_num\_sa\_tran\_tender

struct pt\_sa\_tran\_tender \*\*or\_sa\_tran\_tender

This function gets all sa\_tran\_tender or sa\_tran\_tender\_rev records (if is\_rev\_no is not -1) for a tran\_seq\_no that needs to be processed.

| Table                      | Column                      | Description                                               |
|----------------------------|-----------------------------|-----------------------------------------------------------|
| Sa_tran_tender             | Tran_seq_no                 |                                                           |
| Sa_tran_tender             | Tender_seq_no               |                                                           |
| Sa_tran_tender             | Tender_type_group           |                                                           |
| Sa_tran_tender             | Tender_type_id              |                                                           |
| Sa_tran_tender             | Tender_amt                  | 'SIGN_N' or 'SIGN_P' depending on the sign of tender_amt. |
| Sa_tran_tender             | Tender_amt                  | Absolute value multiplied by 10000.                       |
| Sa_tran_tender             | Cc_no                       |                                                           |
| Sa_tran_tender             | Cc_auth_no                  |                                                           |
| Sa_tran_tender             | Cc_auth_src                 |                                                           |
| Sa_tran_tender             | Cc_cardholder_verf          |                                                           |
| Sa_tran_tender             | Cc_exp_date                 | Format YYYYMMDD                                           |
| Sa_tran_tender             | Cc_entry_mode               |                                                           |
| Sa_tran_tender             | Cc_term_id                  |                                                           |
| Sa_tran_tender             | Cc_spec_cond                |                                                           |
| Sa_tran_tender             | Voucher_no                  |                                                           |
| Sa_tran_head<br>Sa_voucher | Business_date –<br>iss_date | Voucher age                                               |
| Sa_voucher                 | Escheat_date                |                                                           |
| Sa_tran_tender             | Coupon_no                   |                                                           |
| Sa_tran_tender             | Coupon_ref_no               |                                                           |
|                            |                             |                                                           |

The same columns as above are select from the sa\_tran\_tender\_rev table if the rev\_no passed in is not -1.

Set \*ol\_num\_sa\_tran\_tender to the total number of records fetched.

#### saveData()

struct pt\_sa\_tran\_head ir\_sa\_tran\_head

long il\_num\_sa\_tran\_item

struct pt\_sa\_tran\_item \*ia\_sa\_tran\_item

long il\_num\_sa\_tran\_disc

struct pt\_sa\_tran\_disc \*ia\_sa\_tran\_disc

```

 long il_num_sa_tran_tender
 struct pt_sa_tran_tender *ia_sa_tran_tender

```

Set pi\_curtrat to the current transaction type by calling **trat\_lookup**.

Call **WrRDWTHead** to process the current ia\_sa\_tran\_head record if the transaction type (pi\_curtrat) is TRATTT\_COND, TRATTT\_PAIDIN or TRATTT\_PAIDOU.

For each item record:

Call tsv\_lookahead to calculate the total sales value for later use.

Call **WrRDWTHead** to process the current ia\_sa\_tran\_item record.

For each item's discount record:

Call **WrRDWTDetl** to process the current ia\_sa\_tran\_disc record.

For each tender record:

Call **WrRDWFDetl** to process the current ia\_sa\_tran\_tender.

Call **WrRDWTTail** to create a TTAIL record for the RDWT file.

### ProcessStoreDayTotals()

```

 char is_store_day_seq_no[NULL_BIG_SEQ_NO]
 const char is_usage_type[NULL_CODE]

```

This function will loop through the library function **getBalTotals** for the current store day.

Call **WrRDWFHead** to write this header to the RDWS file.

Call **WrRDWFHead** to write this header to the RDWC file.

For each total returned:

- 1 If the total\_id is "OVRSH\_T\_B" then write the data to the RDWC file.
- 2 Else, if the cashier\_id and the register\_id are both nulls, then write to the RDWS file.
- 3 Else, mark this as an error, since the RDWS file can only handle store level totals.
- 4 If the total is not a 'N'egative total, mark the total exported by calling the library function **markTotalExported**.

Call **WrRDWFTail** to write this header to the RDWS file.

Call **WrRDWFTail** to write this header to the RDWC file.

### tsv\_lookahead()

```

 int i

```

This function calculates the total sales value (ps\_total\_sales\_value) by "looking ahead" and summing up the item values and discounts for the current item record (i).

### WrRDWFHead()

```

 char *is_file_type
 FILE *is_file
 long *iol_line_ctr

```

Set \*iol\_line\_ctr to 1. This is the appropriate global line counter variable for the file type.

Writes an RDW\_FHEAD record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field                | Type | Size               | Source              |
|----------------------|------|--------------------|---------------------|
| frecdesc             | char | RDW_FRECDDESC_SIZE | RDW_FHEAD_FRECDDESC |
| flineid              | char | LEN_FILE_LINE_NO   | *iol_line_ctr       |
| file_type_definition | char | LEN_FILE_TYPE_DEF  | is_file_type        |
| file_create_date     | char | LEN_DATETIME       | ps_sysdate          |

Call **putrec** to write the record out to the RDWT or RDWF file.

### WrRDWTHead()

**pt\_sa\_tran\_head \*ir\_head**

**Pt\_sa\_tran\_item \*ir\_item**

Increment pl\_rdwt\_line\_ctr.

Set pi\_tdetl\_count to 0.

This function writes a RDW\_THEAD record (as defined in salib.h) to the output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field          | Type | Size               | Source                                                                                                       |
|----------------|------|--------------------|--------------------------------------------------------------------------------------------------------------|
| fredesc        | char | RDW_FRECDDESC_SIZE | RDW_THEAD_FRECDDESC                                                                                          |
| flineid        | char | LEN_FILE_LINE_NO   | pl_rdwt_line_ctr                                                                                             |
|                |      |                    |                                                                                                              |
| tran_datetime  | char | LEN_DATETIME       | ir_head->s_tran_datetime                                                                                     |
| Location       | char | LEN_LOC            | ps_store                                                                                                     |
| register_id    | char | LEN_REGISTER       | ir_head->s_register                                                                                          |
| cashier_id     | char | LEN_EMP_ID         | ir_head->s_cashier                                                                                           |
| Salesperson_id | char | LEN_EMP_ID         | ir_item->s_sales_person<br>if NULL than use ir_head->s_salesperson                                           |
| cust_id_type   | char | CIDT_SIZE          | ir_head->s_cust_id_type                                                                                      |
| cust_id_number | char | LEN_CUST_ID        | ir_head->s_cust_id                                                                                           |
| tran_no        | char | LEN_TRAN_NO        | ir_head->s_tran_no                                                                                           |
| Orig_register  | Char | LEN_REGISTER       | Ir_head->s_orig_register                                                                                     |
| Orig_tran_no   | Char | LEN_TRAN_NO        | Ir_head->s_orig_tran_no                                                                                      |
| tran_seq_no    | char | LEN_BIG_SEQ_NO     | ir_head->s_tran_seq_no                                                                                       |
| rev_no         | char | LEN_SA_REV_NO      | ir_head->s_rev_no                                                                                            |
| tran_sign      | char | LEN_IND            | ir_head->s_tran_sign                                                                                         |
| tran_type      | char | TRAT_SIZE          | ir_head->s_tran_type                                                                                         |
| sub_tran_type  | char | TRAS_SIZE          | ir_head->s_tran_sub_type                                                                                     |
| emp_cashier_no | char | LEN_EMP_ID         | ir_head->s_ref_no1 if sub_tran_type =<br>TRAS_EMP                                                            |
| receipt_ind    | char | LEN_IND            | ir_head->s_ref_no1 if tran_type =<br>TRAT_RETURN                                                             |
| reason_code    | char | REAC_SIZE          | ir_head->s_reason_code                                                                                       |
| vendor_no      | char | LEN_VENDOR_NO      | ir_head->s_ref_no1 if tran_type =<br>TRAT_PAIDOU                                                             |
| item_type      | char | SAIT_SIZE          | SAIT_ITEM if ir_item->s_item_type is<br>either SAIT_ITEM or SAIT_REF.<br>SAIT_GCN if ir_item->s_item_type is |

|                    |      |                |                                                                                                                                              |
|--------------------|------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------|
|                    |      |                | SAIT_GCN.                                                                                                                                    |
| item_no            | char | LEN_ITEM_NO    | Ir_item->s_item if ir_item->s_item_type is SAIT_ITEM.<br>Ir_item->s_voucher_no if ir_item->s_item_type is SAIT_GCN.                          |
| tax_ind            | char | LEN_IND        | ir_item->s_tax_ind                                                                                                                           |
| item_swiped_ind    | char | LEN_IND        | ir_item->s_item_swiped_ind                                                                                                                   |
| Dept               | char | LEN_DEPT       | ir_item->s_dept                                                                                                                              |
| Class              | char | LEN_CLASS      | ir_item->s_class                                                                                                                             |
| Subclass           | char | LEN_SUBCLASS   | ir_item->s_subclass                                                                                                                          |
| total_sales_qty    | char | LEN_QTY        | ir_item->s_qty                                                                                                                               |
| total_sales_value  | char | LEN_AMT        | ps_total_sales_value if tran_type is not TRAT_COND, TRAT_PADIN or TRAT_PAIDOU.<br>ir_head->value if tran_type is TRAT_PAIDIN or TRAT_PAIDOU. |
| override_reason    | char | ORRC_SIZE      | ir_item->s_override_reason                                                                                                                   |
| Return_reason_code | Char | SARR_SIZE      | Ir_item->s_return_reason_code                                                                                                                |
| total_orig_sign    | char | LEN_SALES_SIGN | ir_item->s_qty_sign                                                                                                                          |
| total_orig_value   | char | LEN_AMT        | ir_item->s_qty * ir_item->s_orig_unit_retail / 10000                                                                                         |
| Weather            | char | LEN_CODE       | ir_head->s_ref_no1 if tran_type is TRAT_COND                                                                                                 |
| Temperature        | char | LEN_CODE       | ir_head->s_ref_no2 if tran_type is TRAT_COND                                                                                                 |
| Traffic            | char | LEN_CODE       | ir_head->s_ref_no3 if tran_type is TRAT_COND                                                                                                 |
| Construction       | char | LEN_CODE       | ir_head->s_ref_no4 if tran_type is TRAT_COND                                                                                                 |
|                    |      |                |                                                                                                                                              |

Call **putrec** to write the record out to the RDWT file.

**WrRDWTDetl()****pt\_sa\_tran\_head \*ir\_head****ps\_sa\_tran\_disc \*ir\_disc**

Increment both pl\_rdwt\_line\_ctr and pl\_tdetl\_count.

Writes an RDW\_TDETL record (as defined in salib.h) to the RDWT output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field           | Type | Size               | Source                               |
|-----------------|------|--------------------|--------------------------------------|
| freccdesc       | char | RDW_FRECCDESC_SIZE | RDW_TDETL_FRECCDESC                  |
| flneid          | char | LEN_FILE_LINE_NO   | pl_rdwt_line_ctr                     |
| Discount_type   | Char | SADT_SIZE          | <b>ir_disc-&gt; s_discount_type</b>  |
| promo_tran_type | char | PRMT_SIZE          | <b>ir_disc-&gt; s_rms_promo_type</b> |
| promo_no        | char | LEN_PROMOTION      | <b>ir_disc-&gt; s_disc_ref_no</b>    |
| tran_sign       | char | LEN_IND            | ir_head-> s_tran_sign                |
| Coupon_no       | Char | LEN_COUPON_NO      | <b>ir_disc-&gt; s_coupon_no</b>      |
| Coupon_ref_no   | Char | LEN_COUPON_REF_NO  | <b>ir_disc-&gt; s_coupon_ref_no</b>  |
| sales_qty       | char | LEN_QTY            | <b>ir_disc-&gt; s_qty</b>            |
| sales_sign      | char | LEN_SALES_SIGN     | <b>ir_disc-&gt; s_qty_sign</b>       |
| sales_value     | char | LEN_AMT            | ps_total_sales_value                 |
| disc_value      | char | LEN_AMT            | <b>ir_disc-&gt; s_unit_disc_amt</b>  |
|                 |      |                    |                                      |

Call **putrec** to write the record out to the RDWT file.**WrRDWTTail()****No arguments**

Increment pl\_rdwt\_line\_ctr.

Writes an RDW\_TTAIL record (as defined in salib.h) to the RDWT output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field            | Type | Size               | Source              |
|------------------|------|--------------------|---------------------|
| freccdesc        | char | RDW_FRECCDESC_SIZE | RDW_TTAIL_FRECCDESC |
| flneid           | char | LEN_FILE_LINE_NO   | pl_rdwt_line_ctr    |
| tran_rec_counter | char | LEN_DTL_LINE_CNT   | pl_tdetl_count      |
|                  |      |                    |                     |

Call **putrec** to write the record out to the RDWT file.**WrRDWFTail()****FILE \*is\_file****long \*iol\_line\_ctr**

Increments \*iol\_line\_ctr. This is the appropriate global line counter variable for the file type.

Writes an RDW\_FTAL record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field     | Type | Size               | Source             |
|-----------|------|--------------------|--------------------|
| freccdesc | char | RDW_FRECCDESC_SIZE | RDW_FTAL_FRECCDESC |
|           |      |                    |                    |

|                  |      |                  |                   |
|------------------|------|------------------|-------------------|
| flineid          | char | LEN_FILE_LINE_NO | *iol_line_ctr     |
| file_rec_counter | char | LEN_DTL_LINE_CNT | *iol_line_ctr - 2 |
|                  |      |                  |                   |

Call **putrec** to write the record out to the RDWT or RDWF file.

**WrRDWSTDctl()**

```

char *is_status
char *is_total_id
char *is_ref_no1
char *is_ref_no2
char *is_ref_no3
char *is_total_value

```

Increment pl\_rdws\_line\_ctr.

Writes an RDWS\_TDETL record (as defined in salib.h) to the RDWS output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field        | Type | Size              | Source                                                                   |
|--------------|------|-------------------|--------------------------------------------------------------------------|
| frecdesc     | char | RDW_FRECDISC_SIZE | RDW_FDETL_FRECDISC                                                       |
| flineid      | char | LEN_FILE_LINE_NO  | pl_rdws_line_ctr                                                         |
| tran_date    | char | LEN_DATEONLY      | ps_business_date                                                         |
| location     | char | LEN_LOC           | ps_store                                                                 |
| sales_sign   | char | LEN_SALES_SIGN    | is_status                                                                |
| total_id     | char | LEN_TOTAL_ID      | is_total_id                                                              |
| Ref_no1      | char | LEN_REF_NO        | Is_ref_no1                                                               |
| Ref_no2      | char | LEN_REF_NO        | Is_ref_no2                                                               |
| Ref_no3      | char | LEN_REF_NO        | Is_ref_no3                                                               |
| total_sign   | char | LEN_SALES_SIGN    | SIGN_N or SIGN_P depending on whether or not is_total_value is negative. |
| total_amount | char | LEN_AMT           | Absolute value of is_total_value.                                        |
|              |      |                   |                                                                          |

Call **putrec** to write the record out to the RDWT file.

**WrRDWCSTDctl()**

```

char *is_cashier_id
char *is_register_id
char *is_status
char *is_total_id
char *is_ref_no1
char *is_ref_no2
char *is_ref_no3
char *is_total_value

```

Increment pl\_rdwc\_line\_ctr.

Writes an RDWC\_FDETL record (as defined in salib.h) to the RDWC output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field    | Type | Size              | Source             |
|----------|------|-------------------|--------------------|
| frecdesc | char | RDW_FRECDISC_SIZE | RDW_FDETL_FRECDISC |



|              |      |                  |                                                                          |
|--------------|------|------------------|--------------------------------------------------------------------------|
| flineid      | char | LEN_FILE_LINE_NO | pl_rdwc_line_ctr                                                         |
| tran_date    | char | LEN_DATEONLY     | ps_business_date                                                         |
| location     | char | LEN_LOC          | ps_store                                                                 |
| cashier_id   | char | LEN_EMP_ID       | is_cashier_id                                                            |
| register_id  | char | LEN_REGISTER     | is_register_id                                                           |
| sales_sign   | char | LEN_SALES_SIGN   | is_status                                                                |
| total_id     | char | LEN_TOTAL_ID     | is_total_id                                                              |
| Ref_no1      | char | LEN_REF_NO       | Is_ref_no1                                                               |
| Ref_no2      | char | LEN_REF_NO       | Is_ref_no1                                                               |
| Ref_no3      | char | LEN_REF_NO       | Is_ref_no1                                                               |
| total_sign   | char | LEN_SALES_SIGN   | SIGN_N or SIGN_P depending on whether or not is_total_value is negative. |
| total_amount | char | LEN_AMT          | Absolute value of is_total_value.                                        |

Call **putrec** to write the record out to the RDWC file.

#### WrRDWFDetl()

**pt\_sa\_tran\_head \*ir\_head**  
**pt\_sa\_tran\_tender \*ir\_tend**

Increment pl\_rdwf\_line\_ctr.

Writes an RDWF\_FDETL record (as defined in salib.h) to the RDWF output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field              | Type | Size               | Source                        |
|--------------------|------|--------------------|-------------------------------|
| frecdesc           | char | RDW_FRECDDESC_SIZE | RDW_FDETL_FRECDDESC           |
| flineid            | char | LEN_FILE_LINE_NO   | pl_rdwf_line_ctr              |
| business_date      | char | LEN_DATEONLY       | ps_business_date              |
| tran_datetime      | char | LEN_DATETIME       | ir_head->s_tran_datetime      |
| location           | char | LEN_LOC            | ps_store                      |
| cashier_id         | char | LEN_EMP_ID         | ir_head->s_cashier            |
| register_id        | char | LEN_REGISTER       | ir_head->s_register           |
| tran_sign          | char | LEN_SALES_SIGN     | ir_head->s_tran_sign          |
| tran_seq_no        | char | LEN_BIG_SEQ_NO     | ir_head->s_tran_seq_no        |
| rev_no             | char | LEN_SA_REV_NO      | ir_head->s_rev_no             |
| tran_type          | char | TRAT_SIZE          | ir_head->s_tran_type          |
| tender_type_group  | char | TENT_SIZE          | ir_tend->s_tender_type_group  |
| tender_type_id     | char | TENS_SIZE          | ir_tend->s_tender_type_id     |
| tender_amt         | char | LEN_AMT            | ir_tend->s_tender_amt         |
| cc_no              | char | LEN_CC_NO          | ir_tend->s_cc_no              |
| cc_exp_date        | char | LEN_DATEONLY       | ir_tend->s_cc_exp_date        |
| cc_auth_no         | char | LEN_CC_AUTH_NO     | ir_tend->s_cc_auth_no         |
| cc_auth_src        | char | CCAS_SIZE          | ir_tend->s_cc_auth_src        |
| cc_entry_mode      | char | CCEM_SIZE          | ir_tend->s_cc_entry_mode      |
| cc_cardholder_verf | char | CCVF_SIZE          | ir_tend->s_cc_cardholder_verf |
| cc_terminal_id     | char | LEN_TERM_ID        | ir_tend->s_cc_terminal_id     |
| cc_special_cond    | char | CCSC_SIZE          | ir_tend->s_cc_special_cond    |
| voucher_no         | char | LEN_VOUCHER_NO     | ir_tend->s_voucher_no         |

|               |      |                   |                          |
|---------------|------|-------------------|--------------------------|
| Voucher_age   | Char | LEN_VOUCHER_AGE   | Ir_tend->s_voucher_age   |
| Escheat_date  | Char | LEN_DATEONLY      | Ir_tend->s_escheat_date  |
| Coupon_no     | Char | LEN_COUPON_NO     | ir_tend->s_coupon_no     |
| Coupon_ref_no | char | LEN_COUPON_REF_NO | ir_tend->s_coupon_ref_no |

Call **putrec** to write the record out to the RDWF file.

### Stored Procedures / Shared Modules (Maintainability)

| Shared Module                         | Module Description                                                                |
|---------------------------------------|-----------------------------------------------------------------------------------|
| libretex.a functions                  | Refer to Library Design – retex.doc for details.                                  |
| retex_init                            | Initialize restart recovery.                                                      |
| retex_close                           | Close restart recovery functions.                                                 |
| Retex_refresh_thread                  | Refresh the current thread so that it may be used again.                          |
| Libresa.a functions:                  | Refer to Library Design – ReSA.doc for details.                                   |
| get_lock                              | used to establish a read lock on a store/day.                                     |
| release_lock                          | used to release a store/day lock.                                                 |
| fetchSaSystemOptions                  | Fetch the values from the sa_system_options table.                                |
| fetchSysDate                          | Fetch the current SYSDATE value.                                                  |
| fetchStoreDayErrorCount               | Fetch the number of errors that corresponds to a particular store/day and system. |
| markStoreDayExported                  | Mark a particular store/day and system as exported                                |
| markTransactionExported               | Mark a particular transaction and system as exported.                             |
| OraNum functions (Add, Sub, Mul, Div) | Used to perform arithmetic operations on strings containing large numbers.        |
| getBalTotal                           | Get the specified balance totals.                                                 |
| putrec                                | Writes a record to a file.                                                        |

### Input Specifications

### Output Specifications

### Output Files

Data is output in the RDW file format. This is described in the file Interface File – SA to RDW.doc.

The filename convention for these valid RDWT, SIF Tender, RDWS and RDWC files will be `rdwt_store_businessdate_curdatetime`, `rdwf_store_businessdate_curdatetime`, `rdws_store_businessdate_curdatetime` and `rdwt_store_businessdate_curdatetime`. The files should start out with a temporary name generated by the Unix `tempnam (3S)` call and then be renamed with Unix `rename (2)` call when the files are complete.

### Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 3.0 is a 24/7 system.

Scheduling Diagram: This program will be run after auditors have made corrections to the data.

Pre-Processing: `sagetref.pc` to get waste data, and `saimptlog.pc` and `saimptlogfin.pc` to get post-void data.

Post-Processing:

- `stslmat.pc` (Sales Transaction SKU-Loc-Day-Minute ATomic) should be run to import data from the RDWT file into the RDW system.
- `ttldmat.pc` (Transaction Tender Loc-Day-Minute ATomic) should be run to import data from the RDWF file into the RDW system.
- `lptotclat.pc` (Loss Prevention Totals Cashier-Loc-Day ATomic) should be run to import data from the RDWC file into the RDW system.
- `lptotldat.pc` (Loss Prevention Totals Loc-Day ATomic) should be run to import data from the RDWS file into the RDW system.

Threading Scheme: N/A

### Locking Strategy

In conjunction with the Performance and the Scheduling Considerations section, this section should describe the locking (and release) strategy required beyond the preset Retek standards. It should describe how the module accesses data and the 'hold' or 'lock' it has on a database and / or its records, during processing. It should also describe the 'lock' release.

### Restart / Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of `pl_commit_max_ctr`. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed. The RDWT, RDWF, RDWS and RDWC formatted output files will be created with temporary names and renamed just before the end of store/day commit.

In case of failure, we rollback all work done to the point right after the call to **get\_lock** and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into `sa_exported` for one store\_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to **get\_lock**.

There are 3 driving cursors in this module. The first picks a store/day to work on:

```
c_store_day CURSOR FOR
SELECT /*+ rule */
 sd.store_day_seq_no,
 el.seq_no,
 sd.store,
 TO_CHAR(sd.business_date, 'YYYYMMDD'),
 ROWIDTOCHAR(el.rowid)
FROM sa_store_day sd, sa_export_log el
WHERE sd.store_day_seq_no = el.store_day_seq_no
AND sd.store_status = :SASS_C /* Closed
*/
```

```

AND sd.data_status = :SADS_F /* Fully loaded
*/
AND sd.audit_status = :SAAS_A /* Audited, but no Errors
*/
AND el.system_code = :SYSE_RDW
AND el.status = :SAES_R /* 'R'eady to be exported
*/
ORDER BY MOD(TRUNC(sd.store_day_seq_no / :pi_num_threads)
 + :pi_thread_val, :pi_num_threads),
 sd.store, sd.business_date;

```

Since RDW cannot accept data from a store\_day with errors pending, we select store\_days that have audit\_status 'A' only. The library function **fetchStoreDayToBeExported** cannot be used here because it fetches store\_days with an audit\_status of 'E' (Errors pending).

The second driving cursor fetches the store/day transaction data to be output:

```

SELECT h.tran_seq_no,
 h.rev_no,
 TO_CHAR(h.tran_datetime, 'YYYYMMDDHH24MISS'),
 NVL(h.register, ' '),
 NVL(TO_CHAR(h.tran_no), ' '),
 NVL(em.emp_id, ' '),
 NVL(em2.emp_id, ' '),
 NVL(c.cust_id_type, ' '),
 NVL(c.cust_id, ' '),
 NVL(h.reason_code, ' '),
 h.tran_type,
 NVL(h.sub_tran_type, ' '),
 NVL(TO_CHAR(h.orig_tran_no), ' '),
 NVL(h.orig_reg_no, ' '),
 NVL(h.ref_no1, ' '),
 NVL(h.ref_no2, ' '),
 NVL(h.ref_no3, ' '),
 NVL(h.ref_no4, ' '),
 NVL(h.vendor_no, ' '),
 h.status,
 DECODE(SIGN(h.value), -1, :SIGN_N, :SIGN_P),
 NVL(TO_CHAR(ABS(h.value) * :pl_multiplier), '0'),
 :SAFD_P,
 ' ',
FROM sa_tran_head h,
 sa_customer c,
 sa_store_emp em,
 sa_store_emp em2,
/* This temporary view selects all cashiers for the given
store */
 (SELECT DISTINCT th.cashier,
 sd.store
 FROM sa_tran_head th,
 sa_store_day sd
 WHERE sd.store_day_seq_no = th.store_day_seq_no
 AND sd.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no))
temp_view1,
/* This temporary view selects all salespersons for the given
store */
 (SELECT DISTINCT th.salesperson,
 sd.store
 FROM sa_tran_head th,
 sa_store_day sd
 WHERE sd.store_day_seq_no = th.store_day_seq_no

```

```

 AND sd.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no))
temp_view2
WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
 AND em.pos_id(+) = temp_view1.cashier
 AND em.store(+) = temp_view1.store
 AND (temp_view1.cashier = h.cashier
 OR (temp_view1.cashier IS NULL
 AND h.cashier IS NULL))
 AND em2.pos_id(+) = temp_view2.salesperson
 AND em2.store(+) = temp_view2.store
 AND (temp_view2.salesperson = h.salesperson
 OR (temp_view2.salesperson IS NULL
 AND h.salesperson IS NULL))
 AND h.tran_seq_no = c.tran_seq_no(+)
 AND h.tran_type IN (:TRAT_SALE, :TRAT_RETURN, :TRAT_EEXCH,
 :TRAT_PAIDIN, :TRAT_PAIDOU, :TRAT_NOSALE,
 :TRAT_VOID, :TRAT_PVOID, :TRAT_COND)
 AND (h.status = :SAST_P
 AND NOT EXISTS
/* and no errors for the
transaction. */
 (SELECT er.tran_seq_no
 FROM sa_error er, sa_error_impact ei
 WHERE h.tran_seq_no = er.tran_seq_no
 AND er.error_code = ei.error_code
 AND ei.system_code = :SYSE_RDW
 AND er.hq_override_ind != :YSNO_Y))
 AND NOT EXISTS
 (SELECT e.store_day_seq_no
 FROM sa_exported e
 WHERE h.store_day_seq_no = e.store_day_seq_no
 AND h.tran_seq_no = e.tran_seq_no
 AND e.system_code = :SYSE_RDW)
UNION ALL
SELECT h.tran_seq_no,
 h.rev_no,
 TO_CHAR(h.tran_datetime, 'YYYYMMDDHH24MISS'),
 NVL(h.register, ' '),
 NVL(TO_CHAR(h.tran_no), ' '),
 NVL(em.emp_id, ' '),
 NVL(em2.emp_id, ' '),
 NVL(c.cust_id_type, ' '),
 NVL(c.cust_id, ' '),
 NVL(h.reason_code, ' '),
 h.tran_type,
 NVL(h.sub_tran_type, ' '),
 NVL(TO_CHAR(h.orig_tran_no), ' '),
 NVL(h.orig_reg_no, ' '),
 NVL(h.ref_no1, ' '),
 NVL(h.ref_no2, ' '),
 NVL(h.ref_no3, ' '),
 NVL(h.ref_no4, ' '),
 NVL(h.vendor_no, ' '),
 h.status,
 DECODE(SIGN(h.value), -1, :SIGN_N, :SIGN_P),
 NVL(TO_CHAR(ABS(h.value) * :pl_multiplier), '0'),
 :SAFD_N,
 NVL(TO_CHAR(e.exp_datetime, 'YYYYMMDDHH24MISS'), ' ')
FROM sa_tran_head h,
 sa_exported e,
 sa_customer c,
 sa_store_emp em,
 sa_store_emp em2,
/* This temporary view selects all cashiers for the given
store */

```

```

 (SELECT DISTINCT th.cashier,
 sd.store
 FROM sa_tran_head th,
 sa_store_day sd
 WHERE sd.store_day_seq_no = th.store_day_seq_no
 AND sd.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no))
temp_view1,
/* This temporary view selects all salespersons for the given
store */
 (SELECT DISTINCT th.salesperson,
 sd.store
 FROM sa_tran_head th,
 sa_store_day sd
 WHERE sd.store_day_seq_no = th.store_day_seq_no
 AND sd.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no))
temp_view2
WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
AND em.pos_id(+) = temp_view1.cashier
AND em.store(+) = temp_view1.store
AND (temp_view1.cashier = h.cashier
 OR (temp_view1.cashier IS NULL
 AND h.cashier IS NULL))
AND em2.pos_id(+) = temp_view2.salesperson
AND em2.store(+) = temp_view2.store
AND (temp_view2.salesperson = h.salesperson
 OR (temp_view2.salesperson IS NULL
 AND h.salesperson IS NULL))
AND h.tran_seq_no = c.tran_seq_no(+)
AND h.tran_type IN (:TRAT_SALE, :TRAT_RETURN, :TRAT_EEXCH,
 :TRAT_PAIDIN, :TRAT_PAIDOU, :TRAT_NOSALE,
 :TRAT_VOID, :TRAT_PVOID, :TRAT_COND)
AND h.status in (:SAST_V, :SAST_D)
AND h.tran_seq_no = e.tran_seq_no(+)
AND e.status = :SAST_P
AND e.system_code = :SYSE_RDW
ORDER BY 3;

```

The third driving cursor is encapsulated in the **getBalTotal** function, which fetches all totals with a usage\_type of 'RDW'. It returns, among other things, the total\_id, the cashier id and the register id. These are then used to determine whether to write a record to the RDWS file or the RDWC file. Only totals with a total\_id of "OVRSH\_T\_B" (over/short balance level) are exported to the RDWC file. The other totals are exported to the RDWS file only if both their register and their cashier ids are empty, i.e. the total is at the store level. If the total cannot be written to neither the RDWC nor the RDWS file, then we write an error to the log and continue.

## Performance

In conjunction with the Scheduling Considerations and Locking Strategy sections, the optimization considerations of a batch module must adhere to Retek standards. This section should call out special performance considerations that may exceed current documented Retek practices. Such considerations should be the basis for update to Retek standards. Each database operation should be optimized based on quantity and quality of the database transactions. Batch modules are executed on the database or dedicated batch server and thus there are no additional performance gains to forcing database interaction logic onto the server.

## Security Considerations

Credit card numbers and other customer information are present in the output files. Access to these files is controlled only by the Unix permissions that these files have.

**Design Assumptions**

Design assumptions are presumed design factors, inferred from current information, expected to hold true over the life of the project. Design assumptions must be documented in order to justify and validate derived design considerations with the Business Requirements (documented within the BRD and PFO).

## Sales audit export to RMS [saexprms]

### Purpose

The Batch Detailed Design is a thorough definition of a single batch program / module within one functional area. The documented information is derived from this functional area's Technical Design.

### Objectives

This Batch Detailed Design must:

- Document specific functions for a single batch program,
- Enable project team review, validation and consensus regarding the individual batch program's scope,
- Document the batch program in preparation for and in response to prototyping, and
- Prepare for and provide a defined and documented framework in which to perform Development Phase activities.

### Functional Area

### Module Affected

### Design Overview

The purpose of this batch module is to fetch all corrected sale and return transactions that do not have RMS errors from the Retek Sales Audit (ReSA) database tables for transmission to the Retek Merchandising System (RMS). If sa\_system\_options.unit\_of\_work is 'S', then the whole store/day is skipped if any RMS error is found. If this value is 'T', then only transactions with RMS errors are skipped. If the transaction has a status of Deleted and it has previously been transmitted, a reversal of the transaction will be sent.

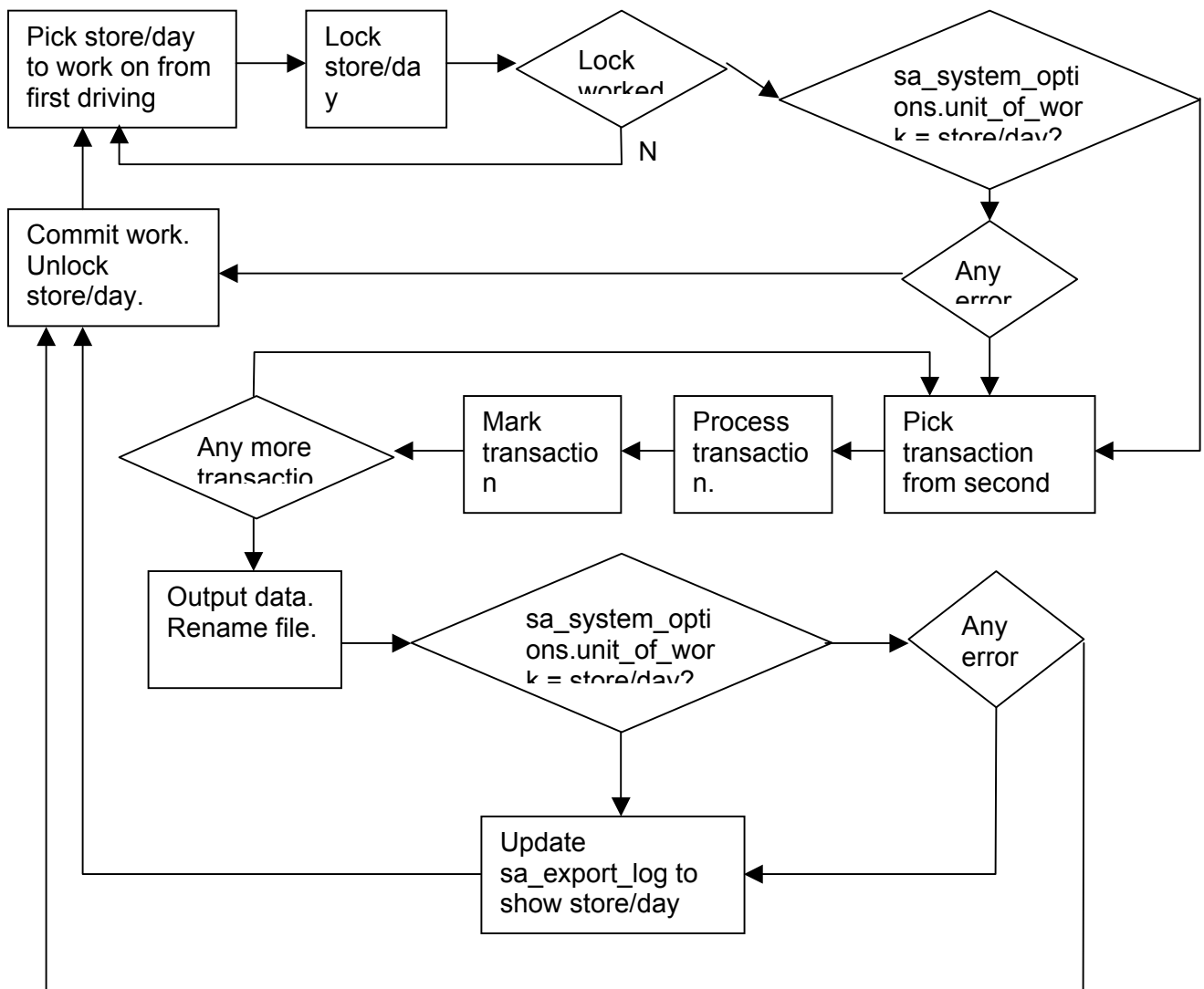
If the transaction has a status of 'D'eled and it has previously been transmitted, a reversal of the transaction will be sent.

A file of type POSU is generated for each store/day.

| Table             | Operations Performed |        |        |        |
|-------------------|----------------------|--------|--------|--------|
|                   | Select               | Insert | Update | Delete |
| sa_system_options | Yes                  | No     | No     | No     |
| sa_store_day      | Yes                  | No     | No     | No     |
| sa_export_log     | Yes                  | No     | No     | No     |
| sa_error          | Yes                  | No     | No     | No     |
| sa_tran_head      | Yes                  | No     | No     | No     |
| sa_tran_item      | Yes                  | No     | No     | No     |
| sa_tran_disc      | Yes                  | No     | No     | No     |
| sa_tran_head_rev  | Yes                  | No     | No     | No     |
| sa_tran_item_rev  | Yes                  | No     | No     | No     |



|                  |     |    |    |    |
|------------------|-----|----|----|----|
| sa_tran_disc_rev | Yes | No | No | No |
| store            | Yes | No | No | No |
| currencies       | Yes | No | No | No |
| sa_exported      | Yes | No | No | No |
| sa_exported_rev  | Yes | No | no | No |

**Program Flow****Global Variable Descriptions**

| Gobal Variable  | Description                                      |
|-----------------|--------------------------------------------------|
| Ps unit_of work | Unit of Work from sa_system_options.             |
| Ps sysdate      | Current sysdate value from the database.         |
| Ps temp_file    | Temporary file name to be used for the RMS file. |

|                           |                                                                       |
|---------------------------|-----------------------------------------------------------------------|
| Ps_file_name              | Final file name of the RMS output file.                               |
|                           |                                                                       |
|                           |                                                                       |
|                           |                                                                       |
|                           |                                                                       |
|                           |                                                                       |
|                           |                                                                       |
|                           |                                                                       |
| Pi_tdetl_count            | TDETL record count for TTAIL record in the RMS file.                  |
| Current_item              | Pointer to the current item node of the binary tree.                  |
| Itemroot                  | Root of the binary tree.                                              |
|                           |                                                                       |
| FileRecOutCount           | Record count out for FTAIL.                                           |
|                           |                                                                       |
| RMSoutFile                | RMS output file pointer.                                              |
| Ps_store                  | Current store/location ID.                                            |
| Ps_business_date          | Business date of store/day combination                                |
| Ps_vat_region             | VAT Region for the current store/day.                                 |
| Ps_vat_include_ind        | VAT Include Indicator for the current store/day.                      |
| Ps_currency_code          | Currency Code for the current store/day.                              |
| Ps_currency_rtl_dec       | Currency Retail Decimal places for the current store/day.             |
| Pi_num_locks_not_released | Counter for the number of store/day locks that could not be released. |
|                           |                                                                       |

## Function Level Description

## main()

```
int argc
char *argv[]
```

Check command line for required arguments.  
Call **LOGON** to connect to the database.  
Call **Init** to initialize the program.  
Call **process** to export the available RMS data.  
Report unlocking errors.  
Call **final** to cleanup.

**init()**

### No arguments

This function initializes Restart recovery.  
Initialize OraNum functions by calling **OraNumInit**.  
Get the value of sa\_system\_options.unit\_of\_work by calling the library function **fetchSaSystemOptions**.  
Get a temporary filename to use for generating the output files. Store this name in ps\_temp\_file.

**process()****No arguments**

Picks a store/day to be processed by using the first driving cursor.

Attempt to lock the store/day with a call to **get\_lock**. If this fails, go on to the next store/day.

If `sa_system_options.unit_of_work` is store/day, then check to see if any of the store/days transactions have RMS errors by calling the library function **fetchStoreDayErrorCount**. If they do, unlock the store/day with a call to **release\_lock** and go on to the next store/day.

Get VAT information by calling **fetchVatCur**.

Call **fetchSysDate** to get the current date/time.

Save the store/location ID in `ps_store`.

Initialize `itemroot` and `current_item` to NULL.

Call **processStoreDay** to process the store/days transactions.

Call **WrOutputData** to format and write the RMS output records.

Update the status in `sa_export_log` to 'E'xported by calling the library function

**markStoreDayExported**

Rename temporary output file to `posu_store_businessdate_curdatetime`.

Call to **release\_lock** and go on to the next store/day. This function commits as a side effect, thus committing the changes to the database.

**final()****No arguments**

Remove the temporary file.

Call **retek\_close**.

**FetchVatCur()**

**Char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**processStoreDay()**

**char \*is\_store\_day\_seq\_no**

**char \*is\_sysdate**

For each transaction from the store/day being processed, get the following information from the second driving cursor and call **processTransHead** with the information.

| Table        | Column           | Description                                                                                                                                                                                                                            |
|--------------|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sa_tran_head | Tran_seq_no      |                                                                                                                                                                                                                                        |
| Sa_tran_head | Rev_no           |                                                                                                                                                                                                                                        |
| Sa_tran_head | Tran_datetime    | Format YYYYMMDDHH24MISS                                                                                                                                                                                                                |
| Sa_tran_head | Status           |                                                                                                                                                                                                                                        |
|              | Transaction Sign | SAFD_P if the transaction has not been deleted (status != SAST_D) and there are no errors and it has not been exported.<br>SAFD_N if the transaction has been deleted (status = SAST_D) and it has been exported after being exported. |
| Sa_exported  | Exp_datetime     | Only for transactions with a Transaction Sign of SAFD_N.<br>Format YYYYMMDDHH24MISS                                                                                                                                                    |

Calls the library function **markTransactionExported** to insert a record into `sa_exported`.

**processTransHead()**

**struct pt\_sa\_tran\_head \*ir\_sa\_tran\_head**

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

If the transaction status is deleted (SAST\_D) and it has been previously exported, then call

**retrieveTransHeadRev**. Also, if the revision number of the transaction is not 1, then a previous

revision may have been exported; call **retrieveTransHeadRev** to get the exported revision (for full disclosure purposes).

Call **retrieveTransItem** and **retrieveTransDisc** to obtain the items and discounts for the transaction, both Positive transactions and Negative ones.

Call **saveData** for both the Positive and Negative transactions to write the information into memory.

#### **retrieveTransHeadRev()**

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**struct pt\_sa\_tran\_head \*or\_sa\_tran\_head\_rev**

This function gets the sa\_tran\_head\_rev record that needs to be processed. A record needs to be processed if it has been previously exported.

| Table            | Column           | Description                                                                         |
|------------------|------------------|-------------------------------------------------------------------------------------|
| Sa_tran_head_rev | Tran_seq_no      |                                                                                     |
| Sa_tran_head_rev | Rev_no           |                                                                                     |
| Sa_tran_head_rev | Tran_datetime    | Format YYYYMMDDHH24MISS                                                             |
| Sa_tran_head_rev | Status           |                                                                                     |
|                  | Transaction Sign | SAFD_N                                                                              |
| Sa_exported_rev  | Exp_datetime     | Only for transactions with a Transaction Sign of SAFD_N.<br>Format YYYYMMDDHH24MISS |

If no data is found, than set or\_sa\_tran\_head\_rev->s\_rev\_no to -1.

#### **retrieveTransItem()**

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**char is\_rev\_no[NULL\_SA\_REV\_NO]**

**long \*ol\_num\_sa\_tran\_item**

**struct pt\_sa\_tran\_item \*\*or\_sa\_tran\_item**

This function gets all sa\_tran\_item records or sa\_tran\_item\_rev (if is\_rev\_no is not -1) that need to be processed for a tran\_seq\_no.

Records should be limited to those with an item\_type of SAIT\_ITEM and SAIT\_REF.

| Table        | Column      | Description                                    |
|--------------|-------------|------------------------------------------------|
| Sa_tran_item | Tran_seq_no |                                                |
| Sa_tran_item | Item_seq_no |                                                |
| Sa_tran_item | Item_status |                                                |
| Sa_tran_item | Item_type   |                                                |
| Sa_tran_item | Item        |                                                |
| Sa_tran_item | Ref_item    |                                                |
| Sa_tran_item | Dept        |                                                |
| Sa_tran_item | Class       |                                                |
| Sa_tran_item | Subclass    |                                                |
| Sa_tran_item | Waste_type  |                                                |
| Sa_tran_item | Waste_pct   | Value multiplied by 10000.                     |
| Sa_tran_item | Qty         | SIGN_N or SIGN_P depending on the sign of qty. |
| Sa_tran_item | Qty         | Absolute value multiplied by 10000.            |
| Sa_tran_item | Unit_retail | Absolute value multiplied by 10000.            |
| Sa_tran_item | Selling_uom |                                                |

|              |               |  |
|--------------|---------------|--|
| Sa_tran_item | Drop_ship_ind |  |
|--------------|---------------|--|

The same columns as above are selected from the sa\_tran\_item\_rev table if the rev\_no passed in is not -1.

Set \*ol\_num\_sa\_tran\_item to the total number of records fetched.

#### retrieveTransDisc()

char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]

char is\_rev\_no[NULL\_SA\_REV\_NO]

long \*ol\_num\_sa\_tran\_disc

struct pt\_sa\_tran\_disc \*\*or\_sa\_tran\_disc

This function gets all sa\_tran\_disc or sa\_tran\_disc\_rev records (if is\_rev\_no is not -1) for a tran\_seq\_no that needs to be processed.

Records should be limited to those with an item\_type of SAIT\_ITEM and SAIT\_REF.

| Table        | Column            | Description                                    |
|--------------|-------------------|------------------------------------------------|
| Sa_tran_disc | Tran_seq_no       |                                                |
| Sa_tran_disc | Item_seq_no       |                                                |
| Sa_tran_disc | rms_promo_type    |                                                |
| Sa_tran_disc | Promotion         |                                                |
| Sa_tran_disc | Discount_type     |                                                |
| Sa_tran_disc | Qty               | SIGN_N or SIGN_P depending on the sign of qty. |
| Sa_tran_disc | Qty               | Absolute value multiplied by 10000.            |
| Sa_tran_disc | Unit_discount_amt | Value multiplied by 10000.                     |
|              |                   |                                                |

The same columns as above are selected from the sa\_tran\_disc\_rev table if the rev\_no passed in is not -1.

Set \*ol\_num\_sa\_tran\_disc to the total number of records fetched.

#### saveData()

struct pt\_sa\_tran\_head ir\_sa\_tran\_head

long il\_num\_sa\_tran\_item

struct pt\_sa\_tran\_item \*ia\_sa\_tran\_item

long il\_num\_sa\_tran\_disc

struct pt\_sa\_tran\_disc \*ia\_sa\_tran\_disc

For each item record:

For each item's discount record:

\

#### AddItem()

Struct pt\_sa\_tran\_head \*ir\_sa\_tran\_head

Struct pt\_sa\_tran\_item \*ir\_sa\_tran\_item

Finds an item or adds a new item to the tree. Returns pointer if OK, or NULL if failure.

If item\_status is SASI\_V, than it needs to be reset depending on the sign of qty. Either SASI\_R if it is negative or SASI\_S if it is positive.

**Nextitem** is called to do the rollups that happen when a subsequent item of the same type is encountered.

If the item is not found, than **newitem** is called to create it and add it to the tree.

**Nextitem()**

```

Struct pt_sa_tran_head *ir_sa_tran_head
Struct pt_sa_tran_item *ir_sa_tran_item
Struct ITEM_TAG *item_tag_ptr

```

**Newitem()**

```

char *pricepoint
int hv
Struct pt_sa_tran_head *ir_sa_tran_head
Struct pt_sa_tran_item *ir_sa_tran_item

```

**AddDiscData()**

```

Struct pt_sa_tran_head *ir_sa_tran_head
Struct pt_sa_tran_disc *ir_sa_tran_disc
char is_item_status[NULL_CODE]
Struct ITEM_TAG *i_item

```

Adds discount data into the existing item.

Calls **finddiscdata** to find a discount or adds a new discount to the item. Returns pointer to the data if OK, or NULL if failure. **Nextdiscdat** is called to do the rollups that happen when a subsequent discount of the same type is encountered.

If the discount is not found, than **newdiscdat** is called to create it and add it to the item.

**Finddiscdata()**

```

struct DISCDAT_TAG *discdat_tag_ptr
Char *is_type
Char *is_amt

```

Searches the ITEM for the matching discount type.

**Nextdiscdat()**

```

Struct ITEM_TAG *item_tag_ptr
struct DISCDAT_TAG *discdat_tag_ptr
char is_item_status[NULL_CODE]
Struct pt_sa_tran_disc *ir_sa_tran_disc

```

**Newdiscdat()**

```

Struct ITEM_TAG *item_tag_ptr
char is_item_status[NULL_CODE]
Struct pt_sa_tran_disc *ir_sa_tran_disc

```

**WrOutputData()**

No arguments

Open the RMS output temporary file (ps\_temp\_file).

Write a RMS FHEAD record by calling **WrRMSFHead**.

Write the RMS transaction records by calling **wod**.

Write a RMS FTAIL record by calling **WrRMSFTail**.

Close the RMS temporary output file.

**Wod**

```

Struct ITEM_TAG *item_tag_ptr

```

Calls itself recursively to output data from the entire binary tree.

If item\_tag\_ptr is not NULL

Call **wod** to recurse down the left branch.

If the total negative sales quantity is not zero than process the reverse sale.

Call **WrRMSTHead** to write the THEAD record.

For each of the discount records attached to this item, call

**WrRMSTDetl** to write TDETL.

Call **WrRMSTTail** to write the reversed sales TTAIL record.

If the total positive sales quantity is not zero than process the positive sale.

Call **WrRMSTHead** to write the THEAD record.

For each of the discount records attached to this item, call

**WrRMSTDetl** to write TDETL.

Call **WrRMSTTail** to write the reversed sales TTAIL record.

If the total negative return quantity is not zero than process the reverse return.

Call **WrRMSTHead** to write the THEAD record.

For each of the discount records attached to this item, call

**WrRMSTDetl** to write TDETL.

Call **WrRMSTTail** to write the reversed sales TTAIL record.

If the total positive return quantity is not zero than process the positive return.

Call **WrRMSTHead** to write the THEAD record.

For each of the discount records attached to this item, call

**WrRMSTDetl** to write TDETL.

Call **WrRMSTTail** to write the reversed sales TTAIL record.

Call **wod** to recurse down the right branch.

### WrRMSFHead()

**No arguments**

Writes an RMS\_FHEAD record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RRMS.doc.

Set FileRecOutCount to 1.

| Field                | Type | Size                 | Source              |
|----------------------|------|----------------------|---------------------|
| Frecdesc             | char | RDW FRECDISC SIZE    | RDW FHEAD FRECDISC  |
| Flineid              | char | LEN_FILE_LINE_NO     | FileRecOutCount     |
| file_type_definition | char | LEN_FILE_TYPE_DEF    | 'POSU'              |
| file_create_date     | char | LEN_DATETIME         | ps_sysdate          |
| Location             | Char | LEN_LOC              | ps_store            |
| Vat_include_ind      | Char | LEN_IND              | Ps_vat_include_ind  |
| Vat_region           | Char | LEN_VAT_REGION       | Ps_vat_region       |
| Currency_code        | Char | LEN_CURRENCY_CODE    | Ps_currency_code    |
| Currency_rtl_dec     | char | LEN_CURRENCY_RTL_DEC | Ps_currency_rtl_dec |

Call **putrec** to write the record out to the RMS file.

### WrRMSTHead()

**Struct ITEM\_TAG \*item\_tag\_ptr**

**const char is\_item\_status[NULL\_CODE]**

**Const char is\_tran\_sign[NULL\_CODE]**

This function writes a RMSTHead record to the output file. This function needs to copy the last\_time\_modified from the ITEM\_TAG struct into the RMS\_THEAD struct before calling putrec.

### WrRMSTDetl()

**Struct DISCDAT\_TAG \*discdat\_tag\_ptr**

Writes a RMSTDetl record to the output file.

### WrRMSTTail()

**No arguments**

Writes a RMSTTail record to the output file.

### WrRMSFTail()

**No arguments**

Writes a RMSFTail record to the output file.

### delete\_item\_tree()

**struct ITEM\_TAG \*itemroot**

This function recursively deletes all items in memory.

Calls **delete\_discounts** to delete discount information from the tree (before deleting the current node).

### delete\_discounts ()

**struct DISCDAT\_TAG \*discdat\_tag\_ptr**

This function recursively deletes all discounts for a particular item in memory.

### log\_and\_exit ()

**char \*is\_message**

This function writes is\_message to the message log, calls final() and then exits.

### Blank\_field ()

**char \*is\_field**

**int ii\_len**

This function fills the character array with spaces up to ii\_len

### createNewExportEntry ()

**char is\_store\_day\_seq\_no[NULL\_BIG\_SEQ\_NO]**

**char is\_seq\_no[NULL\_LITTLE\_SEQ\_NO]**

This function creates a new entry in the sa\_export\_log table if the store/day was partially loaded. Thus, when new data is sent, the store/day will be “ready to export” to RMS.

## Stored Procedures / Shared Modules (Maintainability)

| <i>Shared Module</i> | <i>Module Description</i>                                |
|----------------------|----------------------------------------------------------|
| libretex.a functions | Refer to Library Design – retex.doc for details.         |
| retex_init           | Initialize restart recovery.                             |
| retex_close          | Close restart recovery functions.                        |
| Retex_refresh_thread | Refresh the current thread so that it may be used again. |
| Libresa.a functions: | Refer to Library Design – ReSA.doc for details.          |



| <i>Shared Module</i>                  | <i>Module Description</i>                                                         |
|---------------------------------------|-----------------------------------------------------------------------------------|
| get_lock                              | Used to establish a read lock on a store/day.                                     |
| release_lock                          | Used to release a store/day lock.                                                 |
| fetchSaSystemOptions                  | Fetch the values from the sa_system_options table.                                |
| fetchSysDate                          | Fetch the current SYSDATE value.                                                  |
| fetchStoreDayErrorCount               | Fetch the number of errors that corresponds to a particular store/day and system. |
| markStoreDayExported                  | Mark a particular store/day and system as exported                                |
| markTransactionExported               | Mark a particular transaction and system as exported.                             |
| OraNum functions (Add, Sub, Mul, Div) | Used to perform arithmetic operations on strings containing large numbers.        |
| Putrec                                | Writes a record to a file.                                                        |

### Input Specifications

### Output Specifications

Data is output in the POSU file format. This is described in Interface File – SA to RMS.doc.

The filename convention for these valid POSU files will be `posu_store_businessdate_curdatetime`. The file should start out with a temporary name generated by the Unix `tempnam` (See Unix man page 3S) call and then be renamed with Unix `rename` (See Unix man page 2) call when the file is complete.

### Database Integrity

This information derives from the Database Considerations within the Process / Functional Overview (PFO), the Conversation Flow and Database Objects of the Technical Design.

### Parameter validation

Parameter validation focuses on validating parameter data that is being passed from calling modules.

### Integrity Constraints

Operations that affect other entities in the system must be validated to ensure that integrity constraints have not been violated. If a record cannot exist in the system without a related parent record existing first, it is essential that the application enforce this constraint. Similarly, if a record cannot be deleted due to the existence of child records in the system the application should prevent the user from performing a delete operation.

### Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 10.0 is a 24/7 system.

Scheduling Diagram: This program will be run after auditors have made corrections to the data.

Pre-Processing: `sagetref.pc` to get reference data.

Post-Processing: `posupld.pc` should be run after `saexprms.pc` to import the data into the RMS system.

Threading Scheme: N/A

## Locking Strategy

In conjunction with the Performance and the Scheduling Considerations section, this section should describe the locking (and release) strategy required beyond the preset Retek standards. It should describe how the module accesses data and the 'hold' or 'lock' it has on a database and / or its records, during processing. It should also describe the 'lock' release.

## Restart / Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of pl\_commit\_max\_ctr. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed. The POSU formatted output file will be created with a temporary name and renamed just before the end of store/day commit.

In case of failure, we rollback all work done to the point right after the call to get\_lock() and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into sa\_exported for one store\_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to get\_lock().

There are 2 driving cursors in this module. The first picks a store/day to work on:

```
SELECT sd.store_day_seq_no,
 el.seq_no,
 sd.store,
 TO_CHAR(sd.business_date, 'YYYYMMDD'),
 sd.data_status
FROM sa_store_day sd, sa_export_log el
WHERE sd.store_day_seq_no = el.store_day_seq_no
AND sd.store_status IN (:SASS_W, :SASS_C) /* Worksheet or Closed */
AND sd.data_status IN (:SADS_P, :SADS_F) /* Partially or Fully loaded */

AND el.system_code = :SYSE_RMS
AND el.status = :SAES_R /* 'R'eady to be exported */
ORDER BY sd.store, sd.business_date;
```

The second fetches the transaction data to be output:

```
SELECT h.tran_seq_no,
 LTRIM(h.rev_no, '0'),
 TO_CHAR(h.tran_datetime, 'YYYYMMDDHH24MISS'),
 h.status,
 :SAFD_P,
 ' '
FROM sa_tran_head h
WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
AND h.tran_type IN (:TRAT_SALE, :TRAT_RETURN, :TRAT_EEXCH)
AND (h.status = :SAST_P
 AND NOT EXISTS /* and no errors for the transaction. */
 (SELECT er.tran_seq_no
 FROM sa_error er, sa_error_impact ei
 WHERE h.tran_seq_no = er.tran_seq_no
 AND er.error_code = ei.error_code
 AND ei.system_code = :SYSE_RMS
 AND er.hq_override_ind != :YSNO_Y))
AND NOT EXISTS
 (SELECT e.store_day_seq_no
 FROM sa_exported e
 WHERE h.store_day_seq_no = e.store_day_seq_no)
```

```

 AND h.tran_seq_no = e.tran_seq_no
 AND e.system_code = :SYSE_RMS)
UNION
 SELECT h.tran_seq_no,
 LTRIM(h.rev_no, '0'),
 TO_CHAR(h.tran_datetime, 'YYYYMMDDHH24MISS'),
 h.status,
 :SAFD_N,
 NVL(TO_CHAR(e.exp_datetime, 'YYYYMMDDHH24MISS'), ' ')
 FROM sa_tran_head h, sa_exported e
 WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
 AND h.tran_type IN (:TRAT_SALE, :TRAT_RETURN, :TRAT_EEXCH)
 AND h.status IN (:SAST_V, :SAST_D)
 AND h.tran_seq_no = e.tran_seq_no(+)
 AND e.status != :SAST_P
 AND e.system_code = :SYSE_RMS
 ORDER BY 3;

```

### Performance

The rolled up item data is kept in a binary tree to speed up searching.

### Security Considerations

Highlight any batch security design exceptions / options.

### Design Assumptions

Design assumptions are presumed design factors, inferred from current information, expected to hold true over the life of the project. Design assumptions must be documented in order to justify and validate derived design considerations with the Business Requirements (documented within the BRD and PFO).

## Sales audit export to UAR [saexpuar.pc]

### Design Overview

This module will post specified totals to the *Driscoll UAR* application. Using the typical export process, this module will loop through all available store day combinations. For each store day, all specified totals will be posted to their appropriate output files. All driving cursors will be handled by the libresa library functions.

| Table                  | <i>Operations Performed</i> |        |        |        |
|------------------------|-----------------------------|--------|--------|--------|
|                        | Select                      | Insert | Update | Delete |
| Period                 | Yes                         | No     | No     | No     |
| Sa_store_day           | Yes                         | No     | No     | No     |
| Sa_export_log          | Yes                         | No     | Yes    | No     |
| Sa_exported            | No                          | Yes    | No     | No     |
| Sa_exported_rev        | Yes                         | No     | No     | No     |
| Sa_total_head          | Yes                         | No     | No     | No     |
| Sa_total               | Yes                         | No     | No     | No     |
| Sa_bank_store          | Yes                         | No     | No     | No     |
| Sa_store_day_read_lock | Yes                         | Yes    | No     | Yes    |
| Sa_store_value         | Yes                         | No     | No     | No     |
| Sa_sys_value           | Yes                         | No     | No     | No     |
| Sa_pos_value           | Yes                         | No     | No     | No     |
| Sa_hq_value            | Yes                         | No     | No     | No     |

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA Totaling process.

Threading Scheme: N/A

### Restart Recovery

Logical Unit of Work ( recommended Commit checkpoints )

Driving Cursor

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of pl\_commit\_max\_ctr. Only two commits will be done. One to establish the store/day lock (this will be done by the package) and one at the end after a store/day has been completely processed.

**Driving cursor 1:**

The libresa library function `fetchStoreDayToBeExportedLike` will drive the stores to be processed for any usage type starting with 'UAR'.

### Driving Cursor 2:

The libresa library function `getTotalLike` will drive the totals to be exported for any usage type starting with 'UAR'.

### Program Flow

Structure Chart

Please see the following document for the complete structure chart of the standard export for ReSA.

Functional Design – SA export.doc

### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

libresa library functions:

- `fetchStoreDayToBeExportedLike`
- `fetchSaSystemOptions`
- `fetchSysdate`
- `fetchStoredayErrorCount`
- `markStoreDayExported`
- `updateStoreDayExported`
- `markTotalExported`
- `getTotalLike`
- `get_lock`
- `OraNumInit`
- `OraNumAdd`

### Function Level Description

All database interactions required and error handling considerations

#### **init ()**

1. Call **OraNumInit** to initialize string numbers arithmetic operations.
2. Get the current system date from the library function **fetchSysdate**.
3. Get the unit-of-work by calling the library function **fetchSaSystemOptions**.

**process ()**

1. Loop through the libresa library function **fetchStoreDayToBeExportedLike**.
2. Attempt to obtain a read lock on the store/day with a call to **get\_lock**. If this fails, go on to the next store/day and log the problem to the error log.
3. Call the function **processStoreDay** for the current store day.
4. Call the function **markStoreDayExported**.
5. Call the function **retex\_force\_commit**.
6. Loop from beginning until the return result of the function **fetchStoreDayToBeExportedLike** = 1.

**final ()**

1. Call the library function **updateStoreDayExported** to write any unwritten store days to the database.
2. Close output files.
3. Clean up any memory used.
4. Call the function **retex\_close**.

**addNewOutputFile (char is\_usage\_type,  
char is\_business\_date,  
char is\_sysdate) returns integer**

This function will generate a new output file for any new usage types retrieved from the **getTotalLike** function call. It will also add a new file item to a collection of any files currently being written to.

The file collection should contain the following items:

1. Usage type – the usage type returned by **getTotalLike**.
2. File name - <usage type>\_<business date>\_<system date>
3. File pointer – Pointer to the output file.
4. Wrote header – file header written indicator (0 – no, 1 – yes)
5. File sum – ongoing sum of each transaction in file.

**getOutputFilePointer (char is\_usage\_type) returns integer**

This function will retrieve the output file pointer for the usage type passed by checking to see if the usage type exists on the output file collection.

- If the usage type exists on the file collection, the item number for the collection is returned.
- If the usage type does not exist, the function returns -1.

**writeStoreDayDetail (FILE \*if\_file\_pointer,  
char is\_total\_id,  
char is\_store,  
char is\_business\_date,  
char is\_amount,  
char is\_total\_seq\_no,  
char is\_UAR\_tran\_code) returns integer**

This function will write the current UAR total to the output file specified. Each field is separated by commas and surrounded by double quotes.

UAR Detail record:

| Field | Description           | Sales Audit value                                                                       |
|-------|-----------------------|-----------------------------------------------------------------------------------------|
| 1     | Detail flag           | 1                                                                                       |
| 2     | Serial number         | Store number                                                                            |
| 3     | Amount                | Total value                                                                             |
| 4     | Transaction date      | Transaction date                                                                        |
| 5     | Transaction code      | Mapped value: see the function <code>getAdditionalInfo</code> for detailed explanation. |
| 6     | User defined value 1  | Total sequence number                                                                   |
| 7     | User defined value 2  | Nothing                                                                                 |
| 8     | User defined value 3  | Nothing                                                                                 |
| 9     | User defined value 4  | Nothing                                                                                 |
| 10    | User defined value 5  | Nothing                                                                                 |
| 11    | User defined value 6  | Nothing                                                                                 |
| 12    | User defined value 7  | Nothing                                                                                 |
| 13    | User defined value 8  | Nothing                                                                                 |
| 14    | User defined value 9  | Nothing                                                                                 |
| 15    | User defined value 10 | Nothing                                                                                 |
| 16    | State                 | Nothing                                                                                 |
| 17    | Account               | Total identifier                                                                        |
| 18    | End of line           | \n                                                                                      |

- All 18 fields should be concatenated together.

**writeStoreDayHeader** (**FILE \*if\_file\_pointer**,  
**char is\_total\_id**,  
**char is\_business\_date**) returns integer

This function will write the header record for the current store day to the output file. Each field is separated by commas and surrounded by double quotes.

UAR Header record:

| Field | Description         | Sales Audit value |
|-------|---------------------|-------------------|
| 1     | Header flag         | 0                 |
| 2     | Account number      | Total identifier  |
| 3     | Source              | D                 |
| 4     | Transaction date    | Transaction date  |
| 5     | Organization number | Nothing           |
| 6     | Format              | UAR34             |
| 7     | End of line         | \n                |

- All 7 fields should be concatenated together.

**writeStoreDayFooter** (**FILE \*if\_file\_pointer**,  
**char is\_amount**) returns integer

This function will write the footer record for the current store day to the output file. Each field is separated by commas and surrounded by double quotes.

UAR Footer record:

| Field | Description       | Sales Audit value                    |
|-------|-------------------|--------------------------------------|
| 1     | Footer flag       | 9                                    |
| 2     | Beginning balance | +000000000000000                     |
| 3     | Ending balance    | “+”    the ongoing sum for the file. |
| 4     | Available balance | Nothing                              |
| 5     | End of line       | \n                                   |

- All 5 fields should be concatenated together.

#### **CloseOutputFiles () returns integer**

This function will loop through the output file collection and call the ‘fclose’ C function for each.

**getOutputFileName (char is\_usage\_type,  
char is\_business\_date,  
char is\_sysdate,  
char os\_filename) returns integer**

This function will generate the unique file name for the total usage type passed. The filename will have the following structure:

is\_usage\_type || “\_” || is\_business\_date || “\_” || is\_sysdate

**getAdditionalInfo (char is\_total\_seq\_no,  
char is\_ref\_no1,  
char os\_total\_id,  
char os\_UAR\_tran\_code) returns integer**

This function retrieves both the total identifier and UAR transaction code for the current total sequence number. The UAR transaction code is retrieved as follows:

- If ref\_no1 is not null
  - If ref\_no1 maps to the SA\_CONSTANTS table (ref\_no1 = SA\_CONSTANTS.CONSTANT\_ID).
    - UAR transaction code = SA\_CONSTANTS.CONSTANT\_VALUE
  - If ref\_no1 does not map to the SA\_CONSTANTS table (ref\_no1 != SA\_CONSTANTS.CONSTANT\_ID).
    - UAR transaction code = ref\_no1
- If ref\_no1 is null
  - UAR transaction code = total identifier

**processStoreDay (char is\_store\_day,  
char is\_sysdate,  
char is\_business\_date,  
char is\_store) returns integer**



This function will process an entire store days UAR totals.

1. Loop through all UAR totals by calling the function **getTotalLike** until the function returns anything but zero.
2. Call the function getAdditionalInfo.
3. Determine if the output file exists by calling the function getOutputFilePointer.
4. If the pointer does not exist, call the function addNewOutputFile to create the new file.
5. Check to see if the header detail record has already been written for the current file by checking the current item on the output file collection.
6. If the head has not been written, call the function writeStoreDayHeader. Set the header indicator to 1 in the output file collection for the current item.
7. Write the current total to the current output file by calling the function writeStoreDayDetail.
8. Add the current total value to the running total sum in the output file collection for the current item.
9. Call the function **markTotalExported**.

### I/O Specification

All files layouts input and output

The UAR output file specifications are listed in this document by the functions that write the output:

- writeStoreDayHeader
- writeStoreDayDetail
- writeStoreDayFooter

## Sales audit import adjustment [saimpadj]

### Design Overview

This module will post external system adjustments to the Sales Audit total value table. The module will be passed an input file name. The input files will be of a standard format as detailed in the “Interface File – SA Import Adj.doc” document.

| Table          | <i>Operations Performed</i> |        |        |        |
|----------------|-----------------------------|--------|--------|--------|
|                | Select                      | Insert | Update | Delete |
| Period         | Yes                         | No     | No     | No     |
| Sa_store_day   | Yes                         | No     | Yes    | No     |
| Sa_export_log  | Yes                         | Yes    | No     | No     |
| Sa_total       | Yes                         | No     | No     | No     |
| System_options | Yes                         | No     | No     | No     |
| Sa_total_usage | Yes                         | No     | No     | No     |
| Sa_hq_value    | No                          | Yes    | No     | No     |

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be executed after the ReSA transaction import process, and before the ReSA totaling process.

Threading Scheme: N/A

### Restart Recovery

Logical Unit of Work ( recommended Commit checkpoints )

Driving Cursor

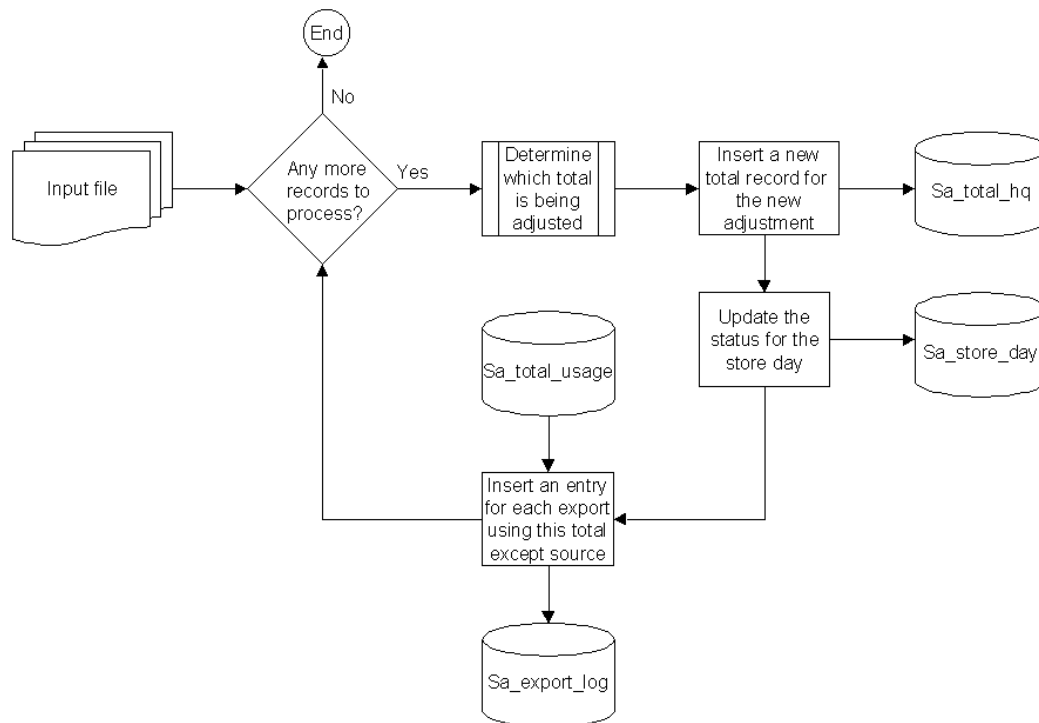
The logical unit of work for this module will be a parameterized number of records from the input file. Records will be processed until an internal counter has been reached and a commit will be executed. The Retek library (rettek\_2.h) function *rettek\_commit* will be called after each record processed. This function will keep track of the internal counter and commit the work once it has been reached.

### Driving cursor:

A standard *while* loop will read through all records in the input file. All file processing will use the new Retek library (rettek\_2.h) functions: *rettek\_init*, *rettek\_get\_record*, *rettek\_write\_rej*, *rettek\_commit*, *rettek\_close*.

## Program Flow

### Structure Chart



The module Determine which total is being adjusted (as shown above) will depend on the source of the record being processed. If the record is from the UAR application the **uar\_source** field will determine which total is being used. If the record is from the General ledger application the **gl\_account** field will help determine which total is being adjusted.

The current status of the program passes through the “determine which total is being adjusted” section and just gives the program the total via the input file. The UAR and GL application information is still in question and will be modified when the best procedure is known.

## Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

## Function Level Description

All database interactions required and error handling considerations

### Init ()

- Read in the header from the input file.

### Process ()

1. Loop through the input file adjustment records by calling the Retek library function **retek\_get\_record**.
  - a) Fetch the store\_day\_seq\_no from the sa\_total table.

b) Get a lock for the record by calling **get\_Lock**.  
 i. Call the function **processAdjustment**. If **processAdjustment** returns a value  $\neq 0$  then call the Retek library function **rettek\_write\_rej** in order to document unidentifiable adjustment records.

Call the Retek library function **rettek\_commit** in order to commit on an internally defined number of records processed.

c) If lock could not be obtained, write an error to the error log.

2.

### **Final ()**

- Call **rettek\_close**
- Clean up any memory used.

### **processAdjustment (Char il\_store\_day\_seq\_no)**

This function will complete one total adjustment by using the following processes:

1. Call the function **determineTotal** in order to identify the total identifier for the incoming adjustment.
2. Call the function **postAdjustmentTotal** in order to move the new adjustment amount into the current total table.
3. Call the function **updateStoreDayStatus** in order to ensure that the store day is re-calculated by the totaling process.
4. Call the function **postExportLog** in order to insert new records for all export requests except the source application of the adjustment.
5. Return 0 if successful adjustment posting.
6. Return 1 if unable to identify total.
7. Return -1 if a database error occurs.

### **determineTotal (o\_totalIdentifier \*char) as Long**

(if necessary – still waiting on) This function will retrieve the total identifier for the current adjustment. This function will be the only location that will be checking specific source application data; any changes to the way applications handle exporting total adjustments will only have to be modified in this function. The following procedures will occur:

1. If the source of the current adjustment record is the General Ledger.
  - a) Determine the total identifier by tracing the store, date and GL account number back to the GL mapping table (depending on which financial application is being used).
2. If the source of the current adjustment record is the UAR.
  - a) Determine the total identifier by tracing the store, date and UAR source type back to the total definition table.
3. If the source of the current adjustment record is not UAR or General Ledger.
  - a) Return the *Total* field in the **o\_totalIdentifier** argument.
4. Return 0 if total identifier is found.
5. Return 1 if total identifier is not found.
6. Return -1 if a database error occurs.

### **postAdjustmentTotal () as Long**

This function will insert the new total value record with the new data from the adjustment record. The following procedures will occur:

1. Fetch the max value\_rev\_no from the four value tables.
2. Find the max of the values returned
3. Insert the new total value into sa\_hq\_value table.
4. Return 0 if successful.
5. Return 1 if the total was not found in the total value table.
6. Return -1 if a database error occurred.

#### **updateStoreDayStatus () as Long**

This function will update the status field in the sa\_store\_day table. The following procedures will occur:

1. Update the sa\_store\_day table setting the audit status field to 'Re-Totaling required' ('R') and the audit changed date to the sysdate.
2. Return 0 if successful.
3. Return 1 if the store day record was not found in the sa\_store\_day table.
4. Return -1 if a database error occurred.

#### **postExportLog () as Long**

This function will make an entry in the export log (sa\_export\_log) for every application that requires export data for the current total being adjusted except the source application of the adjustment. The following procedures will occur:

1. Open a cursor to retrieve all export applications for the current total identifier:

```
Select usage_type
From sa_total_usage
Where total = :ls_total
And usage_type != :ls_data_source
```

2. Fetch the max seq\_no from the sa\_export\_log table.
3. In a standard while loop
  - a) Insert a record into the export log (sa\_export\_log) for each usage type returned from the above query. Place the usage\_type into the system\_type field, and enter the status as 'R' for ready to export.
4. Return 0 if successful.
5. Return -1 if a database error occurred.

#### **I/O Specification**

All files layouts input and output

The standard adjustment input format file can be found in the "Interface File – SA Import Adj.doc" document.

## Sales audit import [saimptlog]

### Purpose

The Batch Detailed Design is a thorough definition of a single batch program / module within one functional area. The documented information is derived from this functional area's Technical Design.

### Objectives

This Batch Detailed Design must:

- Document specific functions for a single batch program,
- Enable project team review, validation and consensus regarding the individual batch program's scope,
- Document the batch program in preparation for and in response to prototyping, and
- Prepare for and provide a defined and documented framework in which to perform Development Phase activities.

### Functional Area

Sales Audit import.

### Module Affected

SAIMPTLOG (formerly saval.pc and saout.pc in 8.X)

```
saimptlog.c
saimptlog.h
saimptlog_final.c
saimptlog_init.c
saimptlog_manval.c
saimptlog_nexttsn.pc
saimptlog_nextvhn.pc
saimptlog_output.c
saimptlog_uom.pc
saimptlog_proto.h
saimptlog_rtlog.c
saimptlog_sqlldr.c*
saimptlog_tdup.c
saimptlog_tdup.h
saimptlog_nextmts.pc
saimptlog_nexttsn.pc
saimptlog_ccval.c
saimptlog_ccval.h
saimptlog_proto.h
```

### SAIMPTLOGI

```
saimptlog.c
```

---

The difference between SAIMPTLOG and SAIMPTLOGI is whether saimptlog\_sqlldr.c or saimptlog\_insert.pc is used. The former generates SQL\*Loader files while the later performs actual inserts into the database.

```

saimptlog.h
saimptlog_final.c
saimptlog_init.c
saimptlog_insert.pc*
saimptlog_manval.c
saimptlog_nexttsn.pc
saimptlog_nextvhn.pc
saimptlog_output.c
saimptlog_uom.pc
saimptlog_proto.h
saimptlog_rtlog.c
saimptlog_tdup.c
saimptlog_tdup.h
saimptlog_nexttmsn.pc
saimptlog_nextesn.pc
saimptlog_ccval.c
saimptlog_ccval.h
saimptlog_proto.h

```

#### SAIMPTLOGFIN

```

saimptlogfin.pc
saimptlog_nexttbgsn.pc
saimptlog.h

```

### Design Overview

Importing POS data is a four or five-step process depending on whether saimptlogi or saimptlog is used. Saimptlog produces SQL\*Loader files while saimptlogi does inserts directly into the database. Saimptlogi is meant for use in a trickle feed environment.

| <i>SAIMPTLOG</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                               | <b>SAIMPTLOGI</b> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| <p>SAGETREF must be run to generate the current reference files:</p> <ul style="list-style-type: none"> <li>• Items</li> <li>• Wastage</li> <li>• Sub-transaction level items</li> <li>• Primary variant relationships</li> <li>• Variable weight PLU</li> <li>• Store business day</li> <li>• Promotions</li> <li>• Code types</li> <li>• Error codes</li> <li>• Credit card validation</li> <li>• Store POS</li> <li>• Tender type</li> <li>• Merchant code types</li> </ul> |                   |

- Partner vendors
- Supplier vendors
- Employee ids

These files are all used as input to SAIMPTLOG and SAIMPTLOGI. Since SAIMPTLOG and SAIMPTLOGI can be threaded, this boosts performance by limiting interaction with the database.

SAIMPTLOG is run against each POS file. SAIMPTLOG creates a write lock for store/day and then sets the data\_status to loading until SAIMPTLOGFIN is executed. This generates distinct SQL\*Loader files for that store/day for the sa\_tran\_head, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error, sa\_customer, sa\_cust\_attrib and (optionally) sa\_missing\_tran tables. A Retek formatted voucher file is produced for processing by SAVOUCH. SAIMPTLOG may be threaded as long as the parallel executions do not include the same store/day.

SAIMPTLOGI is run against each POS file. SAIMPTLOGI creates a write lock for that store/day and then sets the data\_status to loading until SAIMPTLOGFIN is executed. This inserts data into sa\_tran\_head, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error, sa\_customer, sa\_cust\_attrib and (optionally) sa\_missing\_tran tables. A Retek formatted voucher file is produced for processing by SAVOUCH. SAIMPTLOGI may be threaded as long as the parallel executions do not include the same store/day.

SQL\*Loader is executed to load the transaction tables from the files created by SAIMPTLOG. The store/day SQL\*Loader files can be concatenated into a single file per table to optimize load times. Alternatively, multiple SQL\*Loader files can be used as input to SQL\*Loader. SQL\*Loader may not be run in parallel with itself when loading a table. Header data (primary keys) must be loaded before ancillary data (foreign keys). This means that the sa\_tran\_head table must be loaded first; sa\_tran\_item before sa\_tran\_disc; and sa\_customer before sa\_cust\_attrib. The remaining tables may be loaded in parallel.

SAVOUCH is executed to load each of the Retek formatted voucher files. SAVOUCH may not be multiply threaded.

SAIMPTLOGFIN is executed to populate the sa\_balance\_group table, cancel post voided transactions and vouchers, validate missing transactions, and to mark the import as either partially or fully complete loaded. SAIMPTLOGFIN may not be multiply threaded.

This design document encompasses SAIMPTLOG, SAIMPTLOGI and SAIMPTLOGFIN.

| <i>SAIMPTLOG</i> |                             |        |        |        |
|------------------|-----------------------------|--------|--------|--------|
| Table            | <i>Operations Performed</i> |        |        |        |
|                  | Select                      | Insert | Update | Delete |



|                         |     |     |     |    |
|-------------------------|-----|-----|-----|----|
| period                  | yes | no  | no  | no |
| store                   | yes | no  | no  | no |
| sa_system_options       | yes | no  | no  | no |
| sa_store_day            | yes | No  | Yes | no |
| sa_store_day_write_lock | yes | yes | Yes | no |

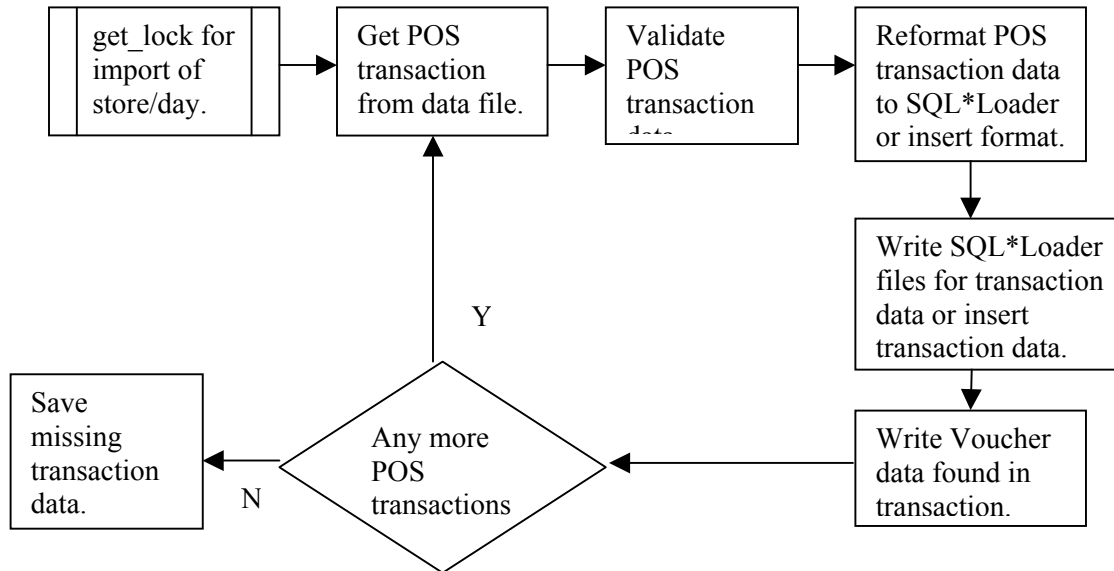
| <i>SAIMPTLOGI</i>       |                             |        |        |        |
|-------------------------|-----------------------------|--------|--------|--------|
| Table                   | <i>Operations Performed</i> |        |        |        |
|                         | Select                      | Insert | Update | Delete |
| period                  | Yes                         | No     | No     | No     |
| store                   | Yes                         | No     | No     | No     |
| sa_system_options       | Yes                         | No     | No     | No     |
| sa_store_day            | Yes                         | No     | Yes    | No     |
| sa_store_day_write_lock | Yes                         | No     | Yes    | No     |
| sa_tran_head            | No                          | Yes    | No     | No     |
| sa_customer             | No                          | Yes    | No     | No     |
| sa_cust_attrib          | No                          | Yes    | No     | No     |
| sa_tran_item            | No                          | Yes    | No     | No     |
| sa_tran_disc            | No                          | Yes    | No     | No     |
| sa_tran_tax             | No                          | Yes    | No     | No     |
| sa_tran_tender          | No                          | Yes    | No     | No     |
| sa_error                | No                          | Yes    | No     | No     |
| sa_missing_tran         | No                          | Yes    | No     | No     |

| <i>SAIMPTLOGFIN</i>     |                             |        |        |        |
|-------------------------|-----------------------------|--------|--------|--------|
| Table                   | <i>Operations Performed</i> |        |        |        |
|                         | Select                      | Insert | Update | Delete |
| period                  | yes                         | no     | No     | no     |
| store                   | yes                         | no     | No     | no     |
| sa_system_options       | yes                         | no     | No     | no     |
| sa_store_day            | yes                         | no     | yes    | no     |
| sa_store_day_write_lock | yes                         | no     | Yes    | No     |
| sa_import_log           | yes                         | no     | yes    | no     |
| sa_balance_group        | Yes                         | Yes    | No     | No     |
| sa_tran_head            | Yes                         | No     | Yes    | Yes    |
| sa_customer             | Yes                         | No     | No     | Yes    |
| sa_cust_attrib          | Yes                         | No     | No     | Yes    |
| sa_tran_item            | Yes                         | No     | No     | Yes    |
| sa_tran_disc            | Yes                         | No     | No     | Yes    |
| sa_tran_tax             | Yes                         | No     | No     | Yes    |
| sa_tran_tender          | Yes                         | No     | No     | Yes    |
| sa_error                | Yes                         | No     | No     | Yes    |
| sa_missing_tran         | Yes                         | No     | No     | Yes    |
| sa_tran_head_rev        | No                          | Yes    | No     | No     |

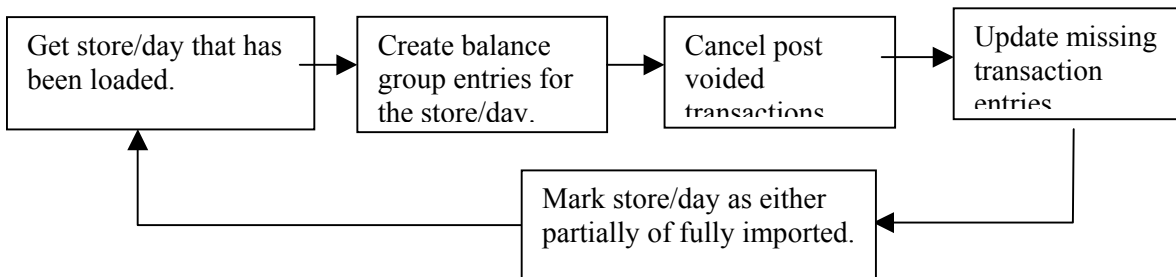
|                    |    |     |    |    |
|--------------------|----|-----|----|----|
| sa_tran_item_rev   | No | Yes | No | No |
| sa_tran_disc_rev   | No | Yes | No | No |
| sa_tran_tax_rev    | No | Yes | No | No |
| sa_tran_tender_rev | No | Yes | No | No |
| sa_error_rev       | No | Yes | No | No |

### Program Flow

#### SAIMPTLOG and SAIMPTLOGI



#### SAIMTLOGFIN



### Function Level Description

#### SAIMPTLOG and SAIMPTLOGI

As noted earlier, the difference between SAIMPTLOG and SAIMPTLOGI is whether `saimptlog_sqlldr.c` or `saimptlog_insert.pc` is used. Routines flagged with a ‡ denote that they exist in both of these modules and that behavior will depend on which module is used.

**main() [saimptlog.c]**

This should be the standard Retek main. Call **LOGON** to connect to the Sales Audit database. Call **Init** to initialize data structures and output file handles. Call **Process** to translate the RTLOG POS data into either the SQL\*Loader files or to insert the data, and to produce a Retek formatted file for vouchers. Call **Final** to close file handles and to generally clean up.

**Process() [saimptlog.c]**

For each transaction in the POS RTLOG file, call **getNextTran** to read in the data and process it. For each transaction, call **WrOutputData**<sup>‡</sup> and **writeSAVoucherData** to write the voucher transaction data to a temporary file.

**Init() [saimptlog\_init.c]**

Call **retex\_init** to initialize threading.

Get the system options by calling **fetchSaSystemOptions**.

Get the current system data (SYSDATE) by calling **fetchSysDate**. This is used later to validate the dates in the POS RTLOGs.

Initialize the RTLOG file parser by calling **InitInputData**.

Load the item data generated by SAGETREF by calling **item\_loadfile**.

Load the sub-transaction level item data generated by SAGETREF by calling **ref\_item\_loadfile**.

Load the variable weight PLU data generated by SAGETREF by calling **vupc\_loadfile**.

Load the primary variant data generated by SAGETREF by calling **primvariant\_loadfile**.

Load the store/day data generated by SAGETREF by calling **store\_day\_loadfile**.

Load the wastage data generated by SAGETREF by calling **waste\_loadfile**.

Load the promotion data generated by SAGETREF by calling **prom\_loadfile**.

Load the code type data generated by SAGETREF by calling **code\_loadfile**.

Load the error data generated by SAGETREF by calling **error\_loadfile**.

Load the store POS data generated by SAGETREF by calling **storepos\_loadfile**.

Load the tender type group and ID data generated by SAGETREF by calling **tendertype\_loadfile**.

Load the merchant code data generated by SAGETREF by calling **merchcode\_loadfile**.

Load the partner vendor data generated by SAGETREF by calling **partner\_loadfile**.

Load the supplier vendor data generated by SAGETREF by calling **supplier\_loadfile**.

Load the employee data generated by SAGETREF by calling **employee\_loadfile**.

Initialize transaction output processing by calling **InitOutputData**<sup>‡</sup>.

Initialize voucher output processing by calling **openSAVoucher**.

Initialize Oracle number arithmetic by calling **OraNumInit**.

If either of these last 2 fail, call **InitOutputClean**<sup>‡</sup>.

**Final() [saimptlog\_final.c]**

If the system option **check\_dup\_miss\_tran** is enabled, then call **tdup\_savedata** to keep track of missing transaction numbers between invocations of SAIMPTLOG or SAIMPTLOGI and call **tdup\_misstran** to create the SQL\*Loader file for the **sa\_missing\_tran** table.

Call **CreateTermRecords**<sup>‡</sup> to mark the end of the data and then call **WrOutputData**<sup>‡</sup> to write them to the temporary files.

Terminate the RTLOG file parser by calling **FinalInputData**.

Call **FinalOutputData**<sup>‡</sup> to finish any pending output processing.

Call **closeSAVoucher** to close and rename the voucher file.

Call **retex\_close** to perform program status record keeping.

Call **retex\_refresh\_thread** to refresh the thread that was used during this execution so that it can be reused.

**InitInputData() [saimptlog\_rtlog.c]**

Open the POS RTLOG file for reading.

Open a bad transaction file for writing.

Initialize the POS RTLOG transaction parser.

**getNextTran() [saimptlog\_rtlog.c]**

This function reads in each transaction (by calling **getRTLRec** for each transaction) and validates each record contained within it (by calling **procRTLFFHead**, **procRTLFTail**, **procRTLHHead**, **procRTLHTail**, **procRTLTCust**, **procRTLCAtt**, **procRTLItem**, **procRTLIDisc**, **procRTLTTax** and **procRTLTTend** as appropriate). To simplify processing, the FHEAD and FTAIL records are treated as individual transactions. The function **rtFind** is used to determine the type of the record read.

Some record types will require some extra processing:

FHEAD – Need to retain the location (store) and business date for later validations. Also, the transaction structures must be reset by calling **resetTran**. Write out a FHEAD record to the voucher file by calling **writeSAVoucherFHEAD**.

FTAIL - Write out a FTAIL record to the voucher file by calling **writeSAVoucherFTAIL**.

TTAIL – Call **chkTranFormat** to check for format and data problems. Call **chkTranTailCount** to validate the number of records found in the transaction. Call **tdup\_addtran** to check for duplicate transactions and to keep track of possible missing transactions, except when the transaction is a 'TOTAL' and its tran\_no is blank. Call **reformatTran** to format the RTLOG transaction data into SQL\*Loader flat file format. If any errors occur, call **WrBadTran** to write the failing transaction to the bad transaction file and call **resetTran** to reinitialize the RTLOG parser for the next transaction.

**FinallInputData() [saimptlog\_rtlog.c]**

Close the POS RTLOG file.

Close the bad transaction file.

**getRTLRec() [saimptlog\_rtlog.c]**

Read and return one record from the POS RTLOG file.

**rtFind() [saimptlog\_rtlog.c]**

Return the type of the record that is passed in (i.e. THEAD, TCUST, TITEM, etc).

**procRTLFFHead() [saimptlog\_rtlog.c]**

Check that this is the first record in the POS RTLOG file. Validate the business date of the data. Call **storeday\_lookup** to verify that there is a sa\_import\_log entry. If an entry is not found, generate an error and do not load any data. Call **get\_lock** to lock the store/day for importing. If a lock is not obtained, keep trying a set number of times. Call **updateDataStatus** to set the store/day's data\_status to loading (SADS\_L). If missing transactions are being tracked, call **storepos\_lookup** to get the transaction number starting and ending values and call **tdup\_loaddata** to load into memory past transaction number ranges for the current store/day. Note that the maximum transaction number allowed is 2147483647.

**procRTLFTail() [saimptlog\_rtlog.c]**

Process a FTAIL record, ensuring that it is the last record in the POS RTLOG file. The record count in the FTAIL record is checked against the number of records processed, if these do not match then records are missing and we should abort.

**procRTLHead() [saimptlog\_rtlog.c]**

Validate that the THEAD record is located within a valid position in the POS RTLOG file, after an FHEAD or TTAIL record.

Initialize the sale and tender transaction totals to 0.

**procRTLTTail() [saimptlog\_rtlog.c]**

Validate that the TTAIL record is located within a valid position in the POS RTLOG file, after a TITEM, IDISC, TTAX, TTEND, TCUST or CATT record.

**procRTLTCust() [saimptlog\_rtlog.c]**

Validate that the TCUST record is located within a transaction in the POS RTLOG file.

**procRTLCatt() [saimptlog\_rtlog.c]**

Validate that the CATT record is located within a transaction following either a TCUST or CATT record in the POS RTLOG file.

**procRTLItem() [saimptlog\_rtlog.c]**

Validate that the TITEM record is located within a transaction in the POS RTLOG file.

Convert selling unit of measure to standard, necessary.

Check if item number type is variable weight PLU. If so, decode it.

Add the quantity \* the unit retail amount to the sale transaction total.

**procRTLIDisc() [saimptlog\_rtlog.c]**

Validate that the IDISC record is located within a valid position in the POS RTLOG file, after either a TITEM or IDISC record.

Convert selling unit of measure to standard, if necessary.

Subtract the quantity \* the unit discount amount from the sale transaction total.

**procRTLTTax() [saimptlog\_rtlog.c]**

Validate that the TTAX record is located within a transaction in the POS RTLOG file.

Add the tax amount to the sale transaction total.

**procRTLTTend() [saimptlog\_rtlog.c]**

Validate that the TTEND record is located within a transaction in the POS RTLOG file.

Add the tender amount to the tender transaction total.

**resetTran() [saimptlog\_rtlog.c]**

Reinitialize the transaction structures.

**chkTranTailCount() [saimptlog\_rtlog.c]**

Checks the counters in a transaction's TTAIL record and produce an error if this figure does not match the actual number of records processed for this transaction.

**chkTranFormat() [saimptlog\_rtlog.c]**

Checks the current transaction format and content. Produces an error if more than one TCUST record is found, an IDISC record does not correspond to a TITEM record, an unknown record type is encountered or the THEAD or TTAIL records are missing from the transaction.

For each record in the transaction call **rrchk** to look for invalid characters in the record.

Call **trat\_lookup** to get the transaction type and then validate that type with the number of records within the transaction.

**rrchk() [saimptlog\_rtlog.c]**

Make sure that there are no embedded null, tab, carriage return or new line characters in the record passed in.

**WrBadTran() [saimptlog\_rtlog.c]**

Writes an erroneously formatted transaction out to the reject file for correction by an auditor. These transactions do not contain enough information to be loaded to the Sales Audit tables.

**reformatTran() [saimptlog\_rtlog.c]**

Validate and reformat the data within the transaction into the SQL\*Loader flat file format (SAIMPTLOG) or insert the data into the database (SAIMPTLOGI). This is accomplished by calling routines that know the validations and formats for each tables SQL\*Loader control file or insert statements. Start by calling **resetFmt<sup>‡</sup>** to initialize the formatting routines. The validation routines are **mvSATHead<sup>‡</sup>**, **mvSATCust<sup>‡</sup>**, **mvSACAtt<sup>‡</sup>**, **mvSATItem<sup>‡</sup>**, **mvSAIDisc<sup>‡</sup>**, **mvSATTax<sup>‡</sup>** and **mvSATTend<sup>‡</sup>**. The reformatting routines are **fmtSATTranHead<sup>‡</sup>**, **fmtSACustomer<sup>‡</sup>**, **fmtSACustAttrib<sup>‡</sup>**, **fmtSATTranItem<sup>‡</sup>**, **fmtSATTranDisc<sup>‡</sup>**, **fmtSATTranTax<sup>‡</sup>** and **fmtSATTranTend<sup>‡</sup>**. If there are any errors that prevent loading this transaction into the database, call **abortFmt<sup>‡</sup>**. If there are correctable errors, call **saErrorSATHead<sup>‡</sup>** for THEAD, TCUST and CATT records, call **saErrorSATItem<sup>‡</sup>**, **saErrorSATDisc<sup>‡</sup>**, **saErrorSATTax<sup>‡</sup>** or **saErrorSATTend<sup>‡</sup>** for the other record types.

If the transaction type is TRAT\_SALE, TRAT\_RETURN or TRAT\_EEXCH, then check that the transaction balances by comparing the sale and tender transaction totals. Generate an error if they do not match.

**updateDataStatus() [saimptlog\_datastat.pc]**

If the data status for this store/day is ready to be loaded (SADS\_R), loading (SADS\_L) or partially loaded (SADS\_P) then update it to loading (SADS\_L) and commit the change.

**mvSATHead() [saimptlog\_manval.c]**

Ensure that the transaction date and time has a valid value.

Ensure that, if they exist and sa\_system\_options.auto\_validate\_tran\_employee\_id is YSN0\_Y, the cashier and salesperson ids are valid by calling **employee\_lookup**.

Ensure that the transaction type has a valid value (code\_type of TRAT) by calling **code\_lookup**.

Ensure that, if the balancing level is register (SABL\_R) or store.tran\_no\_generated is register (STRG\_R), then the register field is populated, and that if the balancing level is cashier (SABL\_C), then the cashier field is populated.

Ensure that the transaction number exists for all transaction types except TRAT\_DCLOSE and TRAT\_TOTAL. If transaction number exists, make sure that it is numeric.

Ensure that the sub transaction type has a valid value if present (code\_type of TRAS) by calling **code\_lookup**.

Ensure that the reason code has a valid value if present (code\_type of REAC) by calling **code\_lookup**.

If the transaction type is TRAT\_PAIDIN, ensure that a reason code is present.

If the transaction type is TRAT\_PAIDOU:

If the sub transaction type is TRAS\_MV or TRAS\_EV, then validate the reason code by calling **merchcode\_lookup**, else validate the reason code by calling **code\_lookup**.

Ensure that the vendor number field is not empty.

If the sub transaction type is TRAS\_MV then validate the vendor number against the suppliers by calling **supplier\_lookup**.

Else if the sub transaction type is TRAS\_EV then validate the vendor number against the partners by calling **partner\_lookup**.  
Else we do not validate.

If the sub transaction type is TRAS\_MV or TRAS\_EV then ensure that at least one of the vendor invoice number, payment reference number and proof of delivery number fields are present.

Else we do not validate.

If the transaction type is TRAT\_TOTAL, ensure that ref\_no1 and value are not empty.

Ensure that the value has a valid numeric value if present.

Return TRUE if all validations pass, else return FALSE.

#### **mvSATCust() [saimptlog\_manval.c]**

Ensure that the customer ID has a value.

Ensure that the customer ID type has a valid value (code\_type of CIDD) by calling **code\_lookup**.

Ensure that the customers birthdate has a valid value if present.

Return TRUE if all validations pass, else return FALSE.

#### **mvSACAtt() [saimptlog\_manval.c]**

Ensure that the customer attribute type has a valid value (code\_type of SACA) by calling **code\_lookup**.

Ensure that the customer attribute value has a valid value (code\_type of attribute type) by calling **code\_lookup**.

Return TRUE if all validations pass, else return FALSE.

#### **mvSATItem() [saimptlog\_manval.c]**

Ensure that the item status has a valid value (code\_type of SASI) by calling **code\_lookup**. Also, if the tran\_type is 'SALE', 'RETURN' or 'EEXCH', then the only valid values are SASI\_S, SASI\_R, and SASI\_V. If the item status is SASI\_S then the quantity sign must be SIGN\_P. If the item status is SASI\_R then the quantity sign must be SIGN\_N.

Ensure that the item type has a valid value (code\_type of SAIT) by calling **code\_lookup**.

Ensure that the item, sub-transaction level item, or voucher number has a valid value depending on what the item type says should be present.

Ensure that the department, class, sub class and system indicator are valid if present.

Ensure that the quantity has a valid numeric value.

Ensure that the unit retail amount has a valid numeric value.

Ensure that the override reason code has a valid value (code\_type of ORRC) by calling **code\_lookup** if present.

Ensure that the original unit retail value has a valid numeric value if there is an override reason code.

Ensure that the tax indicator has a valid value (code\_type of YSNO) by calling **code\_lookup**. If the value is invalid, then an error is flagged and the value is defaulted to YSNO\_Y.

Ensure that the item swiped indicator has a valid value (code\_type of YSNO) by calling **code\_lookup**. If the value is invalid, then an error is flagged and the value is defaulted to YSNO\_Y.

Ensure that the return reason code has a valid value (code\_type SARR) by calling **code\_lookup** if present and the item status is SASI\_R.

Ensure that, if it exists and sa\_system\_options.auto\_validate\_tran\_employee\_id is YSNO\_Y, the salesperson id is valid by calling **employee\_lookup**.

Ensure that if an expiration date exists, that it is valid.

Return TRUE if all validations pass, else return FALSE.



**mvSAIDisc() [saimptlog\_manval.c]**

Ensure that the RMS promotion number has a valid value (code\_type of PRMT) by calling **code\_lookup**.

Ensure that the promotion has a valid value if present by calling **prom\_lookup**. Valid values are PRST\_A, PRST\_E and PRST\_M.

Ensure that the discount type has a valid value (code\_type of SADT) by calling **code\_lookup**.

Ensure that the quantity has a valid numeric value.

Ensure that the unit discount amount has a valid numeric value.

If the discount type is Coupon then ensure that the coupon number is present.

Return TRUE if all validations pass, else return FALSE.

**mvSATTax() [saimptlog\_manval.c]**

Ensure that the tax code has a valid value (code\_type of TAXC) by calling **code\_lookup**.

Ensure that the tax amount has a valid numeric value.

Return TRUE if all validations pass, else return FALSE.

**mvSATTend() [saimptlog\_manval.c]**

Ensure that the tender type group has a valid value (code\_type of TENT) by calling **code\_lookup**.

Ensure that the tender type ID has a valid value by calling **tendertype\_lookup**.

Ensure that the tender amount has a valid numeric value.

If the tender type group is TENT\_CCARD or TENT\_DCARD then:

Ensure that the credit card number and expiration date are valid by calling **ccval**. The expiration date may be an empty field. If it is, no validation will be performed and there is no check as to whether the credit card has expired.

Ensure that the credit card authorization source if present has a valid value (code\_type of CCAS) by calling **code\_lookup**.

Ensure that the credit card cardholder verification if present has a valid value (code\_type of CCVF) by calling **code\_lookup**.

Ensure that the credit card entry mode if present has a valid value (code\_type of CCEM) by calling **code\_lookup**.

Ensure that the credit card special condition if present has a valid value (code\_type of CCSC) by calling **code\_lookup**.

If the tender type group is Coupon then ensure that the coupon number is present.

Return TRUE if all validations pass, else return FALSE.

**nextTranSeqNo() [saimptlog\_nexttsn.c]**

Gets the next free header sequence number for use. This routine goes and gets a block of numbers when starting, and parcels them out as needed. Once they are all used up, another block is gotten.

**tdup\_savedata() [saimptlog\_tdup.c]**

Writes out what is currently known about transaction numbers for the current store/day.

**tdup\_misstran() [saimptlog\_tdup.c]**

Writes the entries for the sa\_missing\_tran table by calling **fmtSAMissTran<sup>†</sup>**.

The sa\_missing\_tran.status column will be filled in with SAMS\_M.

**tdup\_loaddata() [saimptlog\_tdup.c]**

Loads the data file of transaction number past ranges.



**tdup\_addtran() [saimptlog\_tdup.c]**

Adds a transaction number to the list of numbers encountered. If store.tran\_no\_generated is SRTG\_S, then the transaction number must be unique to the store. If store.tran\_no\_generated is SRTG\_R, then the transaction number must be unique to the store and register.

**openSAVoucher() [saimptlog\_output.c]**

Generate a temporary filename for the voucher data and open it for writing.

**closeSAVoucher() [saimptlog\_output.c]**

Close the voucher data file. If the RTLOG has been successfully processed, rename the temporary filename to a permanent name, else remove the temporary file.

**writeSAVoucherFHEAD() [saimptlog\_output.c]**

Format and writes a FHEAD record to the voucher file.

**writeSAVoucherFTAIL() [saimptlog\_output.c]**

Format and writes a FTAIL record to the voucher file.

**writeSAVoucherData() [saimptlog\_output.c]**

If the current transaction type is a sale (SALE), or a return (RETURN) and the TITEM records contain a voucher number, then reformat the TITEM records into a sold voucher data by calling **WrSoldSAVoucher**. However, if the item was voided (i.e. for the same transaction, there is an item with status 'V' for the voucher), then do not call the function.

If the current transaction type is a sale (SALE), a paid in (PAIDIN), a return (RETURN) or paid out (PAIDOU), and the tender type group is a voucher (VOUCH) then:

- if the sign of the tender amount is positive, then reformat the TTEND records into an issued voucher data by calling **WrIssuedSAVoucher**
- else, if the sign of the tender amount is negative, then reformat the TTEND records into an issued voucher data by calling **WrIssuedSAVoucher**.

(Note: it is not possible to return a voucher).

**WrSoldSAVoucher() [saimptlog\_output.c]**

Format and write a sold voucher record to the voucher file.

In addition to the fields that are currently output in this function, information about the customer who purchased the gift certificate is required in the new iss\_cust fields. This information can be copied directly from the RTLTCust record associated with the transaction being processed. The new recipient fields (name, state and country) will be stored in the RTLTIItem record reference number fields for the Sale of a gift certificate. These values provide details on the intended receiver for a gift certificate at the time of sale. This might not be provided by every POS system, in which case they would be null. Expiration date will also be stored on the RTLTIItem record and should be populated; it may also be null.

| Source                    | Target                           |
|---------------------------|----------------------------------|
| RTLTCust.name             | SA_VOUCHER.iss_cust_name         |
| RTLTCust.addr1            | SA_VOUCHER.iss_cust_addr1        |
| RTLTCust.addr2            | SA_VOUCHER.iss_cust_addr2        |
| RTLTCust.city             | SA_VOUCHER.city                  |
| RTLTCust.state            | SA_VOUCHER.state                 |
| RTLTCust.postal_code      | SA_VOUCHER.postal_code           |
| RTLTCust.country          | SA_VOUCHER.country               |
| RTLTIItem.ref_no5         | SA_VOUCHER.recipient_name        |
| RTLTIItem.ref_no6         | SA_VOUCHER.recipient_state       |
| RTLTIItem.ref_no7         | SA_VOUCHER.recipient_countr<br>y |
| RTLTIItem.expiration_date | SA_VOUCHER.exp_date              |

This function validates the datatype of numeric and date fields. The exp\_date should be added to the fields that are validated. If it is populated, it must be in a valid date format.

#### **WrRedeemedSAVoucher() [saimptlog\_output.c]**

Format and write a redeemed voucher record to the voucher file.

#### **WrIssuedSAVoucher() [saimptlog\_output.c]**

Format and write an issued voucher record to the voucher file.

In the case of a credit voucher issued during a return transaction, the iss\_cust fields will also come from the RTLTCust fields as described above. The recipient and exp\_date fields are not relevant for this type of voucher; so do not need to be copied in this function.

#### **InitOutputData() [saimptlog\_sqldr.c]**

Generate temporary filenames for the SQL\*Loader files for the sa\_tran\_head, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error, sa\_customer, sa\_cust\_attrb and (optionally, depending on the value of the system option check\_dup\_miss\_tran) sa\_missing\_tran tables.

Open all of the temporary files for writing.

#### **InitOutputClean() [saimptlog\_sqldr.c]**

Close and remove the SQL\*Loader files for the sa\_tran\_head, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error, sa\_customer, sa\_cust\_attrb and (optionally, depending on the value of the system option check\_dup\_miss\_tran) sa\_missing\_tran tables.

#### **CreateTermRecords() [saimptlog\_sqldr.c]**

Create terminating records for each record type. These records are used by SAIMPTLOGFIN to determine if SQL\*Loader has finished loading all of the transaction data for a store/day. NOT NULL column values are given in the table in the appendix. All other columns should be blank. If check\_dup\_miss\_tran is YSNO\_Y than create a sa\_missing\_tran TERM record and call **putrec** to write it to the SQL\*Loader file.

#### **WrOutputData() [saimptlog\_sqldr.c]**

Writes the current transaction to the SQL\*Loader files.

**FinalOutputData() [saimptlog\_sqlldr.c]**

Close the temporary SQL\*Loader files for the sa\_tran\_head, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error, sa\_customer, sa\_cust\_attrib and (optionally, depending on the value of the system option check\_dup\_miss\_tran) sa\_missing\_tran tables.

Rename the temporary files to *record-type\_store\_business-date\_sys-date.out* (i.e. sathead\_1000\_20000115\_20000116053302.out).

**resetFmt() [saimptlog\_sqlldr.c]**

Clears the arrays used for formatting the SQL\*Loader files.

**abortFmt() [saimptlog\_sqlldr.c]**

Dummy routine to support array inserts. See saimptlog\_insert.pc.

**saveFmt() [saimptlog\_sqlldr.c]**

Dummy routine to support array inserts. See saimptlog\_insert.pc.

**fmtSATranHead() [saimptlog\_sqlldr.c]**

Formats a sa\_tran\_head record. The status of the current transaction is updated, and the next sequential tran\_seq\_no is generated by **nextTranSeqNo** for the following transaction.

If the transaction type is not a 'TOTAL', then copy the sale transaction total to the transaction value column.

**fmtSACustomer() [saimptlog\_sqlldr.c]**

Formats a sa\_customer record.

**fmtSACustAttrib() [saimptlog\_sqlldr.c]**

Formats a sa\_cust\_attrib record.

**fmtSATranItem() [saimptlog\_sqlldr.c]**

Formats a sa\_tran\_item record. If the item contains a variable weight PLU, then call **waste\_lookup** to get the wastage type and percent. If the type is an REF, it will be converted to an ITEM. The merchandise hierarchy information (department, class, sub-class, and system indicator) associated with the item will be retrieved for this item by calling **item\_lookup**.

Produce an error if the item cannot be found, the REF item was not converted to an ITEM, the item type is not ITEM, REF or GCN, or non-numeric data is found in the quantity or amount field.

**fmtSATranDisc() [saimptlog\_sqlldr.c]**

Formats a sa\_tran\_disc record.

**fmtSATranTax() [saimptlog\_sqlldr.c]**

Formats a sa\_tran\_tax record.

**fmtSATranTend() [saimptlog\_sqlldr.c]**

Formats a sa\_tran\_tender record.

**fmtSAMissTran() [saimptlog\_sqlldr.c]**

Formats a sa\_missing\_tran record and writes it out to the SQL\*Loader file by calling **putrec**.

**setErrorSATHead() [saimptlog\_sqlldr.c]**

Sets the error indicator to YSNO\_Y for the current THEAD record.

**setErrorSATItem() [saimptlog\_sqlldr.c]**

Sets the error indicator to YSNO\_Y for the current TITEM record.

**setErrorSATDisc() [saimptlog\_sqlldr.c]**

Sets the error indicator to YSNO\_Y for the current IDISC record.

**setErrorSATTax() [saimptlog\_sqlldr.c]**

Sets the error indicator to YSNO\_Y for the current TTAX record.

**setErrorSATtend() [saimptlog\_sqlldr.c]**

Sets the error indicator to YSNO\_Y for the current TTEND record.

**InitOutputData() [saimptlog\_insert.pc]**

Allocate space for insert arrays for the sa\_tran\_head, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error, sa\_customer, sa\_cust\_attr and (optionally, depending on the value of the system option check\_dup\_miss\_tran) sa\_missing\_tran tables.

**InitOutputClean() [saimptlog\_insert.pc]**

Frees the space allocated by **InitOutputData**.

**CreateTermRecords() [saimptlog\_insert.pc]**

Create terminating records for each record type. These records are used by SAIMPTLOGFIN to determine if SQL\*Loader has finished loading all of the transaction data for a store/day. NOT NULL column values are given in the table in the appendix. All other columns should be blank. If check\_dup\_miss\_tran is YSNO\_Y than create a sa\_missing\_tran TERM record. If the array is full, first call **flushMissTranInsertArray**.

**WrOutputData() [saimptlog\_insert.pc]**

Dummy routine to support SQL\*Loader. See saimptlog\_sqlldr.c.

**FinalOutputData() [saimptlog\_insert.pc]**

Flushes the final entries in the insert arrays by calling **flushTranInsertArray** and **flushTranInsertArray**. Commit the work. Free the memory used for the arrays by calling **InitOutputClean**.

**resetFmt() [saimptlog\_insert.pc]**

Dummy routine to support SQL\*Loader. See saimptlog\_sqlldr.c.

**abortFmt() [saimptlog\_insert.pc]**

Reset array indexes to the last saved transaction.

**saveFmt() [saimptlog\_insert.pc]**

Save the array indexes for inserting the current transaction.

**fmtSATranHead() [saimptlog\_insert.pc]**

Formats a sa\_tran\_head record for array insert. If the array is full, first call **flushTranInsertArray**. The status of the current transaction is updated, and the next sequential tran\_seq\_no is generated by **nextTranSeqNo** for the following transaction. If the transaction type is not a 'TOTAL', then copy the sale transaction total to the transaction value column.

**fmtSACustomer() [saimptlog\_insert.pc]**

Formats a sa\_customer record for array insert. If the array is full, first call **flushTranInsertArray**.

**fmtSACustAttrib() [saimptlog\_insert.pc]**

Formats a sa\_cust\_attrib record for array insert. If the array is full, first call **flushTranInsertArray**.

**fmtSATranItem() [saimptlog\_insert.pc]**

Formats a sa\_tran\_item record for array insert. If the array is full, first call **flushTranInsertArray**. If the item contains a variable weight PLU, then call **waste\_lookup** to get the wastage type and percent. If the type is a REF, it will be converted to an ITEM. The merchandise hierarchy information (department, class, sub-class, and system indicator) associated with the item will be retrieved for this item by calling **item\_lookup**. Produce an error if the item cannot be found, the REF item was not converted to an ITEM, the item type is not ITEM, REF or GCN, or non-numeric data is found in the quantity or amount field.

**fmtSATranDisc() [saimptlog\_insert.pc]**

Formats a sa\_tran\_disc record for array insert. If the array is full, first call **flushTranInsertArray**.

**fmtSATranTax() [saimptlog\_insert.pc]**

Formats a sa\_tran\_tax record for array insert. If the array is full, first call **flushTranInsertArray**.

**fmtSATranTend() [saimptlog\_insert.pc]**

Formats a sa\_tran\_tender record for array insert. If the array is full, first call **flushTranInsertArray**.

**fmtSAMissTran() [saimptlog\_insert.pc]**

Formats a sa\_missing\_tran record for array insert. If the array is full, first call **flushMissTranInsertArray**.

**setErrorSATHead() [saimptlog\_insert.pc]**

Sets the error indicator to YSNO\_Y for the current THEAD record.

**setErrorSATItem() [saimptlog\_insert.pc]**

Sets the error indicator to YSNO\_Y for the current TITEM record.

**setErrorSATDisc() [saimptlog\_insert.pc]**

Sets the error indicator to YSNO\_Y for the current IDISC record.

**setErrorSATTax() [saimptlog\_insert.pc]**

Sets the error indicator to YSNO\_Y for the current TTAX record.

**setErrorSATTend() [saimptlog\_insert.pc]**

Sets the error indicator to YSNO\_Y for the current TTEND record.

**flushTranInsertArray() [saimptlog\_insert.pc]**

Inserts the contents of the transaction arrays and resets the indexes for more loading.

**flushMissTranInsertArray() [saimptlog\_insert.pc]**

Inserts the contents of the missing transaction array and resets the index for more loading.

**SAIMPTLOGFIN****main() [saimptlogfin.pc]**

This should be the standard Retek main. Call **LOGON** to connect to the Sales Audit database. Call **Init** to initialize data structures and output file handles. Call **Process** to populate the sa\_balance\_group table, to mark the import as either partially or fully complete. Call **final** to close files and generally clean up.

**init() [saimptlogfin.pc]**

**retex\_init** should be called to initialize g\_l\_restart\_max\_counter.

Get the system options by calling **fetchSaSystemOptions**.

Load the store/day data generated by SAGETREF by calling **storeday\_loadfile**.

**process() [saimptlogfin.pc]**

Fetch all store/day's that have a data status of loading (L) and that have the terminating records (sa\_tran\_head.tran\_type = TERM) on all of the tables (sa\_tran\_head, sa\_customer, sa\_cust\_attrib, sa\_tran\_item, sa\_tran\_disc, sa\_tran\_tax, sa\_tran\_tender, sa\_error and sa\_missing\_tran). Save the ROWID of these terminating records so that they can be removed. Because of trickle polling, there may be multiple records per table; they must all be present.

For each store/day fetched, get a write lock by calling **get\_lock**. If this fails, go onto the next store/day.

For each completed store/day if the DCLOSE transaction is found and the number of files expected (contained in the ref\_no1 of DCLOSE) equals the number of TERM records found, or if audit\_after\_imp\_ind is YSNO\_Y then create the balance groups by calling **balanceGroupCreate**, remove sa\_missing\_tran records that are now present by calling **fixMissTran**, and process post voids by calling **fixPostVoid**.

Delete the terminating records, if any found.

For each store/day mark the import as either partially or complete by calling **markImportDone**.

For each store/day release the import lock by calling **release\_lock**.

Do a commit after each store/day by calling **retex\_force\_commit**.

**final() [saimptlogfin.pc]**

Call **retex\_close**.

**balanceGroupCreate() [saimptlogfin.pc]**

Depending on the value of the system option balance\_level\_ind (store, register or cashier), insert the necessary records into sa\_balance\_group. The start\_datetime and end\_datetime columns should remain NULL. The bal\_group\_seq\_no is gotten from a call to **nextBalGroupSeqNo**.

**nextBalGroupSeqNo() [nextbgsn.pc]**

Gets the next free balance group sequence number for use. This routine goes and gets a block of numbers when starting, and parcels them out as needed. Once they are all used up, another block is gotten.

**fixPostVoid() [saimptlogfin.pc]**

For each transaction that has a corresponding post void transaction (tran\_type = PVOID) where sale.tran\_no = cancel.orig\_tran\_no and sale.register = cancel.orig\_reg\_no and store\_day\_seq\_no's match, set the status to SAST\_V. Also, if that transaction contained a voucher (either as an item or as a tender), then call the package function

SA\_VOUCHER\_SQL.POST\_VOID\_VOUCHER to undo any processing on this voucher. Call TRANSACTION\_SQL.CREATE\_REVISIONS to create revision. Update sa\_tran\_head with the new revision number, setting the status to postvoided.

#### **fixMissTran() [saimptlogfin.pc]**

Remove sa\_missing\_tran records that may now be present because data was processed out of order.

#### **markImportDone() [saimptlogfin.pc]**

Get the current count of files loaded. Mark the import as either fully (F) or partially (P) loaded by updating the sa\_store\_day table's data\_status column. This is determined by the presence of a transaction with a type of store/day closed (DCLOSE).

If there was a DCLOSE transaction, get the number of files expected contained in the ref\_no1 field, then if the number of files expected equals the number loaded, update the sa\_store\_day table's data\_status, audit\_status and files\_loaded columns. If a DCLOSE record was found and the numbers match, set data\_status to Fully Loaded, audit\_status to Audited, else data\_status = Partially Loaded, audit\_status = Unaudited. Increment files\_loaded by the number of TERM records found. If the import was expected, then set status to loaded (L), else set it to unexpected (U). This is determined by calling **storeday\_lookup**.

#### **Stored Procedures / Shared Modules (Maintainability)**

Refer to the following documents for more details:

Package detail design - salock.doc

Functional Design - SA\_misc.doc

Technical Design - SA\_misc.doc

|                      |                                                                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Retek_init           |                                                                                                                                         |
| Retek_close          |                                                                                                                                         |
| Retek_refresh_thread |                                                                                                                                         |
| fetchSaSystemOptions | Fetch the values from the sa_system_options table.                                                                                      |
| fetchSysDate         | Fetch the current SYSDATE value.                                                                                                        |
| trat_lookup          | Look up TRAT code types and convert them to their sequence number.                                                                      |
| tent_lookup          | Look up TENT code types and convert them to their sequence number.                                                                      |
| get_lock             | used to establish a read lock on a store/day.                                                                                           |
| release_lock         | used to release a store/day lock.                                                                                                       |
| storeday_loadfile    | Loads the store/day data file generated by SAGETREF into memory.                                                                        |
| storeday_lookup      | Checks that a store business day has an import record.                                                                                  |
| item_loadfile        | Loads the item data file generated by SAGETREF into memory.                                                                             |
| item_lookup          | Looks up an item and returns the data (department, class, sub-class and system indicator) associated with it.                           |
| ref_item_loadfile    | Loads the sub-transaction item (ref) data file generated by SAGETREF into memory.                                                       |
| ref_item_lookup      | Looks up a sub-transaction level item.                                                                                                  |
| vupc_loadfile        | Loads the variable weight PLU data file generated by SAGETREF into memory.                                                              |
| vupc_lookup          | Looks up a variable PLU. Call vupc_lookup to see if it is a variable PLU. If it is a variable UPC, then set the variable parts to zero. |



|                     |                                                                       |
|---------------------|-----------------------------------------------------------------------|
| prom_loadfile       | Loads the promotion data file generated by SAGETREF into memory.      |
| prom_lookup         | Checks that a promotion exists.                                       |
| waste_loadfile      | Loads the wastage data file generated by SAGETREF into memory.        |
| waste_lookup        | Looks up the wastage for an item.                                     |
| code_loadfile       | Loads the code type data file generated by SAGETREF into memory.      |
| code_lookup         | Checks that a code type/code exists.                                  |
| error_loadfile      | Loads the error data file generated by SAGETREF into memory.          |
| error_lookup        | Looks up the error and the system codes that we are interested in it. |
| storepos_loadfile   | Loads the store POS data file generated by SAGETREF into memory.      |
| storepos_lookup     | Looks up the store POS data that we are interested in it.             |
| tendertype_loadfile | Loads the tender type data file generated by SAGETREF into memory.    |
| tendertype_lookup   | Checks that a tender type group and ID exists.                        |
| merchcode_loadfile  | Loads the merchant code data file generated by SAGETREF into memory.  |
| merchcode_lookup    | Looks up the merchant code data that we are interested in it.         |
| partner_loadfile    | Loads the partner data file generated by SAGETREF into memory.        |
| partner_lookup      | Looks up the partner data that we are interested in it.               |
| supplier_loadfile   | Loads the supplier data file generated by SAGETREF into memory.       |
| supplier_lookup     | Looks up the supplier data that we are interested in it.              |
| putrec              | Writes a record to a file.                                            |

---

### Input Specifications

The input files for Item, Wastage, sub-transaction level item (reference item), Variable PLU, Store Day, Promotions, Code Types, and Errors are all documented in Batch Design – SAGETREF.doc.

The RTLOG file format is documented in Interface file – SA RTLOG.doc.

Date columns should always be converted to characters with a format of 'YYYYMMDDHH24MISS'. Single digit MM, DD, HH24, MI and SS values need to be 0 padded.

Char and Numeric ID Field Types should be left justified and padded with spaces.

Number Field types should be right justified and padded with zeros. If a Number Field is NULL, than it should be blank not 0's.

### Output Specifications

The filename convention for the SQL\*Loader output files will be *table\_store\_businessdate\_curdatetime.out* where *table* is sathead, satitem, satdisc, sattax, sattend, sacust, sacustatt, or samistr (i.e. sathead\_1000\_20000115\_20000116053302.out for the sa\_tran\_head table). Similarly, the filename convention for the Voucher output file is *savouch\_store\_businessdate\_curdatetime.out*. The files should start out with a temporary name



generated by the Unix `tempnam(3S)` call and then be renamed with Unix `rename(2)` call when the files are complete (see the Unix man pages in the indicated sections for usage details).

The filename convention for storing missing transactions between invocations of SAIMPTLOG is `tdup_store_businessdate.dat`.

Date columns should always be converted to characters with a format of 'YYYYMMDDHH24MISS'. Single digit MM, DD, HH24, MI and SS values need to be 0 padded.

When selecting columns that contain quantities or amounts from the database, the value should be multiplied by 10000 to remove the decimal point. Decimal points are not supposed to be in Retek files. The only exception to this is SQL\*Loader files.

Char and Numeric ID Field Types should be left justified and padded with spaces.

Number Field types should be right justified and padded with zeros. If a Number Field is NULL, than it should be blank not 0's.

The voucher file format is documented in Interface file – SA VOUCH.doc.

SQL\*Loader Control Files will be provided that match the format of the data files. These files will be named *table.ctl*. The format of the SQL\*Loader files is as follows:

| Table Name   | Column Name         | Field Type       | Field Width | Position | Description                           |
|--------------|---------------------|------------------|-------------|----------|---------------------------------------|
| Sa_tran_head | Tran_seq_no         | Integer external | 20          | 1:20     | Format is<br>YYYYMM<br>DDHH24MI<br>SS |
|              | Rev_no              | Integer external | 3           | 21:23    |                                       |
|              | Store_day_seq_no    | Integer external | 20          | 24:43    |                                       |
|              | Tran_datetime       | date             | 14          | 44:57    |                                       |
|              | Register            | char             | 5           | 58:62    |                                       |
|              | Tran_no             | Integer external | 10          | 63:72    |                                       |
|              | Cashier             | char             | 10          | 73:82    |                                       |
|              | Salesperson         | char             | 10          | 83:92    |                                       |
|              | Tran_type           | char             | 6           | 93:98    |                                       |
|              | Sub_tran_type       | char             | 6           | 99:104   |                                       |
|              | Orig_tran_no        | Integer external | 10          | 105:114  |                                       |
|              | Orig_reg_no         | char             | 5           | 115:119  |                                       |
|              | Ref_no1             | char             | 30          | 120:149  |                                       |
|              | Ref_no2             | char             | 30          | 150:179  |                                       |
|              | Ref_no3             | char             | 30          | 180:209  |                                       |
|              | Ref_no4             | char             | 30          | 210:239  |                                       |
|              | Reason_code         | char             | 6           | 240:245  |                                       |
|              | Vendor_no           | char             | 10          | 246:255  |                                       |
|              | Vendor_inv_no       | char             | 30          | 256:285  |                                       |
|              | Payment_ref_no      | char             | 16          | 286:301  |                                       |
|              | Proof_of_delivery_n | char             | 30          | 302:331  |                                       |
|              | o                   |                  |             |          |                                       |

| Table Name   | Column Name               | Field Type       | Field Width | Position | Description                                             |
|--------------|---------------------------|------------------|-------------|----------|---------------------------------------------------------|
|              | Status                    | char             | 6           | 332:337  | Includes an optional negative sign and a decimal point. |
|              | Value                     | char             | 22          | 338:359  |                                                         |
|              | Pos_tran_ind              | char             | 1           | 360:360  |                                                         |
|              | Update_id                 | char             | 30          | 361:390  | Format is YYYYMM DDHH24MI SS                            |
|              | Update_datetime           | date             | 14          | 391:404  |                                                         |
|              | Error_ind                 | char             | 1           | 405:405  |                                                         |
| Sa_tran_item | Tran_seq_no               | Integer external | 20          | 1:20     | Includes an optional negative sign and a decimal point. |
|              | Item_seq_no               | Integer external | 4           | 21:24    |                                                         |
|              | Item_status               | char             | 6           | 25:30    |                                                         |
|              | Item_type                 | char             | 6           | 31:36    |                                                         |
|              | Item                      | char             | 25          | 37:61    |                                                         |
|              | Ref_item                  | char             | 25          | 62:86    |                                                         |
|              | Non_merch_item            | char             | 25          | 87:111   |                                                         |
|              | Voucher_no                | char             | 16          | 112:127  |                                                         |
|              | Dept                      | Integer external | 4           | 128:131  |                                                         |
|              | Class                     | Integer external | 4           | 132:135  |                                                         |
|              | Subclass                  | Integer external | 4           | 136:139  |                                                         |
|              | Qty                       | decimal external | 14          | 140:153  |                                                         |
|              | Unit_retail               | decimal external | 21          | 154:174  |                                                         |
|              | Selling UOM               | char             | 4           | 175:178  |                                                         |
|              | Override_reason           | char             | 6           | 179:184  |                                                         |
|              | Orig_unit_retail          | decimal external | 21          | 185:205  | Includes a decimal point.                               |
|              | Standard_orig_unit_retail | decimal external | 21          | 206:226  |                                                         |
|              | Tax_ind                   | char             | 1           | 227:227  |                                                         |
|              | Item_swiped_ind           | char             | 1           | 228:228  |                                                         |
|              | Error_ind                 | char             | 1           | 229:229  |                                                         |
|              | Drop_ship_ind             | char             | 1           | 230:230  |                                                         |
|              | Waste_type                | char             | 6           | 231:236  |                                                         |

| Table Name   | Column Name          | Field Type       | Field Width | Position | Description                                             |
|--------------|----------------------|------------------|-------------|----------|---------------------------------------------------------|
|              | Waste_pct            | decimal external | 12          | 237:248  | Includes a decimal point.                               |
|              | Pump                 | char             | 8           | 249:256  |                                                         |
|              | Return_reason_code   | char             | 6           | 257:262  |                                                         |
|              | Salesperson          | char             | 10          | 263:272  |                                                         |
|              | Expiration_date      | Date             | 8           | 273:280  | Format is YYYYMM DD                                     |
|              | Standard_qty         | decimal external | 14          | 281:294  | Includes an optional negative sign and a decimal point. |
|              | Standard_unit_retail | decimal external | 21          | 295:315  | Includes a decimal point.                               |
|              | Standard_uom         | char             | 4           | 316:319  |                                                         |
|              | Ref_no5              | char             | 30          | 320:349  |                                                         |
|              | Ref_no6              | char             | 30          | 350:379  |                                                         |
|              | Ref_no7              | char             | 30          | 380:409  |                                                         |
|              | Ref_no8              | char             | 30          | 410:439  |                                                         |
| Sa_tran_disc | Tran_seq_no          | Integer external | 20          | 1:20     |                                                         |
|              | Item_seq_no          | Integer external | 4           | 21:24    |                                                         |
|              | Discount_seq_no      | Integer external | 4           | 25:28    |                                                         |
|              | rms_promo_type       | char             | 6           | 29:34    |                                                         |
|              | Promotion            | Integer external | 10          | 35:44    |                                                         |
|              | Discount_type        | char             | 6           | 45:50    |                                                         |
|              | Coupon_no            | char             | 16          | 51:66    |                                                         |
|              | Coupon_ref_no        | char             | 16          | 67:82    |                                                         |
|              | Qty                  | decimal external | 14          | 83:96    | Includes an optional negative sign and a decimal point. |
|              | Unit_discount_amt    | decimal external | 21          | 97:117   | Includes a decimal point.                               |
|              | Standard_qty         | decimal external | 14          | 118:131  | Includes an optional negative sign and a decimal point. |
|              | Standard_unit_disco  | decimal external | 21          | 132:152  | Includes a                                              |

| Table Name     | Column Name        | Field Type       | Field Width | Position | Description                                             |
|----------------|--------------------|------------------|-------------|----------|---------------------------------------------------------|
|                | unt_amt            |                  |             |          | decimal point.                                          |
|                | Ref_no13           | char             | 30          | 153:182  |                                                         |
|                | Ref_no14           | char             | 30          | 183:212  |                                                         |
|                | Ref_no15           | char             | 30          | 213:242  |                                                         |
|                | Ref_no16           | char             | 30          | 243:272  |                                                         |
|                | Error_ind          | char             | 1           | 273:273  |                                                         |
| Sa_tran_tax    | Tran_seq_no        | Integer external | 20          | 1:20     |                                                         |
|                | Tax_code           | char             | 6           | 21:26    |                                                         |
|                | Tax_seq_no         | Integer external | 4           | 27:30    |                                                         |
|                | Tax_amt            | decimal external | 22          | 31:52    | Includes an optional negative sign and a decimal point. |
|                | Error_ind          | char             | 1           | 53:53    |                                                         |
|                | Ref_no17           | char             | 30          | 54:83    |                                                         |
|                | Ref_no18           | char             | 30          | 84:113   |                                                         |
|                | Ref_no19           | char             | 30          | 114:143  |                                                         |
|                | Ref_no20           | char             | 30          | 144:173  |                                                         |
|                |                    |                  |             |          |                                                         |
| Sa_tran_tender | Tran_seq_no        | Integer external | 20          | 1:20     |                                                         |
|                | Tender_seq_no      | Integer external | 4           | 21:24    |                                                         |
|                | Tender_type_group  | char             | 6           | 25:30    |                                                         |
|                | Tender_type_id     | Integer external | 6           | 31:36    |                                                         |
|                | Tender_amt         | decimal external | 22          | 37:58    | Includes an optional negative sign and a decimal point. |
|                |                    |                  |             |          |                                                         |
|                | Cc_no              | Integer external | 16          | 59:74    |                                                         |
|                | Cc_cc_exp_date     | date             | 8           | 75:82    | Format is YYYYMMDD                                      |
|                |                    |                  |             |          |                                                         |
|                | Cc_auth_no         | char             | 16          | 83:98    |                                                         |
|                | Cc_auth_src        | char             | 6           | 99:104   |                                                         |
|                | Cc_entry_mode      | char             | 6           | 105:110  |                                                         |
|                | Cc_cardholder_verf | char             | 6           | 111:116  |                                                         |
|                | Cc_term_id         | char             | 5           | 117:121  |                                                         |
|                | Cc_spec_cond       | char             | 6           | 122:127  |                                                         |
|                | Voucher_no         | char             | 16          | 128:143  |                                                         |
|                | Coupon_no          | char             | 16          | 144:159  |                                                         |
|                | Coupon_ref_no      | char             | 16          | 160:175  |                                                         |
|                | Ref_no9            | char             | 30          | 176:205  |                                                         |
|                | Ref_no10           | char             | 30          | 206:235  |                                                         |
|                |                    |                  |             |          |                                                         |

| Table Name      | Column Name        | Field Type       | Field Width | Position | Description                           |
|-----------------|--------------------|------------------|-------------|----------|---------------------------------------|
|                 | Ref_no11           | char             | 30          | 236:265  |                                       |
|                 | Ref_no12           | char             | 30          | 266:295  |                                       |
|                 | Error_ind          | char             | 1           | 296:296  |                                       |
| Sa_customer     | Tran_seq_no        | Integer external | 20          | 1:20     | Format is<br>YYYYMM<br>DD             |
|                 | Cust_id            | char             | 16          | 21:36    |                                       |
|                 | Cust_id_type       | char             | 6           | 37:42    |                                       |
|                 | Name               | char             | 40          | 43:82    |                                       |
|                 | Addr1              | char             | 40          | 83:122   |                                       |
|                 | Addr2              | char             | 40          | 123:162  |                                       |
|                 | City               | char             | 30          | 163:192  |                                       |
|                 | Sate               | char             | 3           | 193:195  |                                       |
|                 | Postal_code        | char             | 10          | 196:205  |                                       |
|                 | Country            | char             | 3           | 206:208  |                                       |
|                 | Home_phone         | char             | 20          | 209:228  |                                       |
|                 | Work_phone         | char             | 20          | 229:248  |                                       |
|                 | E_mail             | char             | 100         | 249:348  |                                       |
|                 | birthdate          | date             | 8           | 349:356  |                                       |
| Sa_cust_attrib  | Tran_seq_no        | Integer external | 20          | 1:20     |                                       |
|                 | Attrib_seq_no      | char             | 4           | 21:24    |                                       |
|                 | Attrib_type        | char             | 6           | 25:30    |                                       |
|                 | Attrib_value       | char             | 6           | 31:36    |                                       |
| Sa_error        | Error_seq_no       | Integer external | 20          | 1:20     | Format is<br>YYYYMM<br>DDHH24MI<br>SS |
|                 | Store_day_seq_no   | Integer external | 20          | 21:40    |                                       |
|                 | Bal_group_seq_no   | Integer external | 20          | 41:60    |                                       |
|                 | Total_seq_no       | Integer external | 20          | 61:80    |                                       |
|                 | Tran_seq_no        | Integer external | 20          | 81:100   |                                       |
|                 | Error_code         | char             | 25          | 101:125  |                                       |
|                 | Key_value_1        | Integer external | 4           | 126:129  |                                       |
|                 | Key_value_2        | Integer external | 4           | 130:133  |                                       |
|                 | Rec_type           | char             | 6           | 134:139  |                                       |
|                 | Store_override_ind | char             | 1           | 140:140  |                                       |
|                 | Hq_override_ind    | char             | 1           | 141:141  |                                       |
|                 | Update_id          | char             | 30          | 142:171  |                                       |
|                 | Update_datatime    | date             | 14          | 172:185  |                                       |
|                 | Orig_value         | char             | 50          | 186:235  |                                       |
| Sa_missing_tran | Miss_tran_seq_no   | Integer external | 20          | 1:20     |                                       |
|                 | Store_day_seq_no   | Integer external | 20          | 21:40    |                                       |
|                 | Register           | char             | 5           | 41:45    |                                       |
|                 | Tran_no            | Integer external | 10          | 46:55    |                                       |

| Table Name | Column Name | Field Type | Field Width | Position | Description |
|------------|-------------|------------|-------------|----------|-------------|
|            | status      | char       | 6           | 56:61    |             |

### Database Integrity

This information derives from the Database Considerations within the Process / Functional Overview (PFO), the Conversation Flow and Database Objects of the Technical Design.

### Parameter validation

Parameter validation focuses on validating parameter data that is being passed from calling modules.

### Integrity Constraints

Operations that affect other entities in the system must be validated to ensure that integrity constraints have not been violated. If a record cannot exist in the system without a related parent record existing first, it is essential that the application enforce this constraint. Similarly, if a record cannot be deleted due to the existence of child records in the system the application should prevent the user from performing a delete operation.

### Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: These programs (SAIMPTLOG and SQL\*Loader or SAIMPTLOGI, and SAIMPTLOGFIN) are the second step in the batch process for loading customer POS data into the Sales Audit database.

Pre-Processing: SAGETREF must be run before importing POS logs. POS logs must be converted into the Retek TLOG format by the customer (Unless the saimptlog\_rtlog.c module is rewritten by the customer to handle their POS log files).

Threading Scheme: N/A

### Locking Strategy

In conjunction with the Performance and the Scheduling Considerations section, this section should describe the locking (and release) strategy required beyond the preset Retek standards. It should describe how the module accesses data and the 'hold' or 'lock' it has on a database and / or its records, during processing. It should also describe the 'lock' release.

### Restart / Recovery

The logical unit of work for SAIMPTLOG is defined as a single POS file. This POS file may or may not represent a complete store day.

The logical unit of work for SAIMPTLOGFIN is defined as a store/day. This does not follow the usual restart/recovery. A commit is done after each store/day is processed. This program will then naturally pick up where it left off if it is restarted.

## Performance

In conjunction with the Scheduling Considerations and Locking Strategy sections, the optimization considerations of a batch module must adhere to Retek standards. This section should call out special performance considerations that may exceed current documented Retek practices. Such considerations should be the basis for update to Retek standards. Each database operation should be optimized based on quantity and quality of the database transactions. Batch modules are executed on the database or dedicated batch server and thus there are no additional performance gains to forcing database interaction logic onto the server.

## Security Considerations

POS data contains credit card data. The RTLOG input file and satend SQL\*Loader output file both contain credit card numbers. Access to these files is controlled solely by Unix file permissions.

## Design Assumptions

Design assumptions are presumed design factors, inferred from current information, expected to hold true over the life of the project. Design assumptions must be documented in order to justify and validate derived design considerations with the Business Requirements (documented within the BRD and PFO).

## Appendix

| CreateTermRecords |                  |                                                                                              |
|-------------------|------------------|----------------------------------------------------------------------------------------------|
| Table             | Column           | Value                                                                                        |
| sa_tran_head      | tran_seq_no      | Determined by saimptlog.                                                                     |
|                   | rev_no           | 001                                                                                          |
|                   | store_day_seq_no | Same as last transaction processed.                                                          |
|                   | tran_datetime    | Business Date at midnight                                                                    |
|                   | tran_no          | 0000000000                                                                                   |
|                   | tran_type        | TERM                                                                                         |
|                   | status           | W                                                                                            |
|                   | pos_tran_ind     | N                                                                                            |
|                   | ref_no1          | Corresponding sa_missing_tran.miss_tran_seq_no if sa_system_options.check_dup_miss_tran = Y. |
|                   | update_id        | 00000000000000000000000000000000                                                             |
|                   | update_datetime  | SYSDATE                                                                                      |
|                   | error_ind        | N                                                                                            |
|                   |                  |                                                                                              |
| sa_customer       | tran_seq_no      | Same as sa_tran_head.tran_seq_no.                                                            |
|                   | cust_id          | 0000000000000000                                                                             |
|                   | cust_id_type     | TERM                                                                                         |
|                   |                  |                                                                                              |
| sa_cust_attrib    | tran_seq_no      | Same as sa_tran_head.tran_seq_no.                                                            |
|                   | attrib_type      | TERM                                                                                         |
|                   | attrib_value     | TERM                                                                                         |
|                   |                  |                                                                                              |
| sa_tran_item      | tran_seq_no      | Same as sa_tran_head.tran_seq_no.                                                            |

| CreateTermRecords                                                  |                        |                                     |
|--------------------------------------------------------------------|------------------------|-------------------------------------|
| Table                                                              | Column                 | Value                               |
|                                                                    | item_seq_no            | 0001                                |
|                                                                    | Item_status            | S                                   |
|                                                                    | item_type              | TERM                                |
|                                                                    | qty                    | 000000000000                        |
|                                                                    | unit retail sign       | P                                   |
|                                                                    | unit retail            | 00000000000000000000                |
|                                                                    | tax ind                | N                                   |
|                                                                    | item swiped ind        | N                                   |
|                                                                    | error_ind              | N                                   |
| sa_tran_disc                                                       | tran_seq_no            | Same as sa_tran_head.tran_seq_no.   |
|                                                                    | item_seq_no            | 0001                                |
|                                                                    | rms_promo_type         | TERM                                |
|                                                                    | discount_seq_no        | 0001                                |
|                                                                    | discount_type          | TERM                                |
|                                                                    | qty                    | 000000000000                        |
|                                                                    | unit_discount_amt sign | P                                   |
|                                                                    | unit_discount_amt      | 00000000000000000000                |
|                                                                    | error_ind              | N                                   |
| sa_tran_tax                                                        | tran_seq_no            | Same as sa_tran_head.tran_seq_no.   |
|                                                                    | tax_code               | TERM                                |
|                                                                    | tax_seq_no             | 0001                                |
|                                                                    | tax_amt sign           | P                                   |
|                                                                    | tax_amt                | 00000000000000000000                |
|                                                                    | error_ind              | N                                   |
| sa_tran_tender                                                     | tran_seq_no            | Same as sa_tran_head.tran_seq_no.   |
|                                                                    | tender_seq_no          | 0001                                |
|                                                                    | tran_type_group        | TERM                                |
|                                                                    | tran_type_id           | 000000                              |
|                                                                    | tender_amt sign        | P                                   |
|                                                                    | tender_amt             | 00000000000000000000                |
|                                                                    | error_ind              | N                                   |
| sa_error                                                           | error_seq_no           | Determined by saimptlog.            |
|                                                                    | store_day_seq_no       | Same as last transaction processed. |
|                                                                    | tran_seq_no            | Same as sa_tran_head.tran_seq_no.   |
|                                                                    | error code             | TERM MARKER_NO_ERROR                |
|                                                                    | record type            | THEAD                               |
|                                                                    | store_override_ind     | N                                   |
|                                                                    | hq_override_ind        | N                                   |
|                                                                    | update_id              | TLOG                                |
|                                                                    | update_datetime        | SYSDATE                             |
| This is present only if sa_system_options.check_dup_miss_tran = Y. |                        |                                     |
| sa_missing_tran                                                    | miss_tran_seq_no       | Determined by saimptlog.            |



| CreateTermRecords |                  |                                     |
|-------------------|------------------|-------------------------------------|
| Table             | Column           | Value                               |
|                   | store_day_seq_no | Same as last transaction processed. |
|                   | tran_no          | -000000001                          |
|                   | status           | M                                   |
|                   |                  |                                     |

## Stock ledger append [salapnd]

### Design Overview

The sole purpose of this program is to move data from the transaction staging table into the historical transaction table. This requires placing a lock on the staging table to ensure that no new data will be added to it while the movement is occurring (to handle trickling or real-time processing), moving the data to the historical table, and finally truncating the data from the staging table.

### Scheduling Constraints

Processing Cycle: Should occur after all extractions have completed (RMS – saldly, RDW etc.)

Scheduling Diagram: N/A

Pre-Processing: salstage, all extraction, and all processing

Post-Processing: N/A

Threading Scheme: Threading will be implicit via the use of the Oracle Parallel Query Option. The insert/select query should be tuned for each specific environment to achieve the best throughput.

### Restart Recovery

No specific restart/recovery scheme needs to be defined because of the limited scope of the module. However, in cases where the tran\_data table is very large, a particularly large rollback segment may be specified to reduce the risk of exceeding rollback segment space. This will depend on the size of normal rollback segments and the size of the tran\_data table.

### Program Flow

N/A

### Shared Modules

N/A

### Function Level Description

Init():

The rollback segment should be specified to a large enough size so that the entire rollup will complete in a single transaction.

Process()

This function first calls the lock\_table function to lock the if\_tran\_data table. It then calls the append\_history function to append the data from if\_tran\_data on to the tran\_data\_history table. Finally, it calls lock table to release the lock on the tran\_data table.

Lock\_table()

This function places or remove a lock on a table passed in. It will place or release the lock depending on a input parameter.

#### `Append_history()`

This function appends the data on `if_tran_data` on to the `tran_data_history` table.. This is done using an insert/select statement with a hint for parallelism. The degree of parallelism should be customized to each run-time environment.

## Sales daily [saldly]

### Design Overview

First the program reads in the calendar type option (regular or 454), then gets last day of the month for the previous and current month. If a month has been closed (by running 'salmth'), no further transactions for that month can be processed. If any are found when this program runs, they are not processed and a message is logged indicating the problem. The transactions will be left on IF\_TRAN\_DATA after the run. A decision must be made where to put the data. If it is decided to put this data in the current month, the transaction date should be changed accordingly, and the transaction will then be processed during the next run of this program.

The main process of this program reads IF\_TRAN\_DATA with a processed\_ind of 'N' and transaction date greater than the last end-of-month date and the appropriate field on DAILY\_DATA is updated based on the transaction type. Rows on DAILY\_DATA are inserted if does not exist.

After the tran data has been processed the processed\_ind field on tran data is updated to 'Y' which signifies that it has been processed.

| TABLE            | INDEX | SELECT | INSERT | UPDATE | DELETE |
|------------------|-------|--------|--------|--------|--------|
| DEPT             | No    | Yes    | No     | No     | No     |
| SUBCLASS         | Yes   | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS   | No    | Yes    | No     | No     | No     |
| SYSTEM_VARIABLES | No    | Yes    | No     | No     | No     |
| IF_TRAN_DATA     | No    | Yes    | No     | Yes    | No     |
| DAILY_DATA       | Yes   | No     | Yes    | Yes    | No     |

### Input Specifications

#### Driving Cursor:

```
EXEC SQL DECLARE c_sales CURSOR FOR
 SELECT dept,
 class,
 subclass,
 store,
 wh,
 DECODE(SIGN(tran_date - TO_DATE(:ps_last_eom_date_for_eow, 'YYYYMMDD')),
 -1, :ps_vdate,
 0, :ps_vdate,
 TO_CHAR(tran_date, 'YYYYMMDD')),
 SUM(DECODE(tran_code,1,NVL(units,0),0)),
 SUM(DECODE(tran_code,1,NVL(total_cost,0),0)),
 SUM(DECODE(tran_code,1,NVL(total_retail,0),0)),
 SUM(DECODE(:pi_stklmgr_vat_incl_retl_ind + :pi_vat_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_cost,0),0)),
 SUM(DECODE(tran_code,4,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,13,NVL(total_retail,0),0)),
 SUM(DECODE(tran_code,14,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,20,NVL(total_cost,0),0)),
```

```

SUM(DECODE(tran_code,20,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,22,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,22,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,24,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,24,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,26,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,28,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,29,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,30,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,30,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,31,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,31,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,32,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,32,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,33,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,33,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,34,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,34,NVL(total_retail,0),0)),
SUM(DECODE(tran_code,36,NVL(total_cost,0),0)),
SUM(DECODE(tran_code,36,NVL(total_retail,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))
FROM if_tran_data,
v_restart_dept rv
WHERE rv.driver_value = if_tran_data.dept
AND rv.driver_name = :ps_restart_driver_name
AND rv.num_threads = :pi_restart_num_threads
AND rv.thread_val = :ps_thread_val
AND (if_tran_data.dept > NVL(:ps_restart_dept,if_tran_data.dept - 1) OR
 (if_tran_data.dept = :ps_restart_dept AND
 (if_tran_data.class > :ps_restart_class OR
 (if_tran_data.class = :ps_restart_class AND
 (if_tran_data.subclass > :ps_restart_subclass)))))
GROUP BY dept,
class,
subclass,
store,
wh,
DECODE(SIGN(tran_date - TO_DATE(:ps_last_eom_date_for_eow, 'YYYYMMDD')),
-1, :ps_vdate,
0, :ps_vdate,
TO_CHAR(tran_date, 'YYYYMMDD'))
ORDER BY dept,
class,
subclass;

```

## Output Specifications

### Scheduling Considerations

Processing Cycle: PHASE 3 (daily)

Scheduling Diagram: N/A

Pre-Processing: run salstage to move records from tran\_data to if\_tran\_data

Post-Processing: N/A

Threading Scheme: STORE\_WH  
V\_restart\_store\_wh

## End of half stock ledger processing [saleoh]

### Design Overview

This program deletes rows from MONTH\_DATA, DAILY\_DATA, WEEK\_DATA, HALF\_DATA, MONTH\_DATA\_BUDGET, and HALF\_DATA\_BUDGET which are eighteen months or older.

For each subclass/location, this program will insert one row into half\_data for the next half. For each department/location, it inserts six rows (one for each month) into month\_data\_budget and one row into half\_data\_budget for the same half of next year. The shrinkage percent on half\_data\_budget for the same half of next year will be calculated from half\_data of the same half of current year. For example, if this program runs at the end of the first half of 1997 (i.e. half\_no = 19971) :

- Insert one row on half\_data for half\_no = 19972 for all subclass/location combinations
- Insert six rows on month\_data\_budget for half\_no = 19981 for all department/location combinations
- Insert one row on half\_data\_budget for half\_no = 19981 for all department/location combinations
- For each department/location, sum up shrinkage amount and sales amount from all the subclass/location records on half\_data with subclass belongs to this department and half\_no = 19971 and calculates a shrinkage percent to update half\_data\_budget.shrinkage\_pct for this department/location and half\_no = 19981.

| TABLE             | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-------------------|-------|--------|--------|--------|--------|
| DAILY_DATA        | Yes   | No     | No     | No     | Yes    |
| HALF_DATA         | No    | Yes    | Yes    | No     | Yes    |
| HALF_DATA_BUDGET  | No    | Yes    | Yes    | No     | Yes    |
| MONTH_DATA        | Yes   | No     | No     | No     | Yes    |
| MONTH_DATA_BUDGET | No    | Yes    | Yes    | No     | Yes    |
| SYSTEM_VARIABLES  | No    | Yes    | No     | Yes    | No     |
| SYSTEM_OPTIONS    | No    | Yes    | No     | No     | No     |
| WEEK_DATA         | No    | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: PHASE 3 (end-of-half)

Scheduling Diagram: Must run after completion of the salmth program for month 6 of the current half, and before running the salmth program for month 1 of next half.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

**Function Level Description**

This program

- first checks to make sure that salmth for month 6 of current half has been completed.
  - purges rows on daily\_data, week\_data, month\_data, half\_data, month\_data\_budget and half\_data\_budget that are 18 months or older
  - inserts 1 row on half\_data for next half for all subclass/location combinations
  - inserts 6 rows on month\_data\_budget for next year for all dept/location combinations
  - inserts 1 row on half\_data\_budget for next year all dept/location combinations and rolls-up this year's HALF\_DATA.shrinkage\_pct at subclass level to HALF\_DATA\_BUDGET.shrinkage\_pct at dept level for next year to create the budget shrinkage percent base for next year. To do this,
    - first sums up inter\_stocktake\_shrink\_amt and inter\_stocktake\_sales\_amt on half\_data from subclass to dept level
    - re-calculate shrinkage\_pct at the dept level = 
$$\left( \frac{\text{sum inter\_stocktake\_shrink\_amt}}{\text{sum inter\_stocktake\_sales\_amt}} \right) * 100$$
    - update half\_data\_budget.shrinkage\_pct for next year using the above calculated shrinkage\_pct

## Stock ledger & budget tables insert [salins]

### Design Overview

This program reads the table stock\_ledger\_inserts and inserts rows into month\_data, week\_data, half\_data, month\_data\_budget, and half\_data\_budget for new locations added to the system since the last run. It also inserts rows into month\_data, week\_data and half\_data for new subclasses and into month\_data\_budget and half\_data\_budget for new departments.

| TABLE                | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------------|-------|--------|--------|--------|--------|
| STOCK_LEDGER_INSERTS | No    | Yes    | No     | No     | Yes    |
| WEEK_DATA            | Yes   | No     | Yes    | No     | No     |
| MONTH_DATA           | Yes   | No     | Yes    | No     | No     |
| HALF_DATA            | Yes   | No     | Yes    | No     | No     |
| MONTH_DATA_BUDGET    | Yes   | No     | Yes    | No     | No     |
| HALF_DATA_BUDGET     | Yes   | No     | Yes    | No     | No     |

### Scheduling Constraints

Processing Cycle: PHASE 0 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

### Restart Recovery

```
select type_code,
 nvl(dept,-1),
 nvl(class,-1),
 nvl(subclass,-1),
 nvl(store,-1),
 nvl(wh,-1)
from stock_ledger_inserts
order by type_code desc;
```

### Program Flow

N/A



## Shared Modules

STKLEDGR\_INSERTS\_SQL.ADD\_LOC: A call is made to this package function when a new store or warehouse is added to the system. This creates the week\_data, month\_data and half\_data records at the location / subclass level.

STKLEDGR\_INSERTS\_SQL.ADD\_SUBCLASS: A call is made to this package function when a new subclass is added to the system. This creates the week\_data, month\_data and half\_data records at the location / subclass level.

STKLEDGR\_INSERTS\_SQL.ADD\_DEPT\_BUDGET: A call is made to this package function for each new department or location that is added to the system. Budget numbers are held at a department / location level and this creates the month\_data\_budget and half\_data\_budget records at the location / department level.

## Function Level Description

This program reads the table stock\_ledger\_inserts and inserts rows into week\_data, month\_data, half\_data, half\_data\_budget and month\_data\_budget.

Detailed description of the functions of this program are as follows :

Set a cursor on stock\_ledger\_inserts to read one row at a time :

if type\_code = 'S'

    insert one row of month\_data for the month to be processed in the next salmth run  
     (ie. the half\_no/month\_no for (system\_variables.last\_eom\_date + 1) )  
     for this store and all subclass combinations

if system\_options.stock\_ledger\_time\_level\_code = 'W'

    insert week\_data for each of the weeks, starting from the first week of  
     the month to be processed in the next salmth run, until the last week processed in  
     the last salweek run for this store and all subclass combinations

end if

(example : if we are in the **second** week of June, and salmth has not run for May yet, we would insert month\_data for month of May, and week\_data for all the weeks in May plus 1st week in June)

insert one row of half\_data for the half with half\_no =  
     system\_variables.last\_eom\_next\_half\_no for this store and all subclass  
     combinations

insert 6 rows of month\_data\_budget for each of the current half and next half for this  
     store and all dept (except dept 9999) combinations (identical to inserting  
     month\_data in mth\_data\_add\_store.sql)

insert 1 row each of half\_data\_budget for the current half and next half for this store  
     and all dept (except dept 9999) combinations

else

if type\_code = 'W'

    insert one row of month\_data for the month to be processed in the next salmth run  
     for this store and all subclass combinations (ie. the half\_no/month\_no for  
     (system\_variables.last\_eom\_date + 1) )

if system\_options.stock\_ledger\_time\_level\_code = 'W'

```

 insert week_data for each of the weeks, starting from the first week of
 the month to be processed in the next salmth run, until the last week processed in the
 last salwka run for this warehouse and all subclass combinations
 end if

 insert one row of half_data for the half with half_no =
 system_variables.last_eom_next_half_no for this warehouse and all subclass
 combinations
 insert 6 rows of month_data_budget for each of the current half and next half for this
 warehouse and all dept (except dept 9999) combinations (identical to inserting
 month_data in mth_data_add_wh.sql)
 insert 1 row each of half_data_budget for the current half and next half for this
 warehouse and all dept (except dept 9999) combinations
else
if type_code = 'B'
 insert one row of month_data for the month to be processed in the next salmth run
 for this subclass and all location combinations (ie. the half_no/month_no for
 (system_variables.last_eom_date + 1))

 if system_options.stock_ledger_time_level_code = 'W'
 insert week_data for each of the weeks, starting from the first week of
 the month to be processed in the next salmth run, until the last
 week processed in the last salwka run for this subclass and all
 locations combinations
 end if

 insert one row of half_data for the half with half_no =
 system_variables.last_eom_next_half_no for this subclass and all location combinations
else
if type_code = 'D'
 insert 6 rows of month_data_budget for the current half and 6 for next half for this
 dept and all location combinations (identical to inserting month_data in
 mth_data_add_dept.sql)
 insert 1 row of half_data_budget for the current half and one for next half for this
 dept and all location combinations

```

Delete all rows on stock\_ledger\_inserts after all rows have been processed

#### I/O Specification

N/A

#### Technical Issues

N/A

## Sales monthly [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:

- a. Whether the retail or cost method of accounting is used. Depending on the setting of DEPS.profit\_calc\_type -- 1 = cost, 2 = retail
- b. 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 retrieved 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 accumulate towards the next stock count.

Note that 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     |

### Stored Procedures / Shared Modules (Maintainability)

**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.

### 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

## Input Specifications

### Driving Cursor

```

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)
FROM month_data,
 v_restart_dept rv
WHERE month_data.half_no = :half_no
 AND month_data.month_no = :month_in_half
 AND rv.driver_value = month_data.dept
 AND rv.driver_name = :ps_restart_driver_name
 AND rv.num_threads = :pi_restart_num_threads
 AND rv.thread_val = :pi_restart_thread_val
 AND (month_data.dept > NVL(:ps_restart_dept,month_data.dept - 1) OR
 (month_data.dept = :ps_restart_dept AND
 (month_data.class > :ps_restart_class OR
 (month_data.class = :ps_restart_class AND
 (month_data.subclass > :ps_restart_subclass OR
 (month_data.subclass = :ps_restart_subclass AND
 (month_data.store > :ps_restart_store)))))))
ORDER BY month_data.dept,
 month_data.class,
 month_data.subclass,
 month_data.store;
```

**Output Specifications****Scheduling Considerations**

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

## Purge stock ledger transactions [salprg]

### Design Overview

This program deletes tran\_data records that are older than the user defined number of retention days, which is stored in system\_options.tran\_data\_retained\_days\_no.

Tables Affected:

| TABLE          | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------|-------|--------|--------|--------|--------|
| PERIOD         | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS | No    | Yes    | No     | No     | No     |
| TRAN_DATA      | No    | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: PHASE AD-HOC (daily)

Scheduling Diagram: Advantageous to run this module prior to phase 3 since this will improve select operations from

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

### Restart Recovery

N/A

### Program Flow

N/A

### Shared Modules

N/A

### Function Level Description

N/A

### I/O Specification

N/A

### Technical Issues

N/A

## Stock ledger stage [salstage]

### Design Overview

In order to make the rollup and extraction of the stock ledger transaction data the most flexible, this program will move the data on the tran\_data to the if\_tran\_data staging table. This will enable the processes that are writing records to tran\_data to continue in a seamless manner, whereas the processes that roll the data up to a different level or extract the data to external systems can work without affecting batch timetables.

This process will be achieved by locking the tran\_data table and moving all of the data to the staging table. The original tran\_data table will be emptied and the lock on the table will be released. Before this processing occurs, the staging table will first be emptied to ensure that data is not processed twice.

The data on the tran\_data and if\_tran\_data tables is very transitional. These tables will fill up and be truncated at least once a day if not several times per day.

### Scheduling Constraints

Processing Cycle: anytime (this will take a snapshot of data on tran\_data)

Scheduling Diagram: N/A

Pre-Processing: inventory processing

Post-Processing: saldly, extractions against if\_tran\_data, salapnd

Threading Scheme: Threading will be implicit via the use of the Oracle Parallel Query Option. The insert/select query should be tuned for each specific environment to achieve the best throughput.

### Restart Recovery

No specific restart/recovery scheme needs to be defined because of the limited scope of the module. However, in cases where the tran\_data table is very large, a particularly large rollback segment may be specified to reduce the risk of exceeding rollback segment space. This will depend on the size of normal rollback segments and the size of the tran\_data table.

### Program Flow

N/A

### Shared Modules

N/A

### Function Level Description

Init():

The rollback segment should be specified to a large enough size so that the entire rollup will complete in a single transaction.



**Process()**

This function first call the drop\_index function to remove the indexes on the if\_tran\_data table. It then calls the truncate\_table function to remove any data in if\_tran\_data so it is not processed twice. Then it calls the lock\_table function to ensure that no changes take place while the move is taking place. Once the lock is placed on the tran\_data table, the build\_stage function is called. After the data is moved to the if\_tran\_data table the truncate\_table function is called to remove the old data from the tran\_data table. Next the indexes are rebuilt in the if\_tran\_data table. Finally the lock is removed from the tran\_data table.

**Lock\_table()**

This function places or removes a lock on a table passed in. It will place or release the lock depending on an input parameter.

**Build\_stage()**

This function builds the if\_tran\_data by moving all of the data from the tran\_data table onto the if\_tran\_data table that has a tran\_date <= the vdate. This is done using an insert/select statement with a hint for parallelism. The degree of parallelism should be customized to each run-time environment. After the data is moved the degree of parallelism is set to zero.

**Modify\_indexes()**

This function rebuilds the indexes on the if\_tran\_data table. The indexes are built in parallel, unreciverable. After the indexes are built their degree of parallelism is set to zero.

**Truncate\_table()**

This function truncates the table that is passed in as an input parameter using dynamic SQL.

**Build\_temp()**

This function is a duplicate of the build\_stage() function other than that it moves data off the tran\_data table to the tran\_data\_temp table that has a tran\_date > the vdate.

**Rebuild\_tran\_data()**

This function takes data from tran\_data\_temp that was put there by the build\_temp() function and moves it back on the tran\_data table after it has been truncated.

**I/O Specification**

N/A

**Technical Issues**

This program requires special permissions. It must be run by an oracle user that has been granted the following privileges, or be run by a DBA.

- 'drop any table' AND
- 'alter any index' AND
- 'lock any table'

## Sales weekly [salweek]

### Design Overview

This program processes all weeks that are in the month for which month-end process has not been run, up to the current week. It rolls up data on DAILY\_DATA to the corresponding subclass/location/week record on WEEK\_DATA. The closing stock value and gross margin are calculated by calling the appropriate package function, based on the accounting method (i.e. profit calculation type) chosen for the department. Closing stock value for a processed week becomes opening stock value for the next week. A WEEK\_DATA row for next week is inserted if it does not already exist.

If a stock count occurs during a certain week, stkdly.pc would have already updated that week's week\_data.stocktake\_adj\_cost and retail fields with the difference between book stock value and actual stock value (results of stock count). This stocktake\_adj\_cost and retail are used to adjust weekly closing stock value in this program. The closing stock value for all the weeks that follow will also be adjusted, due to the fact that this week's closing stock will be rolled into next week's opening stock.

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

If budget shrinkage indicator = 'Y' :

shrinkage amount = budgeted shrinkage percent \* sales amount  
for each week processed

else

shrinkage amount = - (stock\_adj\_cost or retail)  
depending on cost or retail method is used

The above calculated shrinkage amount will then be used to reduce the closing stock for each week processed.

This program not only runs every week-end, it must also run immediately before salmth runs, since salmth could be run any day during the week. This is to update week\_data with tran\_data record that comes in between the last time salweek runs and when salmth runs. The only difference between these two salweek runs is that : a WEEK\_DATA row for next week is inserted only if this program runs on week-end.

| TABLE            | INDEX | SELECT | INSERT | UPDATE | DELETE |
|------------------|-------|--------|--------|--------|--------|
| DAILY_DATA       | Yes   | Yes    | No     | No     | No     |
| DEPS             | No    | Yes    | No     | No     | No     |
| HALF_DATA_BUDGET | No    | Yes    | No     | No     | No     |
| PERIOD           | No    | Yes    | No     | No     | No     |
| CALENDAR         | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS   | No    | Yes    | No     | No     | No     |
| SYSTEM_VARIABLES | No    | Yes    | No     | Yes    | No     |
| WEEK_DATA        | No    | Yes    | Yes    | Yes    | No     |

**Stored Procedures / Shared Modules (Maintainability)**

**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.

454 Calendar functions

**Function Level Description**

Main

Init

- Initialize restart variables.
- Collect system date information
- Get\_weeks\_to\_process function callout to fill array with week information since last eom
- Get\_half\_info function callout to determine if data will include more than 1 454 half
- Size\_location function callout to determine the size for the location array (#loc \* # halves) to hold budgeted shrinkage
- Load\_dept function callout to size and fill department array with profit calc and purchase type info
- Size\_arrays function callout to size all other arrays according to value from restart API
- Setup\_daily\_zeros\_array function callout to create a single dimensional array filled with zeros for weeks when no daily transactions exist on daily\_data

Get\_half\_info

- if the start of the current half (the half belonging to the system date) is later than the last end of month date, then more than one half has been spanned.
- Determine value of last half to return to calling function.

Get\_weeks\_to\_process

- Size\_weeks function callout to size the week array appropriately
- select date information from the 1<sup>st</sup> week to process (1<sup>st</sup> week after last end of month) to the current end of week date
- cal\_to\_454\_half function callout to gather 454 calendar information about each date.
- Populate each row of the week array with each week's information

Cal\_to\_454\_half

- CAL\_TO\_454\_HALF PL/SQL procedure callout for centralized function

Size\_week

- use the number of weeks value from the init function to appropriately size the array that will hold weekly calendar information from the last end of month date to the current end of week date.

Setup\_daily\_zeros\_array

- apply values of zero to a one-dimensional array that will be used to recalculate weekly records when no daily records exist.

Process

- The data from week\_data and daily\_data should be scrolled through to evaluate whether daily stock ledger records exist for each week record. Week records should always exist for each department, class, subclass, store, warehouse, currency indicator and end of week category, however daily records may not exist. If daily records exist or if stock take adjustments have been recorded (on weekly records) then the weekly stock ledger data should be reevaluated and updated for that week and all subsequent weeks.
- open\_cursors function callout to open daily and weekly cursors.
- Weekly data will be selected from the week\_data table to drive the program's processing. All weekly data will be selected that has an end of week date that is after the last closed end of month date. Data will be grouped and ordered by department, class, subclass, store, warehouse, currency indicator, and end of week date.
- Daily data will be selected from the daily\_data table to match the data selected from the week\_data cursor. All daily\_data will be selected that has an end of week date that is after the last closed end of month date. Data will be grouped and ordered by department, class, subclass, store, warehouse, currency indicator, and end of week date.
- Fetch\_week\_cursor function callout to handle the fetching of week data into arrays for processing.
- Fetch\_day\_cursor function callout to handle the fetching of day data into arrays for processing.
- First\_week\_handler function callout to evaluate and process the first week\_data record for a stock ledger category (dept, class, subclass, store, warehouse, currency indicator, and end of week date).
- Day\_handler function callout will evaluate fetched records and process them.
- Last\_week\_handler function callout will process the last records in the stock ledger category and insert a new record for the next week.
- Post\_arrays function callout will post the filled and cached insert and update arrays to the database.

#### Open\_cursors

- open the week\_data cursor.
- Open the daily\_data cursor.

#### Fetch\_week\_cursor

- fetch week\_data cursor into arrays.
- Because daily\_data and week\_data records may not match up exactly (daily\_data records may not exist for each week), the daily and week data should be fetched separately.

#### Fetch\_day\_cursor

- fetch daily\_data cursor into arrays.
- Because daily\_data and week\_data records may not match up exactly (daily\_data records may not exist for each week), the daily and week data should be fetched separately.

#### First\_week\_handler

- cache beginning of week data, including opening stock position. The stock position for the first week of the stock ledger category will be correct because of the closing of the previous month.
- Get\_dept\_info function callout to fill department array with profit calculation type and purchase type information.
- Set\_start\_values function callout to populate facts for first week in stock ledger category
- Cache stock ledger category information (dept, class, subclass, store, wh and currency indicator) for use in processing.

- Evaluate whether items should be posted to the database from the arrays and the data committed. The criteria will be based on whether the restart commit counter (of the restart/recovery API) has been reached or exceeded. Further, because this is the first week of a stock ledger category, all of the information in the arrays will be of complete stock ledger categories.
- Post\_arrays function callout will post the cached array data to the database.
- Restart\_commit function callout will commit the data that has been thus far posted to the database.

#### Set\_start\_values

- initialize first values of information that will be re-calculate and accumulated via packages for the stock ledger category.
- Cls\_stk\_cost
- Htd\_gafs\_cost
- Retail shrinkage
- Cost shrinkage
- Cumulative markon percent
- Gross margin amount

#### day\_handler

- This function will be called after the first week of the stock ledger category week\_data information has been evaluated.
- Spool through daily and week arrays until all weeks of the stock ledger category have been processed.
- If no daily records exist for the stock ledger category exist, then evaluate the rest of the weekly records to determine if stock take adjustments have been applied to any of the weeks. If adjustments exist for any of the weeks, then the stock ledger information will have to be recalculated and updated (week\_data) for that week and all subsequent weeks in the category. Values of zero will be passed for what would have been fetched from the daily\_data information.
- If any daily\_data records exist for any weeks, then that week and all subsequent weeks will be processed. The cached and cumulative data will be rolled up across the weeks, the daily\_data information (zeros for weeks without any daily records), and the stock take adjustments will be used to recalculate the stock ledger.
- Set\_up\_daily\_values function callout will populate the variables that will be used in the package call for stock ledger recalculations. Either the current fetched information or zeros will be passed as parameters depending on whether daily data exists or not for the current week.
- increment\_week\_data\_week function callout
- fetch\_week\_cursor function callout to fetch new batch of data from the week\_data table
- make\_fnc\_calls function callout to call the stock ledger calculation packages that recalculate the stock information for the week based on the daily info, stock take adjustments, and the opening stock info.
- Fill\_update\_array function callout will populate the week\_data update array with the new values for the current week
- If the weeks being evaluated span 454 halves, then reset the half to date stock variables (gafs cost & retail) for the new half.

#### Setup\_daily\_values

- either daily\_data values or zeros will be passed in as parameters to this function.

- The passed data will be copied into holder variables to be used in stock ledger function calls and for rolling accumulations for the stock ledger category.

#### Make\_fnc\_calls

- get\_location\_info function callout will get the appropriate shrinkage rate for the current department, location, and half.
- For non-consignment items, the daily\_data values (or zeros) will be passed with the shrinkage rate to the stock ledger calculation package
- Cost\_package\_call function callout will be called if cost method of accounting is used.
- Retail\_package\_call function callout will be called if retail method of accounting is used.
- For consignment items, the stock ledger packages will not be called, but the gross margin amount will be set for the update (purchase retail – purchase cost).

#### Fill\_update\_array

- daily\_data info, calculated stock ledger positions, and open/closing stock position information is copied into the week\_data update array
- for non-consignment items, only the following items will be updated
- open stock retail
- open stock cost
- purchase costs
- purchase retail
- net sales cost
- net sales retail (includes vat)
- net sales retail excluding vat
- sales units
- gross margin amount
- for consignment items all data will be updated.
- For profit calculations of type direct cost, half to date retail, gafs cost and cumulative markon percent will be updated as null
- Post\_update function callout will post the array to the database if it is full.

#### Retail\_package\_call

- STKLEDGR\_ACCTING\_SQL.RETAIL\_METHOD\_CAL function callout will (via a PL/SQL package) use the retail method of accounting to calculate the stock ledger values for the current week's data.
- The accumulated values will be updated for the next week in the stock ledger category
- Cls stk retail
- Cls stk cost
- Half to date gafs retail
- Half to date gafs cost
- Gross margin amount
- Cumulative markon percent
- Shrinkage retail

#### Cost\_package\_call

- STKLEDGR\_ACCTING\_SQL.COST\_METHOD\_CAL function callout will (via a PL/SQL package) use the cost method of accounting to calculate the stock ledger values for the current week's data.
- The accumulated values will be updated for the next week in the stock ledger category
- Cls stk retail

- Cls stk cost
- Gross margin amount
- Shrinkage cost

#### Last\_week\_handler

- last\_ins\_update function callout will populate either insert or update arrays for the last week in the process.
- Progress the week\_data to the first week of the next stock ledger category.
- Fetch\_week\_cursor function callout will fetch new batch of week\_data data if all of the elements of the current fetch have been processed.

#### Last\_ins\_update

- if the last week of the week\_data information from the current stock ledger category is equal to the maximum process date, then populate the update array with the current week's data, otherwise populate the week\_data insert array with the current week's data.
- Fill\_last\_week\_upd function callout will populate the week\_data update array
- Populate\_insert function callout will populate the week\_data insert array

#### Populate\_insert

- get the 454 information for the last week
- CAL\_TO\_454 function callout will calculate the 454 calendar information for the passed date
- For all item types, copy the following fetched and calculated information into the insert array:
- Dept, class, subclass, store, warehouse, currency indicator, end of week date
- Half number, month number
- For non-consignment items with a normal purchase type copy the following additional information:
- Half to date gafs retail as calculated
- Half to date gafs cost as calculated
- For non-consignment items with a direct cost purchase type, copy the following additional information:
- Half to date gafs retail as null
- Half to date gafs cost as calculated
- For consignment items, copy the following additional information:
- Opening stock retail as zero
- Opening stock cost as null
- Half to date gafs retail as null
- Half to date gafs cost as null
- Post\_insert function callout will post the arrays to the database if they are full

#### Fill\_last\_week\_upd

- for normal purchase types
- for retail profit calculation type
- half to date gafs retail will equal the calculated value
- half to date gafs cost will equal the calculated value
- for cost profit calculation type
- half to date gafs retail will be null
- half to date gafs cost will be null
- for consignment purchases
- half to date gafs retail and cost, open stock retail, and open stock cost will all be null
- post\_last function callout to post update array to the database if the array is full

**post\_arrays**

- post\_insert function callout to post array insert to week\_data to database
- post\_update function to post array update of week\_data to database
- post\_last function to post array update of the last week of week\_data info to the database

**post\_insert**

- embedded SQL call to perform bulk insert of array to week\_data

**post\_update**

- embedded SQL call to perform bulk update of array to week\_data

**post\_last**

- embedded SQL call to perform bulk update of array to week\_data

**size\_arrays**

- allocate system memory for arrays in use
- the number of rows of each array should be determined by the counter (commit\_max\_ctr) on the restart\_control table. The value will be obtained in the init via the function call to restart\_init
- the zero array will be sized to have only one row.

**Size\_location**

- this function counts all of the stores and warehouses in the system and sizes the array to hold shrinkage info at the location/half/department level.
- If more than one half will be spanned by processing then, then the size will be twice the number of locations, otherwise it will be equal to the number of locations.
- If more than two halves are spanned then a fatal error will occur. This means that the salweek program hasn't been run for more than a year.
- Allocate memory for the array elements using the sizing method described above.

**Load\_dept**

- this function loads the profit\_calc\_type and purchase type for every dept in the system into an array that will be used by the main processing as it moves through the week\_data.
- The size of the array will be determined by the total number of departments in the system.
- The number of rows in the array is limited to 32K. if more than 32K departments exist, the code will have to be modified to handle this.
- Size\_deps function callout will physically allocate the memory necessary to accommodate the array.
- The array is filled with department, profit calculation type, and purchase type ordered by department.

**Size\_deps**

- this function physically allocates memory for the department array based on the number of departments in the system.

**Get\_dept\_info**

- this function will be called to get the department level info for the stock ledger category.



- The array will be searched for the appropriate department and the profit calculation and purchase type will be obtained.

#### Load\_location

- This function loads the shrinkage rate for each location in a dept for a given half.
- When the department changes, the location/half arrays will be filled with all of the shrinkage and location info.
- An array will be used for each location type (store and warehouse).

#### Get\_location\_info

- this function steps through the location array(s) to get the shrinkage info for the location/half for each department.
- Load\_location function callout will be called to fill the location array(s) if the department has changed.

### Input Specifications

#### Driving Cursor:

```

SELECT wd.dept,
 wd.class,
 wd.subclass,
 wd.store,
 wd.wh,
 wd.half_no,
 wd.week_no,
 wd.currency_ind,
 TO_CHAR(wd.eow_date, 'YYYYMMDD'),
 TO_NUMBER(TO_CHAR(wd.eow_date+7, 'DD')),
 TO_NUMBER(TO_CHAR(wd.eow_date+7, 'MM')),
 TO_NUMBER(TO_CHAR(wd.eow_date+7, 'YYYY')),
 TO_CHAR(wd.eow_date+7, 'YYYYMMDD'),
 NVL(wd.stocktake_adj_retail,0),
 NVL(wd.stocktake_adj_cost,'0'),
 NVL(wd.opn_stk_retail,0),
 NVL(wd.opn_stk_cost,'0'),
 DECODE(wd.currency_ind,
 'L', NVL(wd.htd_gafs_retail,0) - (NVL(wd.purch_retail,0)
 - NVL(wd.rtv_retail,0) +
 + NVL(wd.markup_retail,0)
 - NVL(wd.markup_can_retail,0)
 + NVL(wd.tsf_in_retail,0)
 - NVL(wd.tsf_out_retail,0)
 + NVL(wd.tsf_in_book_retail,0)
 - NVL(wd.tsf_out_book_retail,0)
 + NVL(wd.reclass_in_retail,0)
 - NVL(wd.reclass_out_retail,0)),
 NVL(wd.htd_gafs_retail,0)),
 DECODE(wd.currency_ind,
 'L', NVL(wd.htd_gafs_cost,0) - (NVL(wd.purch_cost,0)
 - NVL(wd.rtv_cost,0)
 + NVL(wd.freight_cost,0)
 + NVL(wd.tsf_in_cost,0)
 - NVL(wd.tsf_out_cost,0)
 + NVL(wd.tsf_in_book_cost,0)
 - NVL(wd.tsf_out_book_cost,0)
 + NVL(wd.up_chrg_amt_profit,0)
 + NVL(wd.up_chrg_amt_exp,0)
 + NVL(wd.reclass_in_cost,0)
 - NVL(wd.reclass_out_cost,0)),
 NVL(wd.htd_gafs_cost,0)),
 ROWIDTOCHAR(wd.rowid)
FROM v_restart_dept rv,

```

```

 week_data wd
WHERE wd.eow_date > to_date(:os_last_eom_date,'YYYYMMDD')
AND wd.eow_date <= to_date(:os_fetch_upto_date,'YYYYMMDD')
AND rv.driver_value = wd.dept
AND rv.driver_name = 'DEPT'
AND rv.num_threads = :ora_restart_num_threads
AND rv.thread_val = :ora_restart_thread_val
AND (wd.dept > NVL(:os_restart_dept, -999) OR
 (wd.dept = :os_restart_dept AND
 (wd.class > :os_restart_class OR
 (wd.class = :os_restart_class AND
 (wd.subclass > :os_restart_subclass))))))
ORDER BY wd.dept,
 wd.class,
 wd.subclass,
 wd.store,
 wd.wh,
 wd.currency_ind,
 wd.eow_date;

```

### Scheduling Considerations

Processing Cycle: Stock Ledger

Scheduling Diagram: Can be run anytime after the week has completed.

Pre-Processing: Saldly

Post-Processing: Salmth (if applicable)

Threading Scheme: Department via v\_restart\_dept view

### Restart/Recovery

Restart is on dept, class, and subclass. Clients with substantial number of stores may consider including store and warehouse in the restart logic.

### Performance Considerations

Due to the transaction volumes that this program must process, array processing has been heavily utilized in making Oracle system calls.

The following describe the arrays created for Oracle calls

- Daily\_data, select array
- Week\_data, select, update and insert arrays

Additional structures of arrays:

- Week and 454 calendar information (for weeks from last end of month to today)
- department level array for searching for profit calc type and purchase type information
- location/half cache for searching for budgeted shrinkage information

### Design Assumptions

The base RMS package will recalculate weekly stock ledger (week\_data) records if daily stock ledger records (daily\_data) exist or if weekly records have been updated for stock take adjustments. If clients have customizations or interfaces built that update other weekly record information, then additional logic will have to be added to the program to handle the update.

## Sales audit pre-export [sapreexp]

### Design Overview

When a user modifies or revises a transaction through the Sales Audit user application, numerous totals will be affected through re-totaling. The sales audit pre-export module is designed to compare the latest prioritized version of each total defined for export with the version that was previously sent to each system. If they are the same, an SA\_EXPORTED entry should be created for the total for that particular system so that the same value will not be exported twice. By determining which totals have not changed since the last export date time (SA\_EXPORTED\_REV), this module will then create entries on SA\_EXPORTED to prohibit any third party application from receiving multiple export revisions.

| Table            | Operations Performed |        |        |        |
|------------------|----------------------|--------|--------|--------|
|                  | Select               | Insert | Update | Delete |
| Sa_store_day     | Yes                  | No     | No     | No     |
| Sa_export_log    | Yes                  | No     | No     | No     |
| Sa_exported      | Yes                  | Yes    | No     | No     |
| V_sa_total_value | Yes                  | No     | No     | No     |
| Sa_exported_rev  | Yes                  | No     | No     | No     |

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA auditing process and before any export processes.

Threading Scheme: N/A

### Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

Driving Cursor

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of g\_l\_restart\_max\_counter. Only two commits will be done. One to establish the store/day lock (this will be done by the package) and one at the end after a store/day or store/day/total has been completely processed.

#### Driving cursor:

```
SELECT DISTINCT a.store_day_seq_no
 FROM sa_export_log a
 WHERE a.status = 'R'
 AND a.seq_no =
 (SELECT MAX(b.seq_no)
```

```

FROM sa_export_log b
WHERE b.store_day_seq_no = a.store_day_seq_no)
ORDER BY a.store_day_seq_no;

```

**Work cursors:**

There are two work cursors. The first work cursor will get totals that were previously exported. It is used in the **fetch\_already\_exported()** function.

```

EXEC SQL DECLARE c_already_exported CURSOR FOR
/* select all totals that have been exported already */
SELECT v.total_seq_no,
 v.value * :pl_multiplier,
 el.system_code
FROM v_sa_get_total v,
 sa_total_usage tu,
 sa_export_log el,
 sa_exported_rev er
WHERE v.store_day_seq_no = :is_store_day_seq_no
 AND el.store_day_seq_no = v.store_day_seq_no
 AND el.status = :SAES_R
 AND el.seq_no > 1
 AND tu.total_id = v.total_id
 AND tu.total_rev_no = v.total_rev_no
 AND substr(tu.usage_type, 1, 3) = el.system_code
 AND er.total_seq_no = v.total_seq_no
 AND substr(er.system_code, 1, 3) = el.system_code
 AND er.rev_no = v.value_rev_no
 AND er.exp_datetime =
/* select the last one exported */
 (SELECT MAX(exp_datetime)
 FROM sa_exported_rev
 WHERE total_seq_no = v.total_seq_no
 AND substr(system_code, 1, 3) = el.system_code)
ORDER BY 1, 2, 3;

```

The second work cursor retrieves all the totals that are ready for export.

```

EXEC SQL DECLARE c_ready_to_export CURSOR FOR
/* select all totals ready for export */
SELECT v.total_seq_no,
 v.value * :pl_multiplier,
 el.system_code
FROM v_sa_get_total v,
 sa_total_usage tu,
 sa_export_log el
WHERE v.store_day_seq_no = :is_store_day_seq_no
 AND v.precedence =
 (SELECT min(v2.precedence)
 FROM v_sa_get_total v2
 WHERE v2.total_seq_no = v.total_seq_no)
 AND v.value_rev_no =
 (SELECT max(v2.value_rev_no)
 FROM v_sa_get_total v2
 WHERE v2.total_seq_no = v.total_seq_no
 AND v2.precedence = v.precedence)
 AND el.store_day_seq_no = v.store_day_seq_no
 AND el.status = :SAES_R
 AND el.seq_no > 1

```

```

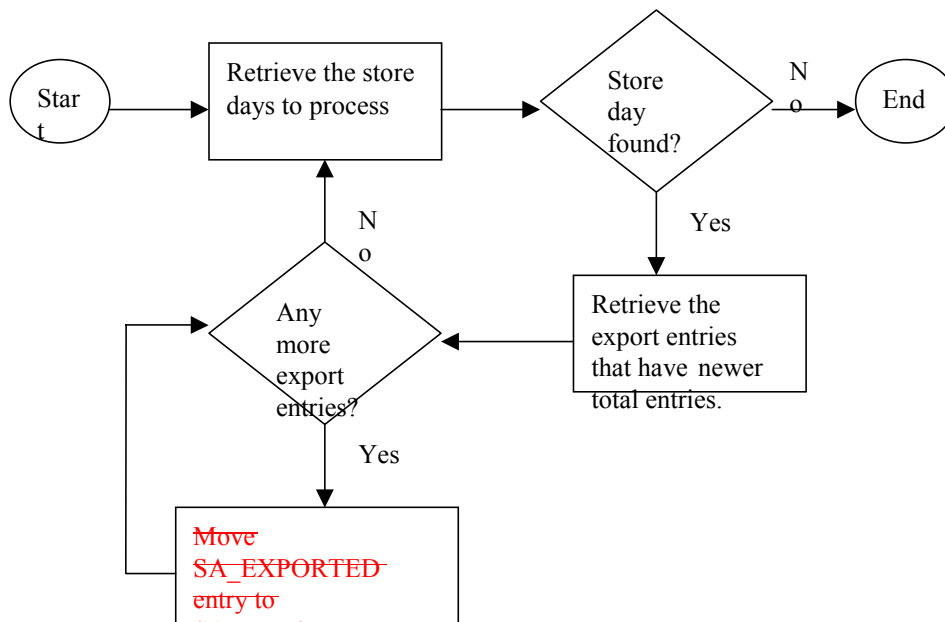
AND tu.total_id = v.total_id
AND tu.total_rev_no = v.total_rev_no
AND substr(tu.usage_type, 1, 3) = el.system_code
AND NOT EXISTS
 (SELECT 1
 FROM sa_exported
 WHERE total_seq_no = v.total_seq_no
 AND substr(system_code, 1, 3) = el.system_code);

```

The program then compares the values retrieved by these two cursors and if a match is found, then an entry in the sa\_exported table is created to prevent this total value from being re-exported.

### Program Flow

Structure Chart



### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

#### ReSA Batch Library functions used:

**get\_lock** – used to establish a read lock on a store/day.

**release\_lock** – used to release a store/day lock.

#### Function Level Description

All database interactions required and error handling considerations

##### init()

- Call **init\_exported\_array**.
- Call **retek\_init**

##### process()

- The Driving Cursor should retrieve the first store day to process.

- Attempt to lock the store/day with a call to **get\_lock** library function. If this fails, go on to the next store/day.
- Call the procedure **process\_store\_day\_exported**.
- Release lock and commit store day work completed.
- Loop through driving cursor.

#### **final()**

- Call **free\_exported\_array**.
- Call **retek\_close** for a final commit.
- Call **retek\_refresh\_thread** to refresh the current thread.

#### **Process\_store\_day\_exported()**

- Call **fetch\_already\_exported()** to retrieve the first work cursor.
- Retrieve the working cursor for the current store day.
- Find if the current row from the second work cursor exists in the data fetched by the first work cursor by calling **already\_exported\_lookup()**.
- If it does exist, insert into the **sa\_exported** table using the values from the **sa\_exported\_rev** table for the **total\_seq\_no** and **system\_code**. Else do nothing.

#### **fetch\_already\_exported()**

- This function uses a global variable to hold all of the data fetched from the first work cursor. The array holding this data is resized as necessary.

#### **already\_exported\_lookup()**

- This function uses the C library function **bsearch** to search the array populated by **fetch\_already\_exported()** for data that matches the parameter passed in. The actual comparison is performed in **nodecmp()**.

#### **nodecmp()**

- This function concatenates the two **pa\_export\_data** rows passed in as parameter and compares them using **strcmp**.

#### **init\_exported\_array()**

- Allocate memory for **il\_size** items for the working cursor arrays.

#### **Free\_exported\_array()**

- Free the working cursor arrays from memory.

## Sales audit purge [sapurge]

### Design Overview

This program will be run daily to control the size of the tables in the sales audit database. Older information will be deleted to ensure optimal performance of the system as a whole.

Different kinds of data need to be kept in the system for different amounts of time. Transactions, and all associated transaction details, and Totals calculated or reported for a store day will be deleted when they meet the following criteria:

- The Business Date for those transactions and totals is older than or equal to today's date minus the `days_before_purge` parameter setup on the `sa_system_options` table.
- No locks exist on the store/day.
- One of the two following statements is true for the store/day:
  1. Fully loaded, and all errors either corrected or overridden (`sa_store_day.audit_status` is 'A' (Audited) and `sa_store_day.data_status` = 'F' (Fully loaded)). In addition, there are no outstanding exports (records for the store/day in the `sa_export_log` table where `sa_export_log.status` = 'R' (Ready for export)).
  2. Never loaded (`sa_store_day.audit_status` is 'U' (Unaudited) and `sa_store_day.data_status` = 'R' (Ready for import)).

Flash Sales data will be deleted when it meets the following criteria:

- Date is two years before today's date minus the `days_before_purge` parameter setup on the `sa_system_options`.
- Company open and close dates will also need to be kept for two years plus `days_before_purge` so that the historical comparisons in flash sales reporting carry the appropriate weight.

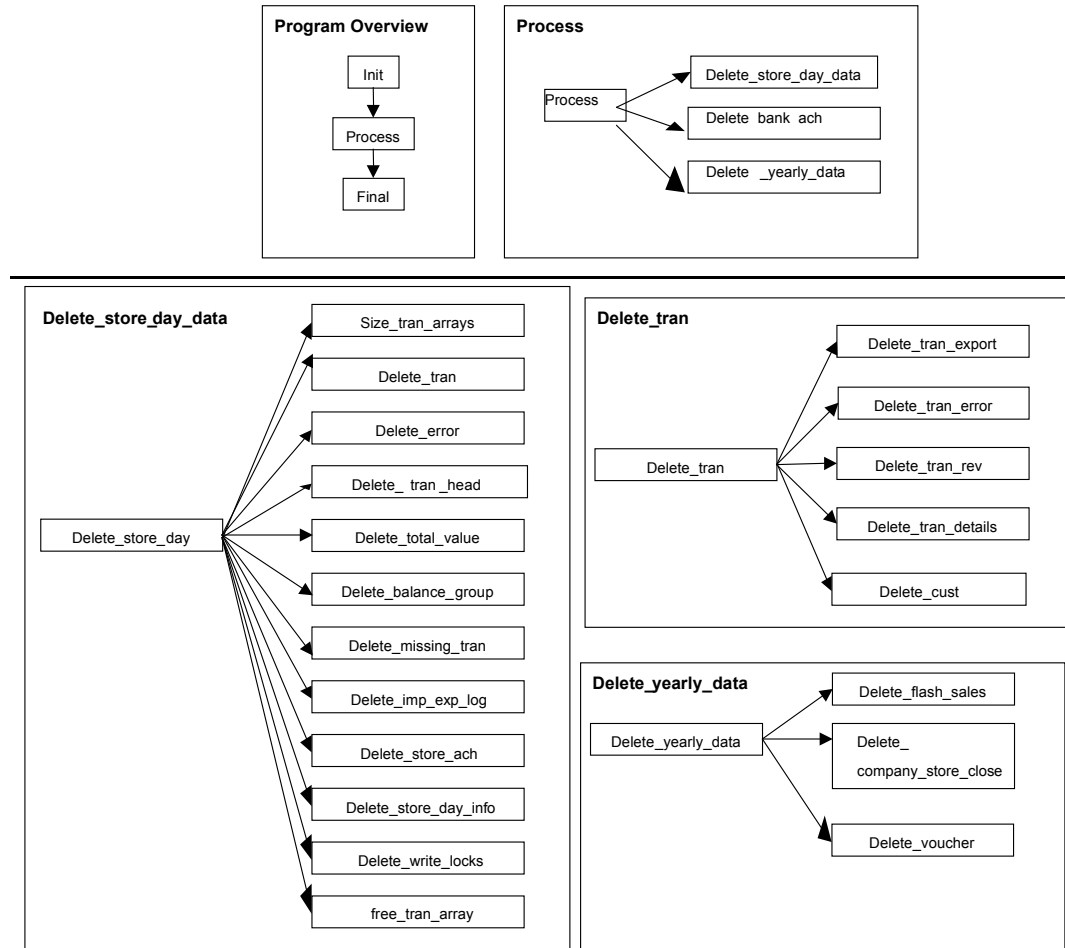
Voucher data will be deleted when it meets the following criteria:

- The redeemed date or the escheat date for the specific voucher type is before today's date minus the `purge_no_days` on the `sa_voucher_options` table for the corresponding voucher type.

The program can also take in a list of `store_day_seq_no` to delete. For example, the command line could be: `sapurge userid/passwd 1000 1001 1002`, where 1000, 1001 and 1003 are `store_day_seq_nos` that the user wants to delete. These must also meet the criteria defined above. If a `store_day_seq_no` is passed to this program, but does not meet the criteria, an error will be written out to the error log.

An output file will be created to store a record for each store and business date that was purged. The file name must be passed in at the command line as a parameter to `sapurge`.

## sapurge.pc



### Stored Procedures / Shared Modules (Maintainability)

Libretek.a library functions:

Retek\_init/retek\_close – to initialize and close Retek's restart/recovery module

FetchVdate – to get the current date and time

FetchSaSystemOptions – to get days before purge value from ReSA system options table.

### Input Specifications

#### 'Table-To-Table'

The list of tables from which data is going to be deleted are:

sa\_store\_day  
 sa\_tran\_head  
 sa\_tran\_tax  
 sa\_tran\_item  
 sa\_tran\_disc  
 sa\_tran\_tender



```

sa_exported
sa_tran_head_rev
sa_tran_tax_rev
sa_tran_item_rev
sa_tran_disc_rev
sa_tran_tender_rev
sa_exported_rev
sa_customer
sa_cust_attrib
sa_balance_group
sa_total
sa_pos_value
sa_sys_value
sa_store_value
sa_hq_value
sa_total_value_rev
sa_error
sa_error_rev
sa_store_error
sa_missing_tran
sa_import_log
sa_export_log
sa_store_ach
sa_bank_ach
sa_flash_sales
sa_company_close
sa_company_excep
sa_store_close
sa_voucher

```

## Output Specifications

### Function Level Description

Init – This function will fetch the `system_options.days_before_purge` and the `period.vdate` into global variables to be used throughout the program. This function will also call `restart_init` to begin the restart/recovery bookmark process. The file to store purged store/days will need to be opened.

If a list of `store_days` are passed at the command line, then it will copy them into an array of the appropriate size.

CopySDArray – This function is called from `Init` and copies the list of `store_days` passed at the command line into an array.

Process – This function will call the function `process_delete_store_day_data`, and if no `store_day_seq_nos` are passed at the command line, it will also call the function `delete_yearly_data`, each of which will deal with records from different ‘time’ periods. `process_delete_store_day_data` deletes all associated detail records that correspond to a single store/day. `delete_yearly_data` deletes records for which two years of history is needed.

process\_delete\_store\_day\_data – This function will call functions to delete information that grows by store/day.

If a list of store\_day\_seq\_nos are passed at the command line, then only these store\_days will be passed in to the function to determine which store\_days to delete, otherwise all will be considered for deletion.

The driving cursor in this function will select the outdated sa\_store\_day.store\_day\_seq\_no. A sa\_store\_day.store\_day\_seq\_no is outdated when it's sa\_store\_day.business\_date is equal to the period.vdate – sa\_system\_options.days\_before\_purge.

Also, the program will only delete those store/days without errors.

Before any purging occurs, the function should set sa\_store\_day.data\_status to 'G' for Purging. This lets the user know that he/she should not be locking the data.

This function must also try to obtain a write lock by using the get\_lock() function in the sales audit library. If this function returns a 1, then the store/day is locked. The store/days that meet this criteria should be fetched into an array. If a 0 is returned by get\_lock(), then a lock on this store/day is obtained. This program will size this array by calling the function size\_tran\_arrays. This function will pass a store/day at a time to the functions Delete\_tran, Delete\_cust, Delete\_balance\_group, Delete\_total\_value and Delete\_missing\_tran to delete transactions, customers, balance groups, errors, totals and missing transactions respectively. As a processing completes for a store/day, the Store and Business Date should be written out to the purged file. If errors caused the store/day to be only partially completed, this should be noted in this log as well. If no store/days at all were purged, this message should be written out to this file.

Note: When a commit or a rollback occurs, the lock is lost and must be reobtained by calling get\_lock(). Due to the limited size of the rollback segment, we commit at the end of each of the functions below. Thus get\_lock() must be called after the commit statement.

Size\_tran\_arrays – This function will size the arrays needed for the deletes.

Delete\_tran – This function will call several functions to delete all transaction level information from the database. This function will pass the array of store/days it receives to the following functions: Delete\_tran\_export, Delete\_tran\_rev, Delete\_tran\_detail and Delete\_tran\_head.

Delete\_tran\_export – This function will delete from the sa\_exported and sa\_exported\_rev for the transactions with expired store/days.

Delete\_tran\_error – This function will delete from the sa\_error\_rev for the transactions with expired store/days.

Delete\_tran\_rev – This function will delete from the sa\_tran\_tax\_rev, sa\_tran\_disc\_rev, sa\_tran\_item\_rev, sa\_tran\_tender\_rev, and sa\_tran\_head\_rev tables for transactions with expired store/days.

Delete\_tran\_detail – This function will delete from the sa\_tran\_tax, sa\_tran\_disc, sa\_tran\_item and sa\_tran\_tender tables for transactions with expired store/days.

Delete\_cust – This function will delete from the sa\_cust\_attrib and sa\_customer tables all records related to transactions with expired store/days.

Delete tran comments – This function will delete records from the sa\_comments table that are related to transactions with expired store/days. The comment\_type will be 'SATR' for the transaction-level comments.

Delete\_error – This function will delete from the sa\_error and sa\_store\_error tables for expired store/days.

Delete\_tran\_head – This function will delete from the sa\_tran\_head table for transactions with expired store/days.

Delete\_total\_value – This function will delete from the sa\_total, sa\_pos\_value, sa\_pos\_value\_wksht, sa\_sys\_value, sa\_sys\_value\_wksht, sa\_store\_value, sa\_hq\_value, sa\_error\_wksht, sa\_comments where comment\_type is 'SATL' and sa\_total\_value\_rev tables for expired store/days.

Delete\_balance\_group – This function will delete from the sa\_comments where comment\_type is 'SABG' and sa\_balance\_group table for expired store/days.

Delete\_missing\_tran – This function will delete from the sa\_missing\_tran table for expired store/days.

Delete\_imp\_exp\_log – This function will delete from the sa\_import\_log and sa\_export\_log tables.

Delete\_store\_bank\_ach – This function will delete from the sa\_store\_ach and sa\_bank\_ach tables.

Delete\_store\_day\_comments – This function will delete records from the sa\_comments table that are related to expired store/days. The comment\_type will be 'SASD' for the transaction-level comments.

Delete\_escheat\_totals – This function will delete from the sa\_escheat\_total and sa\_escheat\_voucher tables for expired store/days.

Delete\_store\_day\_info – This function will delete from the sa\_store\_day table. We do not attempt to get a lock at the end of this function.

Delete\_write\_locks – This function will delete from the sa\_store\_day\_write\_lock table. We do not attempt to get a lock at the end of this function.

Free\_tran\_array – This function will free up the dynamically allocated memory from size\_tran\_array.

Delete\_yearly\_data – Three types of information are deleted in this function : flash sales information, reference information about company and store close dates and vouchers. Since the next 3 functions are not data-dependent, different threads can call them.

Delete\_flash\_sales – This function will delete records from the sa\_flash\_sales table for days that are (two years plus days\_before\_purge) older than the period.vdate.

Delete voucher – This function will delete from the sa\_voucher table. Voucher deletes are based on the sa\_voucher.red\_date or the escheat date. If the voucher has been redeemed, escheated or taken as an income adjustment, it should be deleted when the the redeemed date or the escheated date is earlier than the period.vdate – sa\_voucher\_options.purge\_no\_days for the corresponding voucher type (tender\_type\_id). If the voucher is still in Assigned or Issued status, it has not yet been redeemed or escheated, and therefore it should not be deleted.

Final – This function should call restart\_close to appropriately bookmark the program's status, close the purged file and should free any memory used.

### **Scheduling Considerations**

This program should be run as the last program in the batch schedule. It can be run as part of the daily or monthly ReSA schedules.

### **Locking Strategy**

### **Restart/Recovery**

Restart/recovery is implicit in purge programs. The program only needs to be run again to restart appropriately. However, simple restart/recovery should be implemented to bookmark the status of the program.

### **Performance Considerations**

Array Processing: As noted in the function level descriptions, as much processing as possible is done in arrays. Each driving cursor should fetch into an array, and then the same array should be used to delete from the database.

Row\_id: Row\_id should be selected in the driving cursors wherever possible and used during deletion as it is considerably more efficient than deleting by primary key.

Threading: This program will be threaded by store. Many of the tables to be deleted from have a store column, making the threading simple. Other tables do not have store, but instead have store\_day\_seq\_no. To thread by store, the store associated with a store\_day\_seq\_no will be used.

Performance: Performance is of the utmost importance to this program. In a production environment, this program may have to delete upwards of 20 million records a day. Performance should be tested against a data set of that size.

## Sales audit rules [sarules]

### Functional Area

Automated Audit

### Module Affected

sarules.pc

### Design Overview

Evaluating rules is integral to the sales auditing process. Rules make the comparisons between data from various sources. These comparisons find data errors that could be the result of either honest mistakes or fraud. Finding these mistakes during the sales auditing process prevents these errors from being passed on to merchandising and data warehouse systems.

Rules in ReSA are dynamic. Rules are not predefined in the system – retailers have the ability to define through the online Rule Definition Wizard. The rule definition process produces a PL/SQL package that evaluates the rule and inserts an error into the error table if the rule condition is met. This batch program is simply a shell that calls these PL/SQL packages from batch during the data import/polling process.

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: This program should be run during each data import/polling.

Scheduling Diagram: The ReSA data import/polling process involves multiple programs. A complete description of the batch schedule is located in:

Pre-Processing: satotals.pc

Post-Processing: Optional, depending on export scheduling.

Threading Schema: This program runs against a single store at a time.

### Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

Driving Cursor

This program is designed to run multiple times a day, and therefore does not use restart-recovery.

### Program Flow/Technical Overview

Structure Chart

N/A

### Shared Modules

Listing of all externally referenced functions and stored procedures and description of usage

SA\_AUDIT\_RULES\_SQL.PROCESS\_AUDIT\_RULES – This PL/SQL function accepts a store\_day\_seq\_no as input and evaluates all of the rules defined for that store/day combination. It also writes any errors produced by the rules to the error tables.

**Function Level Description**

All database interactions required and error handling considerations

Process() – this function contains the driving cursor. This cursor selects all of the store\_day\_seq\_nos for the store inputted to the program that have a status of 'totaled'.

ProcessStoreDayTotals() - For each of these store\_day\_seq\_nos selected in process, the program calls SA\_AUDIT\_RULES\_SQL.PROCESS\_AUDIT\_RULES to evaluate the rules and insert any errors into the error table.

**I/O Specification**

All files layouts input and output

This program does not accept an input file, but does accept a store number on the command line.

# Sales audit store day create [sastdycr]

## Design Overview

This batch program will create store/day, import log and export log records. The program should be run prior to uploading the data for a given store/day.

## Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Four (?) - just before datsys.

Scheduling Diagram:

Pre-Processing:

Post-Processing:

Threading Scheme: N/A

## Restart Recovery

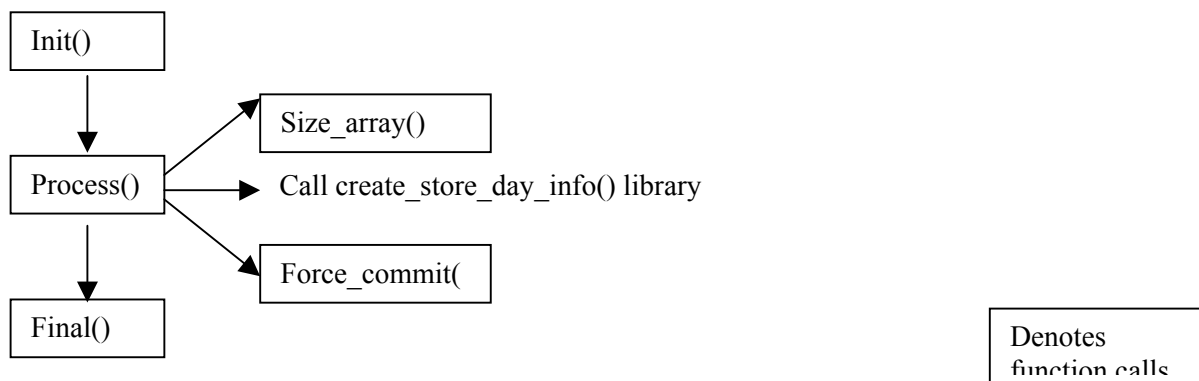
Logical Unit of Work ( recommended Commit check points )

Driving Cursor

This program must be restartable. The logical unit of work is store. The program will be threaded by store.

## Program Flow

Structure Chart



## Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

`create_store_day_info ()` – This function will insert into the `sa_store_day`, `sa_import_log`, `sa_flash_sales` and `sa_export_log` tables.

### Function Level Description

All database interactions required and error handling considerations

`Init()` – This function will call `restart_init` and get the `vdate` (using a cursor). The `vdate + 1` will be the 'action date' for which store/day records are created. The user may also pass in a date as an input parameter to the module. If this is done, the passed date will be used in place of `vdate + 1`.

`Process()` - This function will contain the driving cursor. The driving cursor should select all of the stores that are open today that do not already have store/days created for them. Three table hold store openings and closings. The `company_closed` table holds company wide closings. If there is a company wide closing, the `company_closed_excep` table holds any stores that are exceptions to the company closing (i.e. these stores are open). The `location_closed` table holds individual location closings. The function will call the `size_array` function to appropriately size the array of open stores. The array of open stores, the action date and the size of the array will be passed into the library function `create_store_day_info()`.

`Size_array()` – This function will size the array that holds the open stores.

`Force_commit()` – This function will commit records to the database by calling `restart_commit()`.

`Final()` - This function will call `restart_close`.

### I/O Specification

All files layouts input and output

N/A



## Sales audit totals [satotals]

### Design Overview

Totaling is integral to the sales auditing process. Totaling provides the values against which auditors can compare receipts. These comparisons find data errors that could be the result of either honest mistakes or fraud. Finding these mistakes during the sales auditing process prevents these errors from being passed on to merchandising and data warehouse systems. Totaling also provides quick access to other numeric figures about the days sales transactions.

Totaling in ReSA is dynamic. Totals are not predefined in the system – retailers have the ability to define through the online Total Definition Wizard. The total definition process produces a PL/SQL package that calculates a value for the total. This batch program is simply a shell that calls these PL/SQL packages from batch during the data import/polling process.

### Scheduling Constraints

#### Pre/Post Logic Description

Processing Cycle: This program should be run during each data import/polling.

Scheduling Diagram: The ReSA data import/polling process involves multiple programs. A complete description of the batch schedule is located in:

Pre-Processing: saimptlog\_finish.pc

Post-Processing: sarules.pc

Threading Schema: This program runs against a single store at a time.

### Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)  
Driving Cursor

This program is designed to run multiple times a day, and therefore does not use restart-recovery.

### Program Flow/Technical Overview

#### Structure Chart

N/A

### Shared Modules

Listing of all externally referenced functions and stored procedures and description of usage

SA\_TOTAL\_SQL.PROCESS\_CALC\_TOTALS – This PL/SQL function accepts a store\_day\_seq\_no as input and calculates all totals for that store/day combination. It also writes these calculated values to the appropriate values tables.

### Function Level Description

All database interactions required and error handling considerations

Process() – this function contains the driving cursor. This cursor selects all of the store\_day\_seq\_nos for the store inputted to the program that have a status of 'retotaling/reauditing required'.

ProcessStoreDayTotals() - For each of these store\_day\_seq\_nos selected in process, the program calls SA\_TOTAL\_SQL.PROCESS\_CALC\_TOTALS to calculate the total values and insert them into the total values tables.

**I/O Specification**

All files layouts input and output

This program does not accept an input file, but does accept a store number on the command line.

## Retek supplier cost changes update [sccext]

### Design Overview

The Supplier Cost Change Extract (sccext) module selects supplier cost change records, which are set to go into effect tomorrow and updates the RMS item/supplier table with the new cost. The item/location tables are also updated with the new cost if the cost change is a base cost change (supplier is the primary supplier for the item). In addition to the updates to the unit costs on the item/location tables when the base cost changes, records are also written for price history, supplier history and transaction-level stock ledger (if using standard cost). The costs on approved orders may also be updated if the recalculate order indicator is set to Yes for the item/supplier combination. The Pre/Post program for multi-threadable programs should be run after this program to update the status of the cost change to Extracted.

### Tables Affected:

| TABLE                          | INDEX | SELECT | INSERT | UPDATE | DELETE |
|--------------------------------|-------|--------|--------|--------|--------|
| COST_SUSP_SUP_DETAIL           | No    | Yes    | No     | No     | No     |
| COST_SUSP_SUP_HEAD             | No    | Yes    | No     | No     | No     |
| DEAL_CALC_QUEUE                | No    | Yes    | Yes    | Yes    | No     |
| DEAL_SKU_TEMP                  | No    | No     | Yes    | No     | No     |
| PERIOD                         | No    | Yes    | No     | No     | No     |
| PRICE_HIST                     | No    | No     | Yes    | No     | No     |
| ITEM_SUPPLIER                  | No    | Yes    | No     | Yes    | No     |
| SUPS                           | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS                 | No    | Yes    | No     | No     | No     |
| V_RESTART_COST_CHG             | No    | Yes    | No     | No     | No     |
| COST_SUSP_SUP_DETAIL_LOC       | No    | Yes    | No     | No     | No     |
| ORDLOC                         | No    | Yes    | No     | Yes    | No     |
| ORDHEAD                        | No    | Yes    | No     | No     | No     |
| ITEM_SUPP_COUNTRY              | No    | Yes    | No     | Yes    | No     |
| ITEM_SUPP_COUNTRY_LOC          | No    | Yes    | No     | Yes    | No     |
| ITEM_SUPP_COUNTRY_BRACKET_COST | No    | Yes    | Yes    | Yes    | No     |
| ITEM_MASTER                    | No    | Yes    | No     | No     | No     |
| PACKITEM                       | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Ad Hoc (run as required since not impacted by inventory position)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing:        sccext\_post() – records in cost\_susp\_sup\_head are updated to set status = 'E' where status = 'A' and active\_date = (select vdate + 1 from period);

Threading Scheme:     COST\_CHANGE (V\_restart\_cost\_chg)

### Restart Recovery

```
EXEC SQL DECLARE c_head CURSOR FOR
 SELECT h.cost_change,
 h.reason,

 FROM v_restart_cost_chg rv,
cost_susp_sup_head h
 WHERE h.status = 'A'
 AND h.active_date = to_date(:tomorrow, 'DDMMYYYY')

 AND rv.driver_value = h.cost_change
 AND rv.driver_name = :ora_restart_driver_name
 AND rv.thread_val = :ora_restart_thread_val
 AND rv.num_threads = :ora_restart_num_threads
 AND (h.cost_change > NVL(:ora_restart_cost_chg, -999) OR
 (h.cost_change = :ora_restart_cost_chg))
 ORDER BY h.cost_change,
```

### Program Flow

N/A

### Shared Modules

**COST\_EXTRACT\_SQL.UPDATE\_COSTS** – This function will update costs in RMS based on the cost change and cost change reason passed in (all processing will be handled in this package). If the cost change reason is 1 or 2 for the cost change, then the brackets will be inserted onto **ITEM\_SUPP\_COUNTRY\_BRACKET\_COST**. The unit costs on **ITEM\_SUPP\_COUNTRY\_LOC** will be updated with the default bracket cost. If the location is the primary location, update the unit cost on **ITEM\_SUPP\_COUNTRY**. If the item is on an approved order, then the item cost on the order will be updated. If the item is a component item in a buyer pack, the pack cost will be updated using the new component cost. If the item has children, the same processing as described above will occur for all children items. If it is ran in a multi-channel environment, only virtual locations will receive the new brackets. For any other cost change reason:

- If bracket are to be updated with new costs, **ITEM\_SUPP\_COUNTRY** and **ITEM\_SUPP\_COUNTRY\_LOC** will be updated with the default bracket costs. If the item is on an approved order, its cost will be updated with the new base cost. If the item is in a buyer pack, the pack will be updated with the new cost base cost of the item. If the item has children, the child items will be processed in the same manner.
- If location costs are only updated, **ITEM\_SUPP\_COUNTRY\_LOC** will be updated with the new location costs. The unit cost will be updated on **ITEM\_SUPP\_COUNTRY** if the location updated is the primary location. If the item is on an approved order, its cost will be updated with the new base cost. If the item is in a buyer pack, the pack will be updated with the new cost base cost of the item. If the item has children, the child items will be processed in the same manner.

- If the item is costed at the item level only, then ITEM\_SUPP\_COUNTRY will be updated with the new unit cost. This cost will be updated to all locations on ITEM\_SUPP\_COUNTRY\_LOC. If the item is on an approved order, its cost will be updated with the new base cost. If the item is in a buyer pack, the pack will be updated with the new cost base cost of the item. If the item has children, the child items will be processed in the same manner.

If these three bullets occur in a multi-channel environment and the location is a warehouse, only the virtual warehouse locations will be updated.

#### **Function Level Description**

process() – main processing function

while(1) – main loop

```
{
 call COST_EXTRACT_SQL.UPDATE_COSTS ()

}
```

#### **I/O Specification**

N/A

#### **Technical Issues**

N/A

## Store ship schedule purge [schedprg]

### Design Overview

This program will purges all store\_ship\_date, company\_closed, company\_close\_excep, and location\_closed records whose dates are outside of the system options which control the retention period. The number of months of retention for the store ship schedule and the number of months of retention for the location close schedules is kept on the system\_options table. This program should be run monthly.

The following database tables are accessed by this program:

| TABLE                | SELECT | INSERT | UPDATE | DELETE |
|----------------------|--------|--------|--------|--------|
| SYSTEM_OPTIONS       | Yes    | No     | No     | No     |
| STORE_SHIP_DATE      | No     | No     | No     | Yes    |
| COMPANY_CLOSED       | No     | No     | No     | Yes    |
| COMPANY_CLOSED_EXCEP | No     | No     | No     | Yes    |
| LOCATION_CLOSED      | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

This program will use the commit\_max\_ctr on the restart\_control table to periodically commit SQL delete operations.

### Program Flow

N/A

### Shared Modules

Restart\_init() - initialize restart recovery tables

Restart\_close() - commit changes and update restart recovery tables

### Function Level Description

Init()

This function calls restart\_init function to initialize the restart\_recovery process. The g\_l\_restart\_max\_counter is assigned by the restart\_init function from the max\_commit\_ctr on the restart\_control table. This counter is used in the process function to perform periodic commits.

This function also queries the `system_options` table for the `ship_sched_history_mths` and the `loc_close_hist_months` values. Also the period table for the current vdate.

#### Process()

This function deletes `store_ship_date` records that are older than the number of months specified by `ship_sched_history_mths`. Also, `company_closed`, `company_closed_excep`, and `location_closed` records that are older than the `loc_close_hist_months` variable are deleted. A while loop is used to force periodic commits of the deletes based on the value of `g_l_restart_max_counter`. This is done to ensure that rollback segments will not be exceeded in the event of considerable volume

#### Final()

This function calls the restart close function to update the restart recovery tables with the final program status (completed if finished successfully, aborted if handled exceptions occur).

## Scheduled item maintenance main [sitmain]

### Design Overview

This new batch program will have two different processes where the first one needs to be complete before the second one could commence.

First process will involve the tracking of addition to item lists or location lists. If there is a pre-existing link that involves item list or location list where addition took place, this program will get the latest status from previous update that have occurred in the past. If the given item/loc relationship exists then the status and status\_update\_date field in the ITEM\_LOC table will be updated accordingly. Otherwise, a new item/loc relationship will be created. After the update/insert into ITEM\_LOC table all records in SIT\_EXPLODE table with update\_ind of 'Y' will be updated to 'N'.

In the second process, scheduled item maintenance will take place by obtaining the data from SIT\_DETAIL table. It will update the item/location status on the item\_loc table as specified on the sit\_detail table for given item list/location list combination. If the item/location relationship does not exist at the time of update, it will be created. At the end of the second process, old records in the SIT\_DETAIL table will be deleted.

The LUW for this module is location/item.

### Stored Procedures / Shared Modules (Maintainability)

**NEW\_ITEM\_LOC:** This procedure will be used to create new item/location relationship prior to status update.

### Function Level Description

**Main():** Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

**Init():** Standard Retek init function. Calls retex\_init().

**Process():** Drives the business logic for this program. There are two processes that are involved in the program.

- Call process\_additions().
- Call process\_maintenance().

**Process\_additions():** This function will process the first driving cursor that consist of a union between two select statements. First select statement should get item, location from SIT\_EXPLODE table, its corresponding status from SIT\_DETAIL table and '1' where status\_update\_date is less than vdate + 1 in SIT\_DETAIL table and update\_ind from SIT\_EXPLODE is 'Y'. It is necessary to check that select item/location relationship exists in ITEM\_LOC table. The second select statement should get be similar to the first one except that is should select '2' instead of '1' and there should be no pre-existing item/location relationship in the ITEM\_LOC table for each record.

- Array-fetch the driving cursor. For each record fetched:
  - Call update\_insert() function.
- After the insert/update process to the ITEM\_LOC table, update all records in SIT\_EXPLODE with update\_ind value of 'Y' to 'N'.



**Process\_maintenance():** This function will process the second driving cursor that consist of a union between two select statements. First select statement should get item, location from SIT\_EXPLODE table, its corresponding status from SIT\_DETAIL table and '1' where status\_update\_date is vdate + 1 in SIT\_DETAIL table. It is necessary to check that select item/location relationship exists in ITEM\_LOC table.

The second select statement should get be similar to the first one except that is should select '2' instead of '1' and there should be no pre-existing item/location relationship in the ITEM\_LOC table for each record.

- Array-fetch the driving cursor. For each record fetched:
  - Call update\_insert() function.
- After all the records in the array are processed, call delete\_sit\_detail() function to delete old records.

**Update\_insert():** This function will loop through the arrays fetched into in process\_additions and process\_maintenance.

- If update indicator is '1' then copy values to new arrays for updating.
- If update indicator is '2' then, call NEW\_ITEM\_LOC to create the item/location relationship.
- After looping through all the indices, array update the status of item/location in ITEM\_LOC table using the arrays copied to when update indicator was '1'.

**Delete\_sit\_detail():** This function will select all itemloc\_link\_id in SIT\_DETAIL table that has status\_update\_date equal to vdate + 1. For each itemloc\_link\_id, it will delete any record that has status\_update\_date that is before vdate + 1.

**Final():** Performs restart/recovery close logic. Calls retek\_close().

### Input Specifications

#### 'Table-To-Table'

Select date from:

| Table Name  | Column Name        | Column Type  | Transformation |
|-------------|--------------------|--------------|----------------|
| SIT_EXPLODE | ITEM               | VARCHAR2(25) | NONE           |
| SIT_EXPLODE | LOCATION           | NUMBER(10)   | NONE           |
| SIT_DETAIL  | STATUS_UPDATE_DATE | DATE         | NONE           |
| SIT_DETAIL  | STATUS             | VARCHAR2(1)  | NONE           |

### Output Specifications

#### 'Table-To-Table'

The following table will be updated:

| Table Name  | Column Name        | Column Type | Transformation       |
|-------------|--------------------|-------------|----------------------|
| ITEM_LOC    | STATUS             | VARCHAR2(1) | 'A', 'I', 'C' or 'D' |
| ITEM_LOC    | STATUS_UPDATE_DATE | DATE        | NONE                 |
| SIT_EXPLODE | UPDATE_IND         | VARCHAR2(1) | 'N'                  |

**Scheduling Considerations**

This program should always run after lclrbld.pc on an ad hoc basis. More specifically the running of this program is dependant on the date in SIT\_DETAIL table. If today's date + 1 is equal to a date in SIT\_DETAIL table then this program should be executed.

**Locking Strategy**

N/A

**Restart/Recovery**

Restart recovery for this module will be based on location. Each of the processes will have separate restart/recovery maintenance. First process must be committed before second process can begin.

**Security Considerations**

N/A

## Location security rebuild [slocrbld]

### Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Slocrbld.pc will handle the maintenance for the location security data. Locations will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/organizational level to define whether a user can select or update every single location covered by the defined rules. The functional document describes the architecture of the security features and how it works. Rules that have a smaller scope overwrite those with a broader scope. For example, a user is assigned to two groups -- one of the groups has no update capability for a given region, while the other group allows updating for a specific store within that region. Which applies? The rule applying to the lowest level in the organization hierarchy is the rule granting the update capability for the store. Therefore, for the one store in the region, updates will be allowed. For the rest of the locations in the region, no updating will be allowed. In addition, if there are conflicting security definitions at the same hierarchy level because a user is associated with more than one group, the user is, as expected, granted the capability.

To accomplish this task as efficiently as possible, the program should be built as follows. It will be multi-threaded by store/warehouse, and use restart\_recovery. In the Init routine, an array which will closely resemble the final destination security table, will be sized to handle all the locations in the particular thread running. This array will be loaded with all the locations and used repeatedly for every user/functional area combination. There will be an additional indicator (in addition to the select/update indicators) that will keep track of which locations have a rule affecting them and have therefore been "touched". Each rule will affect certain locations in the array and their attributes may be changed multiple times. When they are changed, this indicator will be raised. After all the rules are processed for a given user/functional area, the data in the array that has the "touched" indicator raised will be written out to sec\_user\_loc\_matrix table and its indicator reset. This cycle will be repeated until all users and functional areas are exhausted.

| TABLE                | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------------|-------|--------|--------|--------|--------|
| SEC_USER_GROUP       | No    | Yes    | No     | No     | No     |
| SEC_GROUP_LOC_MATRIX | No    | Yes    | No     | No     | No     |
| V_RESTART_STORE_WH   | No    | Yes    | No     | No     | No     |
| REGION               | No    | Yes    | No     | No     | No     |
| DISTRICT             | No    | Yes    | No     | No     | No     |
| STORE                | No    | Yes    | No     | No     | No     |
| WH                   | No    | Yes    | No     | No     | No     |
| SYSTEM_VARIABLES     | No    | Yes    | No     | No     | No     |
| SEC_USER_LOC_MATRIX  | No    | No     | Yes    | No     | No     |

**Scheduling Constraints**

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters *slocrbld pre* , *slocrbld.pc* and *prepost.pc* with parameters *slocrbld post* in series.

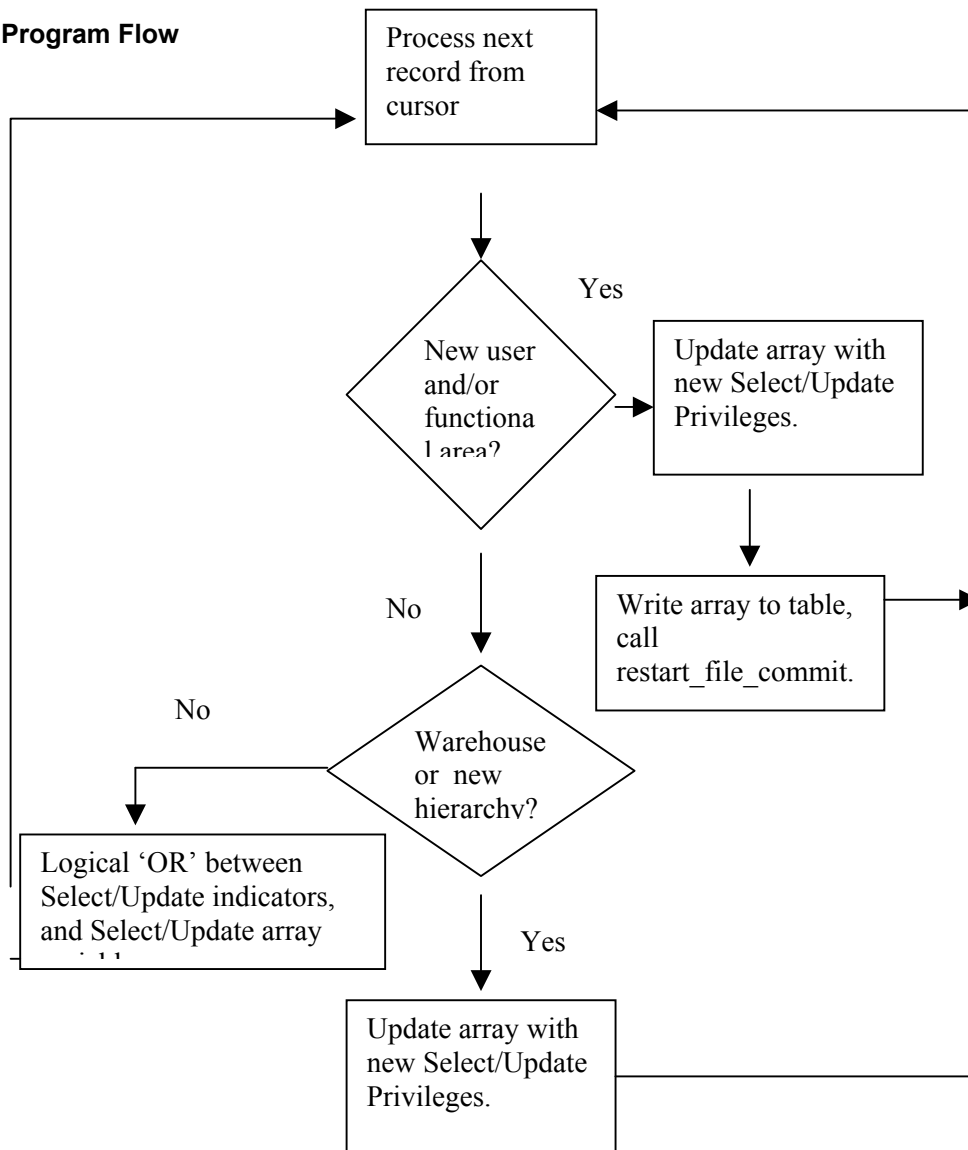
Pre-Processing: Prepost with parameters: *slocrbld pre*

Post-Processing: Prepost with parameters: *slocrbld post*

Threading Scheme: Store/Warehouse

**Restart Recovery**

The logical unit of work for location security rebuild will be the user-functional area (column\_code). Restart/recovery will be based on the user-functional area. The restart commit counter will need to be carefully determined by each client according to the number of store/warehouse combinations that will be affected by the location security rebuild. Large location security rebuilds with thousands of stores/warehouses need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small location security rebuilds with small amount of stores/warehouses can have much larger commit counters since fewer rows will be inserted into the database each time for one user-functional area.

**Program Flow****Shared Modules**

N/A

**Driving Cursor:**

```

SELECT distinct u.user_id,
 l.column_code,
 l.region,
 l.district,
 l.store,
 l.wh,
 l.select_ind,
 l.update_ind
FROM sec_user_group u,
 sec_group_loc_matrix l,
 district d,
 store s,

```

```

 v_restart_store_wh v
WHERE u.group_id = l.group_id
 AND ((l.region = d.region
 AND l.district is NULL
 AND d.district = s.district
 AND v.driver_value = s.store)
 OR (l.district = s.district
 AND l.store is NULL
 AND v.driver_value = s.store)
 OR (l.store = v.driver_value
 OR v.driver_value = l.wh)
 AND v.num_threads = :pi_restart_num_threads
 AND v.thread_val = :pi_restart_thread_val
 AND (u.user_id > NVL(:ps_restart_user, '-999')
 OR (u.user_id = :ps_restart_user
 AND l.column_code > :ps_restart_column_code))
ORDER BY u.user_id, l.column_code, l.region desc, l.district desc, l.store desc,
 l.wh desc;

```

### Function Level Description

Main

Init()

- **Check SYSTEM\_VARIABLES.update\_loc\_sec\_ind.** If the indicator is not set then the program exit normally without further processing.
- Call retek\_init() to get restart-recover variables.
- Get\_total\_locs()  
Get total stores and warehouses in the current thread.
- Size\_loc\_array()  
Size location array based on the number of locations in the current thread. The location array includes region, district, loc, loc\_type, select\_ind, update\_ind and touched columns.
- Size\_loc\_insert\_array()  
Size location insert array based on the maximum commit count set in restart\_control table. This array will be used for holding the exact data that is going to be insert into the sec\_user\_loc\_matrix table. The location insert array includes user\_id, column\_code, loc, loc\_type, select\_ind and update\_ind columns.
- Load\_loc\_array()  
Load all the stores/warehouses including organizational info in the current thread to the location array.

Process()

The driving cursor is ordered to return records defining rules for entire regions first, and then those for districts, and on down. The records are processed in that order. That is to say, first work with the region level rules, then move to the more specific rules so that the rules with the

smaller scope take priority over the higher level rules. Warehouses are not included in the hierarchy and Each warehouse translates into a rule to be written out.

- Call `size_rule_array()` to allocate memory for arrays that store security rules.
- Open the driving cursor in a while loop. Fetch the data into rule array.
- Call `set_null_to_field()` to set fields to null when those fields' indicators are -1 in the rule array.
- Check if this is a second array fetch or greater, if yes, call `process_record()` to process the last record in last array fetch and the first record in current array fetch.
- Open a for loop
  - Call `process_record()` to process the current and last record.
- End of for loop
- Copy the last record in the current array fetch to last rule array. Since the last record of an array fetch hasn't been processed until compared to the first record of the next array fetch. However, with each new array fetch, the last record of the previous array fetch is overwritten. Thus here it needs to be copied.
- End of while loop.

#### `Size_rule_array()`

This function allocates memory for arrays that store security rules based on the maximum commit count set in table `restart_control` table. The rule array includes `user_id`, `column_code`, `region`, `region_ind`, `district`, `district_ind`, `store`, `store_ind`, `wh`, `wh_ind`, `select_ind` and `update_ind`.

#### `Set_null_to_field()`

This function loops through all the records in rule array and set a field to null when the field's indicator is -1.

#### `Process_record()`

This function does the majority of the processing. The data from the driving cursor is ordered by region and district. Therefore, all rules for a particular organizational hierarchy will be grouped together and processed so that a single security rule will be decided for that particular hierarchy. When multiple records do occur at the same level, the logical OR will be used to determine whether to grant update/select privileges.

- Compare the user/functional area of the current record and the last record :
  - If it isn't new:
    - Compare the hierarchy/warehouse of the current record and the last record:
      - If it isn't new, call `logical_or_indicators()` to update the current record's select and update indicators according to the logical 'OR' between the current and last records' indicators.
      - If it is new, call `update_array()` to blow security rule down to the store/warehouse level according to the last record rule.
  - If it is new:
    - Call `update_array()` to blow security rule down to the store/warehouse level according to the last record rule.
    - Call `write_array()` to insert the security rules of last record's user/functional area (down to store/warehouse level) to table `sec_user_loc_matrix`.

- Call `retek_force_commit()` to commit the output to database and set book mark in the `restart_bookmark` table.

#### Logical\_or\_indicators()

This function updates the input current record's select and update indicators according to the logical 'OR' between the input current and last records' indicators. For example, if the current record's select indicator is 'N', the last record's select indicator is 'Y', then the current record's select indicator is updated to 'Y'; If the current record's select indicator is 'N', the last record's select indicator is 'N', then the current record's select indicator is kept untouched('N'). If the current record's select indicator is 'Y', no matter what last record's select indicator is, the current record's select indicator is kept untouched('Y'). So does update indicator.

#### Update\_array()

This function updates the location array according to the input security rule.

- If the input rule is a region level security rule, then loop through the location array, for all the stores within the region, set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set touched column of each processed row to 'Y'.
- If the input rule is a district level security rule, then loop through the location array, for all the stores within the district, set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set touched column of each processed row to 'Y'.
- If the input rule is a store level security rule, then loop through the location array, set the `select_ind` and `update_ind` of the store equal to the input rule's `select_ind` and `update_ind`, respectively. Set touched column of each processed row to 'Y'.
- If the input rule is a warehouse level security rule, then loop through the location array, set the `select_ind` and `update_ind` of the warehouse equal to the input rule's `select_ind` and `update_ind`, respectively. Set touched column of each processed row to 'Y'.

#### Write\_array()

This function writes out rows with touched = 'Y' in the location array to `sec_user_loc_matrix` table.

- Create a loop through the location array
  - Check the location insert array, if it is full, call `insert_array_to_table()`.
  - If the touched column of the row is 'Y', copy the row to location insert array. Set the touched column of the row to 'N'.
- If the location insert array is not empty, call `insert_array_to_table()`.

#### Insert\_array\_to\_table()

This function uses array insert to insert the contents in the location insert array to table `sec_user_loc_matrix`.

#### final():

restart/recovery close

#### I/O Specification

N/A

#### Technical Issues

N/A



# Stock on hand download [sohdnld]

## Functional Area

RMS Extract to RPAS

## Module Affected

sohdnld.pc

## Design Overview

This batch program will query stock on hand information for all items and write a flat file to be used by the RPAS.

## Stored Procedures / Shared Modules (Maintainability)

N/A

## Input Specifications

The driving cursor for this program will select the item, location, and stock on hand for each item for which RMS has a stock on hand record. It will also calculate the total unit retail and total cost for each item/location combination.

The cursor is as follows:

```
SELECT ils.item,
 ils.loc,
 ils.stock_on_hand+ils.pack_comp_soh, /* stores never have pack_comp_soh */
 NVL(il.unit_retail,0)*(ils.stock_on_hand+ils.pack_comp_soh),
 DECODE(:ps_std_av_ind, 'A', NVL(ils.av_cost,0),
 NVL(ils.unit_cost,0))*(ils.stock_on_hand+ils.pack_comp_soh),
 ils.loc_type
FROM item_loc_soh ils,
 item_loc il,
 v_restart_store_wh vrs
WHERE ils.item = il.item
 AND ils.loc = il.loc
 AND (ils.loc > :ps_restart_location
 OR (ils.loc = :ps_restart_location
 AND ils.item > :ps_restart_item))
 AND vrs.driver_value = ils.loc
 AND vrs.thread_val = :ps_thread_val
 AND vrs.num_threads = :ps_num_threads
ORDER BY 2, 1; /* loc, item */
```

## Output Specifications

Output Files

| Field Name | Field Type   | Default Value    | Description                                         |
|------------|--------------|------------------|-----------------------------------------------------|
| Item       | Varchar2(25) | Item_master.item | Unique alphanumeric value that identifies the item. |

|               |              |                            |                                                                      |
|---------------|--------------|----------------------------|----------------------------------------------------------------------|
| Location      | Number(20)   |                            | Unique numeric value that identifies the location.                   |
| Location type | Varchar2(1)  | 'S' or 'W'                 | Identifies which type the location is, 'S'tore or 'W'arehouse.       |
| Date          | Date(8)      |                            | System date (vdate)                                                  |
| Soh           | Number(17,4) | Item_loc_soh.stock_on_hand | Current stock on hand for the item.                                  |
| Retail        | Number(25,4) | Varies                     | The retail value for the total stock on hand for this item/location. |
| Cost          | Number(25,4) | Varies                     | The cost for the total stock on hand for this item/location.         |

### Function Level Description

**Init()** : This function will get the vdate from period and std\_av\_ind from system\_options. It will also set the filename to sohndld.dat.nn where nn is thread value.

**Process()**: This function will first allocate memory for the cursor array by calling size\_array(). It will then call define\_buffer(). Next it will fetch the driving cursor and loop through the array, calling write\_to\_file each time though. At the completion of the loop, it will call free\_array passing in the driving cursor array.

**Define\_buffer()** : This function will set up the format string for the file.

**Write\_to\_file(char\* is\_item,  
char\* is\_location,  
double id\_soh,  
double id\_retail,  
double id\_cost,  
char\* is\_type)** This function convert the retail and cost to the primary currency by calling convert\_to\_primary. Then it will write to the file the information passed in from the driving cursor array.

**Size\_array(driving\_cursor\_struct\*)** This function will allocate memory for the driving cursor array.

**Free\_array(driving\_cursor\_struct\*)** This function will free all the memory taken up by the driving cursor array.

**Final()** : This function will call retek\_close().

### Restart/Recovery

This program will use restart/recovery functionality. The logical unit of work is the item/location combination. Records will be committed when the number of records processed equals the commit\_max\_counter.

## Store orders upload [soupld]

### Design Overview

The purpose of this batch module is to process store orders created from an external system. The upload will provide information for the replenishment process to generate recommended order quantities. The input file will contain the item to be ordered, the store requesting the item, the need quantity, the unit of purchase of the need quantity and the need date. The need quantity will be stored in the standard unit of measure. Validation will check that the item and store are on replenishment with a replenishment method of “Store Orders.”

If a record exists on STORE\_ORDERS for the item, store and need date, the need quantity will be added onto the existing need quantity on the table. If no record exists, a new record will be inserted into STORE\_ORDERS.

### Stored Procedures / Shared Modules (Maintainability)

None

### Program Flow

```

main()
|
+--init()
|
+--process()
| +--size_arrays()
| |
| +--validate_FDETL()
| | |
| | +--verify_repl_info()
| | |
| | +--convert_need_qty ()
| |
| +--verify_on_store()
| |
| +--process_FDETL()
|
+--final()

```

### Function Level Description

#### Init()

This function will initialize input files and restart recovery. It will also allocate memory for the insert and update arrays.

#### Process()

This function will read each input file and check if it is a detail record or a trailer record. If it is a detail record, it validates each record field. If a non-fatal error exists as a result of the validation, a reject field record is created. If there are no errors found, either a record is inserted into

STORE\_ORDERS or an existing STORE\_ORDERS record is updated. Restart-recovery control variables are updated. The function returns to the main function when a trailer record is read.

### **Validate\_FDETL()**

This function will validate all of the input data fields. Each field will be validated as follows:

- *Item* must be populated.
- *Store* must be populated and numeric.
- *Need Quantity* must be populated and numeric.
- *Unit of Purchase* must be populated and must be 'EA', 'CA' or 'PA'.
- *Need Date* must be populated and a valid date in the YYYYMMDD format.
- The combination of the item and store must exist on REPL\_ITEM\_LOC with a replenishment method of "Store Orders".

If any of the above validation fails, a non-fatal error is produced.

### **Verify\_repl\_info()**

This function will verify that the item and store exist on replenishment with a replenishment method of 'Store Orders'.

It also provides for the following checks on need date:

- If repl\_item\_loc.reject store orders ind is 'N', all need dates are valid.
- If the indicator is 'Y', only files with the need date on or after REPL\_ITEM\_LOC.NEXT\_DELIVERY\_DATE are valid.
- If the indicator is 'Y' and NEXT\_DELIVERY\_DATE is NULL then, all need dates are valid.

### **Convert\_need\_qty()**

If the unit of purchase is not 'EA', this function will convert the need quantity into eaches by looking at the SUPP\_PACK\_SIZE or TI and HI values on ITEM\_SUPP\_COUNTRY where the supplier and country are the primary supplier and primary country stored on ITEM\_LOC for the item and store.

### **Process\_FDETL()**

This function will either insert into or update STORE\_ORDERS. If the item, store and need date already exist on STORE\_ORDERS, update this record on STORE\_ORDERS by adding the need quantity to the record's existing need quantity. Otherwise, insert a record.

### **Verify\_on\_store()**

This is called by Process\_FDETL to verify that a record exists on STORE\_ORDERS for the item/store/need\_date combination.

### **Final()**

This function will free the memory used in the insert and update record arrays, cleans up the restart-recovery tables and performs a rollback or final commit.

## **Input Specifications**

### **'Table-To-Table'**

Input Files

| Record Name  | Field Name                  | Field Type            | Default Value                         | Description                                                                                 |
|--------------|-----------------------------|-----------------------|---------------------------------------|---------------------------------------------------------------------------------------------|
| File Header  | File Type Record Descriptor | Char(5)               | FHEAD                                 | Identifies file record type                                                                 |
|              | File Line Identifier        | Numeric ID(10)        | Sequential number Created by program. | ID of current line being created for output file.                                           |
|              | File Type Definition        | Char(4)               | 'SOUP'                                | Identifies file as 'Store Order Upload'                                                     |
|              | File Create Date            | Char(14)              | create date                           | current date, formatted to 'YYYYMMDDHH24MISS'.                                              |
| File Detail  | File Type Record Descriptor | Char(5)               | FDETL                                 | Identifies file record type                                                                 |
|              | File Line Identifier        | Numeric ID(10)        | Sequential number Created by program. | ID of current line being created for output file.                                           |
|              | Item                        | Varchar2(25)          | Item identifier                       | The item that is being ordered by the store.                                                |
|              | Store                       | Number(10)            | Store Identifier                      | The store is ordering the item.                                                             |
|              | Need_qty                    | Number(12, implied 4) |                                       | Number of units to be ordered.                                                              |
|              | UOP                         | Varchar2(4)           |                                       | The unit of purchase that the order quantity is in. Valid values are 'EA', 'CA' and 'PA'.   |
|              | Need_date                   | Date                  |                                       | The date that the order needs the items by, formatted to 'YYYYMMDD'                         |
| File Trailer | File Type Record Descriptor | Char(5)               | FTAIL                                 | Identifies file record type                                                                 |
|              | File Line Identifier        | Numeric ID(10)        | Sequential number Created by program. | ID of current line being created for output file.                                           |
|              | File Record Counter         | Numeric ID(10)        |                                       | Number of records/transactions processed in current file (only records between head & tail) |

### Output Specifications

#### 'Table-To-Table'

Tables updated by this program:

## STORE\_ORDERS

Tables inserted into by this program:  
STORE\_ORDERS

### Output File

#### Reject File:

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

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

### Scheduling Considerations

This batch program should run before any replenishment programs in order to ensure that all relevant store orders are captured in the replenishment process. More specifically, it should run in the phase 2 processing cycle.

### Locking Strategy

N/A

### Restart/Recovery

The logical unit of work for the `soupld` module is a store order for an item. The `COMMIT_MAX_CTR` field on `RESTART_CONTROL` will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups of numbers equal to the `COMMIT_MAX_CTR`. After a logical unit of work is processed, the restart-commit logic and restart-file-writing logic will be called, and then the next logical unit of work will be processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters), and commit all database transactions. The file-writing logic will append the temporary holding files to the final output files.

The `COMMIT_MAX_CTR` field should be set to prevent excessive rollback space usage and to reduce the overhead of file I/O.

Validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining records will be validated, but the invalid record will be rejected and written to the reject file. If a fatal error is returned, then file processing will end immediately. A restart will be

initiated from the file pointer position saved in the RESTART\_BOOKMARK string at the time of the last commit point that was reached during file processing.

**Performance Considerations**

N/A

**Security Considerations**

N/A

**Unit Test Considerations**

Test cases should include an item and store record that does not exist on replenishment with a replenishment method of 'Store Orders'. They should also include input files where the item, store and need date do or do not already exist on STORE\_ORDERS. Finally, the input files should contain different units of purchase to ensure that the quantity is converted correctly.

## Product security rebuild [sprdrbld]

### Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Sprdrbld.pc will handle the maintenance for the product security data. Items will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/merchandise level to define whether a user can select or update every single Item covered by the defined rules. The functional document describes the architecture of the security features and how it works. Rules that have a smaller scope overwrite those with a broader scope. For example, a user is assigned to two groups -- one of the groups has no update capability for a given department, while the other group allows updating for a specific class within that department. Which applies? The rule with the lowest item hierarchy in its definition is the rule granting the update capability for the class. Therefore, for every Item in the department and in the class will be allowed to update. For the rest of the items in the department, no updating will be allowed. In addition, if there are conflicting security definitions at the same hierarchy level because a user is associated with more than one group, the user is, as expected, granted the capability.

Performance is a crucial consideration for this program as it involves writing records for different functional areas at the item level for every user in the system. To accomplish this task as efficiently as possible, the program should be built as follows. It will be multi-threaded by department, and use restart\_recovery. In the Init routine, an array which will closely resemble the final destination security table, will be sized to handle all the Items in the particular thread running. This array will be loaded with all the Items and used repeatedly for every user/functional area combination. There will be an additional indicator (in addition to the select/update indicators) that will keep track of which Items have a rule affecting them and have therefore been "touched". Each rule will affect certain items in the array and their attributes may be changed multiple times. When they are changed, this indicator will be raised. After all the rules are processed for a given user/functional area, the data in the array that has the "touched" indicator raised will be written out to a SQL Loader file and its indicator reset. This cycle will be repeated until all users and functional areas are exhausted.

| TABLE                 | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-----------------------|-------|--------|--------|--------|--------|
| SEC_USER_GROUP        | No    | Yes    | No     | No     | No     |
| SEC_GROUP_PROD_MATRIX | No    | Yes    | No     | No     | No     |
| V_RESTART_DEPT        | No    | Yes    | No     | No     | No     |
| ITEM_MASTER           | No    | Yes    | No     | No     | No     |
| SYSTEM_VARIABLES      | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters



*sprdrbld pre* , *sprdrbld.pc* and *prepost.pc* with parameters *sprdrbld post* in series. Then use SQL load control file *sprdrbld.ctl* to load the output file from *sprdrbld.pc* to database.

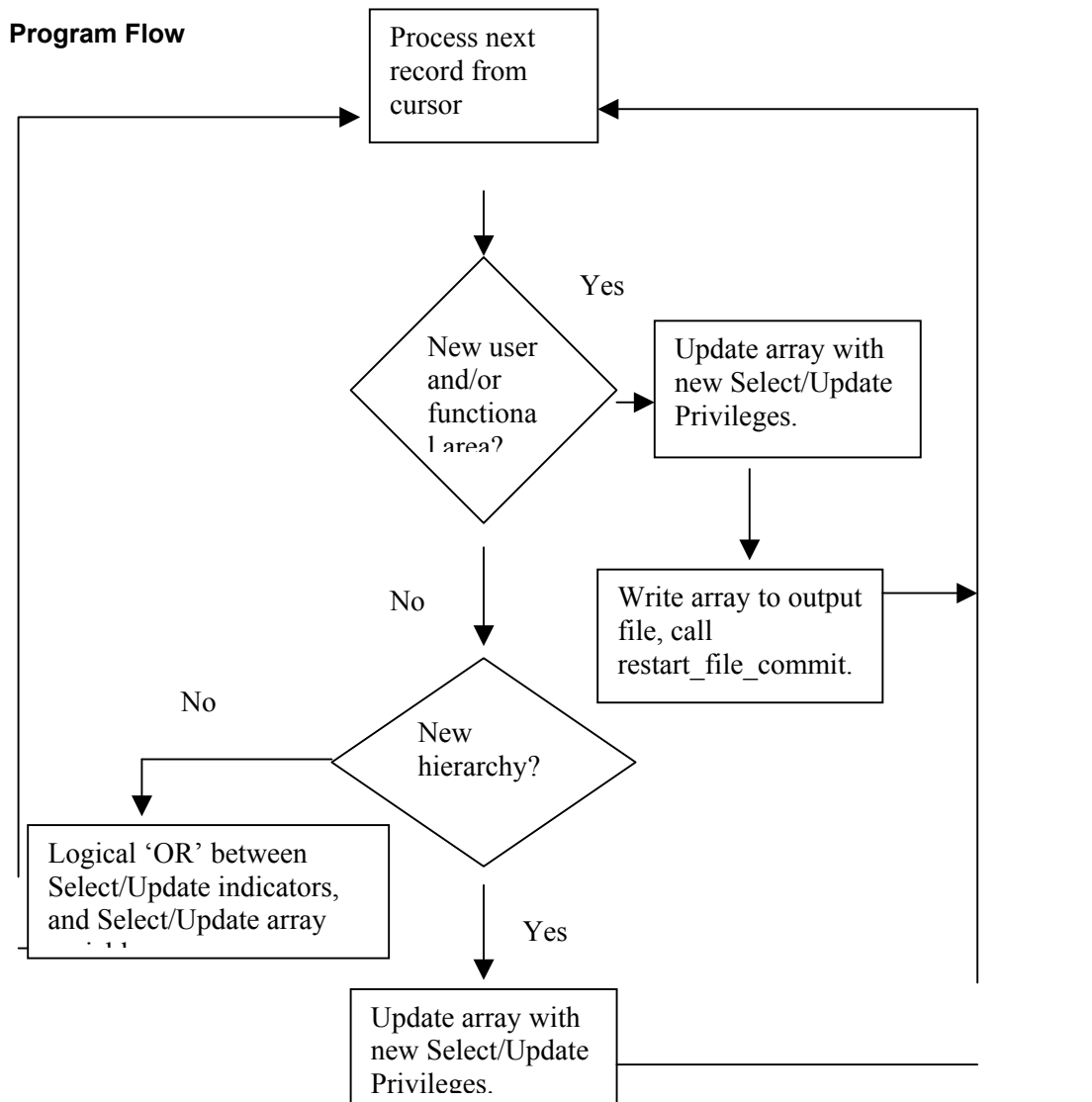
Pre-Processing: Prepost with parameters: *sprdrbld pre*

Post-Processing: Prepost with parameters: *sprdrbld post*

Threading Scheme: Department

### **Restart Recovery**

The logical unit of work for location security rebuild will be the user-functional area (column\_code). Restart/recovery will be based on the user-functional area. The restart commit counter will need to be carefully determined by each client according to the number of departments that will be affected by the product security rebuild. Large product security rebuilds with thousands of items need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small location security rebuilds with small amount of items can have much larger commit counters since fewer rows will be inserted into the database each time for one user-functional area.



### Shared Modules

N/A

### Function Level Description

Main()

Init()

- Check `SYSTEM_VARIABLES.update_prd_sec_ind`. If the indicator is not set then the program exit normally without further processing.
- Call `retek_init()` to get restart-recover variables.

- `Get_total_skus()`  
Get total items in the current thread.
- `Size_item_array()`  
Size item array based on the number of items in the current thread. The item array includes dept, class, subclass, level1, level2, level3, select\_ind, update\_ind and touched columns.
- `Load_item_array()`  
Load all items in the current thread to the item array.

#### Process()

The driving cursor is ordered to return records defining rules for entire department first, and then those for class, and on down. The records are processed in that order. That is to say, first work with the department level rules, then move to the more specific rules so that the rules with the smaller scope take priority over the higher level rules.

- Call `size_rule_array()` to allocate memory for arrays that store security rules.
- Open the driving cursor in a while loop. Fetch the data into rule array.
- Call `set_null_to_field()` to set fields to null when those fields' indicators are -1 in the rule array.
- Check if this is a second array fetch or greater, if yes, call `process_record()` to process the last record in last array fetch and the first record in current array fetch.
- Open a for loop
  - Call `process_record()` to process the current and last record.
- End of for loop
- Copy the last record in the current array fetch to last rule array. Since the last record of an array fetch hasn't been processed until compared to the first record of the next array fetch. However, with each new array fetch, the last record of the previous array fetch is overwritten. Thus here it needs to be copied.
- End of while loop.

#### Size\_rule\_array()

This function allocates memory for arrays that store security rules based on the maximum commit count set in table restart\_control table.

#### Set\_null\_to\_field()

This function loops through all the records in rule array and set a field to null when the field's indicator is -1.

#### Process\_record()

This function does the majority of the processing. The data from the driving cursor is ordered by dept, class, subclass, level1, level2, and level3 such that the department level rules are selected first, then the class level, etc. Also, all rules for a particular merchandise hierarchy will be grouped together and processed so that a single security rule will be decided for that particular hierarchy. When multiple records do occur at the same level, the logical OR will be used to determine whether to grant update/select privileges.

- Compare the user/functional area of the current record and the last record :
  - If it isn't new:
    - Compare the hierarchy of the current record and the last record:

- If it isn't new, call `logical_or_indicators()` to update the current record's select and update indicators according to the logical 'OR' between the current and last records' indicators.
- If it is new, call `update_array()` to blow security rule down to the SKU level according to the last record rule.
- If it is new:
  - Call `update_array()` to blow security rule down to the item level according to the last record rule.
  - Call `write_array()` to output the security rules of last record's user/functional area (down to item level) to SQL load file.
  - Call `retex_force_commit()` to set book mark in the `restart_bookmark` table.

#### `Logical_or_indicators()`

This function updates the input current record's select and update indicators according to the logical 'OR' between the input current and last records' indicators. For example, if the current record's select indicator is 'N', the last record's select indicator is 'Y', then the current record's select indicator is updated to 'Y'; If the current record's select indicator is 'N', the last record's select indicator is 'N', then the current record's select indicator is kept untouched('N'). If the current record's select indicator is 'Y', no matter what last record's select indicator is, the current record's select indicator is kept untouched('Y'). So does update indicator.

#### `Update_array()`

This function updates the item array according to the input security rule. There are five kinds of security rules. They are department, class, subclass, style and level1, level2, and level3 security rules.

- If the input rule is a department level security rule, then loop through the item array, for all the items within the department, set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a class level security rule, then loop through the item array, for all the items within the class, set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a subclass level security rule, then loop through the item array, for all the items within the subclass, set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a level1 level security rule, then loop through the item array, for all the items corresponding to the level1 item (the item, and all its children and grandchildren), set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set all three touched indicators of each processed row to 'Y'.
- If the input rule is a level2 level security rule, then loop through the item array, for all the items corresponding to the level2 item (the item, and all its children), set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set the level2 and level3 touched indicators of each processed row to 'Y'.
- If the input rule is a level3 level security rule, then loop through the item array and set the `select_inds` and `update_inds` equal to the input rule's `select_ind` and `update_ind`, respectively. Set the level3 touched indicator of each processed row to 'Y'.

#### `Write_array()`

This function writes out rows with touched indicator equals 'Y' in the SKU array to SQL load file.

#### `final();`

restart/recovery close

### I/O Specification

Each row of the output SQL load file `outputfilename.dat` corresponds to one record row in the `sec_user_prod_matrix` table. The format of the output file is as follows:

Column\_code;user\_id;item;select\_ind;update\_ind

Example:

PPRM;JOHN;10007986;N;N

PPRC;CLINTON;10001000;Y;N

PPRM;CLINTON;10007986;Y;Y

...

#### Technical Issues

N/A

## Stock count shrinkage update [stkdlly]

### Design Overview

This program processes the 'Unit & Dollar' type of stock count that the user has submitted for processing for the stock ledger. The main functions are to calculate actual shrinkage amount that will be used to correct the book stock value on the stock ledger and to calculate a shrinkage rate.

For a subclass/location that a stock count is to be processed, this program calculates the book stock value as of the date of the stock count, by using the beginning of the month stock value from MONTH\_DATA and summing the transaction data from DAILY\_DATA table since the beginning of the month to the date of stock count. It then compares the book stock value to the actual stock value stored on STAKE\_PROD\_LOC table to calculate the actual shrinkage amount.

An inter-stock-count shrinkage amount (i.e. inter\_stocktake\_shrink\_amount, stored on MONTH\_DATA table), which is an shrinkage amount that has been accumulated since the last time a stock count is taken, is added to the difference between the actual stock and book stock to come up with the actual shrinkage amount. Note that depending on the system option, budget shrink indicator, is set to 'Y' or 'N', the inter-stock-count shrinkage amount had been accumulated with either the budgeted shrinkage amount or the actual shrinkage amount that had been included in the book stock calculation by the stock ledger monthly process (salmth) since the last stock count.

| TABLE            | INDEX | SELECT<br>S | INSERT | UDPAT<br>E | DELET<br>E |
|------------------|-------|-------------|--------|------------|------------|
| SYSTEM_OPTIONS   | No    | Yes         | No     | No         | No         |
| SYSTEM_VARIABLES | No    | Yes         | No     | No         | No         |
| DAILY_DATA       | No    | Yes         | No     | No         | No         |
| STAKE_PROD_LOC   | No    | Yes         | No     | Yes        | No         |
| STAKE_HEAD       | No    | Yes         | No     | No         | No         |
| HALF_DATA_BUDGET | No    | Yes         | No     | No         | No         |
| MONTH_DATA       | No    | Yes         | No     | Yes        | No         |
| HALF_DATA        | No    | No          | No     | Yes        | No         |
| WEEK_DATA        | No    | No          | No     | Yes        | No         |

### Stored Procedures / Shared Modules (Maintainability)

**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.

### Input Specifications

#### Driving Cursor

```
EXEC SQL DECLARE c_stocktake CURSOR FOR
 SELECT s.cycle_count,
 spl.dept,
```

```

 spl.class,
 spl.subclass,
 spl.store,
 spl.wh,
 TO_CHAR(s.stocktake_date, 'DDMMYYYY'),
 TO_NUMBER(TO_CHAR(s.stocktake_date, 'DD')),
 TO_NUMBER(TO_CHAR(s.stocktake_date, 'MM')),
 TO_NUMBER(TO_CHAR(s.stocktake_date, 'YYYY')),
 NVL(spl.total_cost,0) + NVL(spl.adjust_cost,0),
 NVL(spl.total_retail,0) + NVL(spl.adjust_retail,0),
 ROWIDTOCHAR(spl.rowid),
 ';' || TO_CHAR(spl.dept) ||
 ';' || TO_CHAR(spl.store) ||
 ';' || TO_CHAR(spl.class)
FROM stake_prod_loc spl, stake_head s, v_restart_dept rv
WHERE s.stocktake_date >= to_date(:fdom_date,'DDMMYYYY')
 AND s.stocktake_date <= to_date(:next_eom_date,'DDMMYYYY')
 AND spl.cycle_count = s.cycle_count
 AND spl.processed = 'P'
 AND rv.driver_name = :ora_restart_driver_name
 AND rv.driver_value = spl.dept
 AND rv.num_threads = :ora_restart_num_threads
 AND rv.thread_val = :ora_restart_thread_val
 AND (spl.dept > NVL(:ora_restart_dept, -999) OR
 (spl.dept = :ora_restart_dept AND
 (spl.class > :ora_restart_class OR
 (spl.class = :ora_restart_class AND spl.store >
 :ora_restart_store))))
ORDER BY spl.dept,
 spl.class,
 spl.store;

```

### Scheduling Considerations

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: DEPT

## Purge stock count [stkprg]

### Design Overview

The function of this program is to delete records from the cycle count tables for stock takes with a STAKE\_HEAD.STOCKTAKE\_DATE that is less than the SYSTEM\_VARIABLES.LAST\_EOM\_DATE. The program works through stake\_head looking for out of date records and then deletes them along with their child records from the stake\_sku\_loc and stake\_prod\_loc tables.

Re-run:

If this program terminates normally, restart without recovery.

If this program terminates abnormally, restart without recovery.

Tables Affected:

| TABLE              | INDEX | SELECT | INSERT | UPDATE | DELETE |
|--------------------|-------|--------|--------|--------|--------|
| PERIOD             | No    | Yes    | No     | No     | No     |
| STAKE HEAD         | No    | Yes    | No     | No     | Yes    |
| STAKE_LOCATIO<br>N | No    | No     | No     | No     | Yes    |
| STAKE PRODUCT      | No    | No     | No     | No     | Yes    |
| STAKE_PROD_LO<br>C | No    | No     | No     | No     | Yes    |
| STAKE QTY          | No    | No     | No     | No     | Yes    |
| STAKE_SKU_LOC      | No    | No     | No     | No     | Yes    |

### Scheduling Constraints

Processing Cycle: AD HOC

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

```
EXEC SQL DECLARE get_old_records CURSOR FOR
 SELECT cycle_count
 FROM stake_head
 WHERE ((stocktake_date + :no_days_before_purge) <
 (select vdate from period))
 OR delete_ind = 'Y';
```



**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Stock count snapshots update [stkupd]

### Design Overview

| TABLE          | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------|-------|--------|--------|--------|--------|
| STAKE_SKU_LOC  | Yes   | Yes    | No     | No     | No     |
| PERIOD         | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS | No    | Yes    | No     | No     | No     |
| ITEM_MASTER    | No    | Yes    | No     | No     | No     |
| STAKE_HEAD     | No    | Yes    | No     | No     | No     |
| TSFDETAIL      | No    | Yes    | No     | No     | No     |
| SHIPSKU        | No    | Yes    | No     | No     | No     |
| SHIPMENT       | No    | Yes    | No     | No     | No     |
| TSFHEAD        | No    | Yes    | No     | No     | No     |
| ITEM_LOC       | Yes   | Yes    | No     | No     | No     |
| ITEM_LOC_SOH   | No    | Yes    | No     | No     | No     |

Indexes: STAKE\_DATE (stocktake\_date),  
ITEM\_LOC(item, loc)

Sets the stake\_sku\_loc snapshot\_on\_hand\_qty to the current item\_loc\_soh stock on hand for each store and warehouse for which a stocktake has been scheduled for today. Also fetches snapshot\_unit\_retail & snapshot\_unit\_cost the same way.

Re\_run:

If this program terminates normally/abnormally, the restart\_program\_status table must be updated by setting restart\_flag to 'Y' for current restart\_name, and schema.

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE/WH

### Restart Recovery

```
EXEC SQL DECLARE c_get_item_loc CURSOR FOR
 SELECT stake_sku_loc.item,
 stake_sku_loc.store,
 stake_sku_loc.wh,
 stake_sku_loc.cycle_count,
 ROWIDTOCHAR(stake_sku_loc.ROWID),
 item_master.pack_ind,
 ';' || (stake_sku_loc.item) ||
 ';' || TO_CHAR(stake_sku_loc.store) ||
 ';' || TO_CHAR(stake_sku_loc.wh)
```

```

FROM stake_sku_loc,
 item_master,
 stake_head,
 v_restart_store_wh rv
WHERE stake_sku_loc1.item = item_master.item
AND stake_sku_loc1.cycle_count =
 stake_head.cycle_count
AND TO_CHAR(stake_head.stocktake_date,
 'YYYYMMDD') = :ps_vdate
AND rv.driver_value = stake_sku_loc1.store
AND rv.driver_name = :restart_driver_name
AND rv.num_threads = :restart_num_threads
AND rv.thread_val = :restart_thread_val
AND (stake_sku_loc.item >
 NVL(:ps_restart_item, ' '))
OR
 (stake_sku_loc.item = :ps_restart_item
AND
 (stake_sku_loc.store > :ps_restart_store
OR
 (stake_sku_loc.store = :ps_restart_store
AND
 stake_sku_loc.wh > :ps_restart_wh))))
ORDER BY stake_sku_loc1.item, stake_sku_loc1.store, stake_sku_loc1.wh;

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Upload stock count results [stkupld]

### Design Overview

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

- if record exists on STAKE\_SKU\_LOC then update it
- if record doesn't exist on STAKE\_SKU\_LOC
- validate that item/location exists in system
- insert a record into STAKE\_SKU\_LOC
- insert stock take record into STAKE\_SKU\_LOC.
  - if record is a pack - update/insert information on STAKE\_SKU\_LOC for all component items

| TABLE          | SELECT | INSERT | UPDATE | DELETE |
|----------------|--------|--------|--------|--------|
| stake_qty      | No     | Yes    | No     | No     |
| stake_sku_loc  | No     | Yes    | Yes    | No     |
| item_loc_soh   | Yes    | No     | No     | No     |
| item_loc       | Yes    | No     | No     | No     |
| item_master    | Yes    | No     | No     | No     |
| wh             | Yes    | No     | No     | No     |
| stake_head     | Yes    | No     | No     | No     |
| stake_location | Yes    | Yes    | No     | No     |
| stake_prod_loc | Yes    | No     | No     | No     |
| v_packsku_qty  | Yes    | No     | No     | No     |
| system_options | Yes    | No     | No     | No     |

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

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

File layout will be verified by interface library routines:

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

Re-run:

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

### Scheduling Constraints

Processing Cycle: PHASE 3 (Daily)

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

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

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

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

The `commit_max_ctr` field should be set to prevent excessive rollback space usage and to reduce the overhead of file I/O. The recommended commit counter setting is 10,000 records (subject to change based on experimentation).

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

### Program Flow

N/A

### Shared Modules

`validate_all_numeric`: interface library function.  
`validate_all_numeric_signed`: interface library function.  
`valid_date`: interface library function.

**Function Level Description**

init():

initialize restart recovery

open input file

- file should be specified as input parameter to program

declare final output filename (used in restart\_write\_file logic)

open reject file ( as a temporary file for restart )

- file should be specified as input parameter to program

call restart\_file\_init logic

assign application image array variables- line counter (g\_l\_rec\_cnt), reject counter (g\_l\_rej\_cnt),  
cycle\_count, stocktake date

if fresh start (l\_file\_start = 0)

read file header record (get\_record)

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

validate file type = 'STKU'

else fseek to l\_file\_start location

validate head (validate\_head())

validate cycle count id & cycle count date are valid on stake\_head

---

process()

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

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

for loop to process all records read

copy input detail structure elements to stake\_sku\_loc structure elements

validate elements (validate\_detail())

if non-fatal error occurs write detail structure to reject file (write\_to\_rej\_file) and  
continue at the top of the for-loop

update stake\_sku\_loc

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

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

insert to stake\_sku\_loc (if display pack also insert component items)

end loop for loop to process individual records

insert structure of arrays (for valid record counter) into stake\_sku\_loc

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

restart write file function

if end of file reached then break from while loop

end outer loop to read from file

restart commit final

---

validate\_head()

if file type != 'STKU' then fatal file type error

copy stocktake\_date into variable

    nullpad stocktake\_date

copy loc\_type into variable ( value will always be warehouse 'W')

    nullpad stocktake\_dat

    nullpad loc\_type

copy loc\_value into variable

    nullpad loc\_value

copy store\_value, wh\_value, and loc\_value into variables ( store will always be -1)

get cycle count for location and stocktake\_date.

validate cycle count.

---

---

validate\_detail()

if record type != FDETL then fatal file layout error

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

    nullpad all fields

    left shift item and qty

    check that store and qty are all numeric

    place decimal in qty field

---

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

- write out rejected record to the reject file using write\_to\_rej\_file functionn, pass pointer to detail record structure, number of bytes in structure, and reject file pointer

**I/O Specification**Input File

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

| Record Name | Field Name                  | Field Type | Description                          |
|-------------|-----------------------------|------------|--------------------------------------|
| File Header | file type record descriptor | Char(5)    | hardcode 'FHEAD'                     |
|             | file line identifier        | Number(10) | Id of current line being processed., |

|                       |                                |                                 |                                                                     |
|-----------------------|--------------------------------|---------------------------------|---------------------------------------------------------------------|
|                       |                                |                                 | hardcode<br>'000000001'                                             |
|                       | file type                      | Char(4)                         | hardcode 'STKU'                                                     |
|                       | file create date               | Date(14)<br>YYYYMMDD<br>DHHMISS | date written by<br>convert program                                  |
|                       | stocktake_date                 | Date(14)<br>YYYYMMDD<br>DHHMISS | stake_head.stocktak<br>e_date                                       |
|                       | cycle count                    | Number(8)                       | stake_head.cycle_co<br>unt                                          |
|                       | loc_type                       | Char(1)                         | hardcode 'W'                                                        |
|                       | location                       | Number(10)                      | stake_location.wh                                                   |
|                       |                                |                                 |                                                                     |
| Transaction<br>record | file type record<br>descriptor | Char(5)                         | hardcode 'FDETL'                                                    |
|                       | file line<br>identifier        | Number(10)                      | Id of current line<br>being processed,<br>internally<br>incremented |
|                       | item type                      | Char(3)                         | hardcode 'ITM'                                                      |
|                       | item value                     | Number(25)                      | item id                                                             |
|                       | inventory<br>quantity          | Number(12,4)                    | total units or total<br>weight                                      |
|                       | location<br>description        | Char(30)                        | NULL                                                                |
|                       |                                |                                 |                                                                     |
| File trailer          | file type record<br>descriptor | Char(5)                         | hardcode 'FTAIL'                                                    |
|                       | file line<br>identifier        | Number(10)                      | Id of current line<br>being processed,<br>internally<br>incremented |
|                       | file record<br>count           | Number(10)                      | Number of detail<br>records.                                        |

### *Reject File*

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

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the restart\_write\_function should be called (passing



the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

#### Error File

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

#### Technical Issues

N/A

## Stock count stock on hand updates [stkvar]

### Design Overview

| Table          | Index | Select | Insert | Update | Delete |
|----------------|-------|--------|--------|--------|--------|
| STAKE_SKU_LOC  | no    | yes    | no     | yes    | no     |
| STAKE_CONT     | no    | yes    | no     | no     | yes    |
| STAKE_HEAD     | no    | yes    | no     | no     | no     |
|                |       |        |        |        |        |
| ITEM_LOC       | yes   | no     | no     | yes    | no     |
| ITEM_LOC_SOH   | yes   | no     | no     | yes    | no     |
|                |       |        |        |        |        |
|                |       |        |        |        |        |
|                |       |        |        |        |        |
|                |       |        |        |        |        |
| STAKE_PROD_LOC | yes   | no     | no     | yes    | no     |
| ITEM_MASTER    | no    | yes    | no     | no     | no     |
|                |       |        |        |        |        |

Indexes:

STAKE\_PROD\_LOC (dept, store, wh, data\_type)

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

The program is driven by STAKE\_CONT, in conjunction with STAKE\_SKU\_LOC where the ITEM, store, warehouse and cycle count on STAKE\_SKU\_LOC match those on STAKE\_CONT, and where STAKE\_CONT run\_type = 'A' (for adjustment).

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

A ITEM\_LOC\_SOH record is updated for every ITEM/store combination . The new stock on hand = item\_loc\_soh .stock\_on\_hand - snapshot\_stock\_on\_hand\_qty (from the STAKE\_SKU\_LOC table) + the physical count quantity on STAKE\_SKU\_LOC. In addition, the pack\_comp\_soh field is updated on ITEM\_LOC when a pack is processed for each component ITEM in the pack.

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

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

Re-run: If this program terminates normally, , ITEM\_LOC, STAKE\_QTY, ITEM\_LOC\_SOH, STAKE\_PROD\_LOC\_STOCK and STAKE\_CONT must be recovered prior to restart. If this program terminates abnormally, restart without recovery.

The syntax for invoking this program is:

stkvar *userid/pswd* [ *report\_name* ].

Here are some examples:

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

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: DEPT

### Restart Recovery

```
EXEC SQL DECLARE c_skus CURSOR FOR
 SELECT stake_cont.item,
 stake_cont.store,
 stake_cont.wh,
 stake_sku_loc.snapshot_on_hand_qty,
 nvl(stake_sku_loc.snapshot_in_transit_qty, 0),
 nvl(stake_sku_loc.snapshot_unit_cost, 0),
 nvl(stake_sku_loc.snapshot_unit_retail, 0),
 stake_sku_loc.physical_count_qty,
 stake_sku_loc.pack_comp_qty,
 nvl(stake_sku_loc.dept, 0),
 nvl(stake_sku_loc.class, 0),
 nvl(stake_sku_loc.subclass, 0),
 item_master.pack_ind,
 stake_head.stocktake_date,
 stake_head.stocktake_type,
 stake_head.cycle_count,
 ';' || TO_CHAR(stake_cont.item) ||
 ';' || TO_CHAR(stake_cont.store) ||
 ';' || TO_CHAR(stake_cont.wh)
 FROM stake_head,
 item_master,
 stake_sku_loc,
 stake_cont,
 v_restart_dept rv
 WHERE stake_cont.run_type = 'A'
 AND stake_cont.item = stake_sku_loc.item
 AND stake_cont.store = stake_sku_loc.store
```

```

AND stake_cont.wh = stake_sku_loc.wh
AND stake_cont.cycle_count = stake_sku_loc.cycle_count
AND stake_head.cycle_count = stake_cont.cycle_count
AND item_master.item = stake_cont.item
 AND item_master.item_level = item_master.tran_level
AND rv.driver_value = item_master.dept
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 (stake_cont.item > NVL(:ora_restart_item,-999) OR
 (stake_cont.item = :ora_restart_item AND
 (stake_cont.store > :ora_restart_store OR
 (stake_cont.store = :ora_restart_store AND
 stake_cont.wh > :ora_restart_wh)
)
)
)
ORDER BY stake_cont.item,
 stake_cont.store,
 stake_cont.wh;

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Stock count explode [stkxpld]

Stock Count Explode to ITEM/Location for Stores/Warehouses. This program combined stkxplst and stkxplwh to create a new program [stkxpld]

### Design Overview

| TABLE              | INDEX | SELECT | INSERT | UPDATE | DELETE |
|--------------------|-------|--------|--------|--------|--------|
| STAKE_SKU_LOC      | Yes   | No     | Yes    | No     | No     |
| STAKE_PROD_LO<br>C | Yes   | No     | Yes    | No     | No     |
| STAKE_LOCATIO<br>N | Yes   | Yes    | No     | No     | No     |
| STAKE_HEAD         | Yes   | Yes    | No     | No     | No     |
| STAKE_PRODUCT      | Yes   | Yes    | No     | No     | No     |

This program inserts into stake\_sku\_loc and stake\_prod\_loc tables for all valid item/store and department/store combinations for unit and dollar type stock counts, based on store, department, class and subclass information stored on stake\_location and stake\_product. The stock count can be requested at department, class or subclass level and exploded to Item/store from any of these levels. Inserts are made into stake\_prod\_loc now at subclass/store level.

Re-run:

If this program aborts and terminates normally or if this program terminates abnormally, restart after setting restart\_flag = 'Y' on the restart\_program\_status table for current restart\_name, schema, and thread\_val.

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: DEPT

### Restart Recovery

```
EXEC SQL DECLARE C_product CURSOR FOR
 SELECT h.cycle_count,
 s.dept,
 s.class,
 s.subclass,
 h.store_or_warehouse,
 ':' || TO_CHAR(h.cycle_count) ||
 ':' || TO_CHAR(s.dept) ||
 ':' || TO_CHAR(s.class) ||
 ':' || TO_CHAR(s.subclass)
```

```

FROM stake_product p,
 stake_head h,
 subclass s,
 v_restart_dept rv
WHERE h.cycle_count = p.cycle_count
 AND h.stocktake_date = (TO_DATE(:threshold_date,'DDMMYYYY'))
 AND h.stocktake_type = 'B'
 AND h.delete_ind = 'N'
 AND s.dept = p.dept
 AND s.class = NVL(p.class,s.class)
 AND s.subclass = NVL(p.subclass,s.subclass)
 AND rv.driver_value = p.dept
 AND rv.driver_name = :ps_restart_driver_name
 AND rv.num_threads = :pi_restart_num_threads
 AND rv.thread_val = :pi_restart_thread_val
 AND (h.cycle_count > NVL(:ps_restart_cycle_count, -999) OR
 (h.cycle_count = :ps_restart_cycle_count AND
 (s.dept > :ps_restart_dept OR
 (s.dept = :ps_restart_dept AND
 (s.class > :ps_restart_class OR
 (s.class = :ps_restart_class AND
 (s.subclass > :ps_restart_subclass)))))))
ORDER BY h.cycle_count,
 s.dept,
 s.class,
 s.subclass;

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

# Stock ledger extract [stlgdnld.pc]

## Design Overview

This program extracts stock ledger data at the item level. The program can extract data for a historic period or for the most current complete week. The program accepts an input file that determines whether the extract is a historic extract or a weekly extract.

The information extracted by this program includes: Regular Sales Value (retail, cost, and units), Promotional Sales (retail, cost, and units), Clearance Sales (retail, cost, and units), Sales Retail Excluding Vat, Custom Returns (retail, cost, and units), RTV (retail, cost, and units), Reclass In (retail, cost, and units), Reclass Out (retail, cost, and units), Permanent Markdown Value, Promotional Markdown Value, Clearance Markdown Value, Markdown Cancel Value, Markup Value, Markup Cancel Value, Stock Adj Value (retail, cost, and units), Received Value (retail, cost, and units), Transfer In Value (retail, cost, and units), Transfer Out Value (retail, cost, units), Freight Cost, Employee Disc Retail, Cost Variance, Wkroom Other Cost Sales, and Cash Disc Retail.

This information is intended for use by the Retek Predictive Planning application.

## Stored Procedures / Shared Modules (Maintainability)

### Input Specifications

#### Input file:

Will have the following format:

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

W[from date in the 'YYYYMMDD' format][to date in the 'YYYYMMDD' format]

Ex: H1996010120000222 or W199601012000022

### Driving Cursor:

The eow\_date filtering will be driven by the download type input parameter, whether the extract will occur for a historic date range (H) for the most current complete week (W).

Driving cursor for current complete week:

```
SELECT tdh.item,
 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')-tdh.tran_date),
 -1,to_date(:ps_eow_date,'YYYYMMDD'),0,
to_date(:ps_eow_date,'YYYYMMDD'),TO_DATE(tdh.tran_date,'YYYYMMDD')))/7)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'R',NVL(tdh.total_retail,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'R',NVL(tdh.total_cost,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'R',NVL(tdh.units,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'P',NVL(tdh.total_retail,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'P',NVL(tdh.total_cost,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'P',NVL(tdh.units,0),0),0)),
```

```

SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'C',NVL(tdh.total_retail,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'C',NVL(tdh.total_cost,0),0),0)),
SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'C',NVL(tdh.units,0),0),0)),
 SUM(DECODE(:pi_stkldgr_vat_incl_retl_ind,2,0,
 DECODE(tdh.tran_code,1,NVL(tdh.total_retail,0),0))+
 DECODE(tdh.tran_code,2,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,4,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,4,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,4,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,24,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,24,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,24,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,34,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,34,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,34,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,36,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,36,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,36,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,13,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,15,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,16,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,14,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,11,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,12,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,22,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,22,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,22,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,20,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,20,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,20,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,30,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,30,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,30,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,32,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,32,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,32,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,26,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,60,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,70,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,80,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,81,NVL(tdh.total_retail,0),0)),
 ':'||TO_CHAR(tdh.item)
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.item > NVL(:ps_restart_item, ' ')
 AND rv.driver_value = tdh.dept
 AND rv.num_threads = TO_NUMBER(:ps_num_threads)
 AND rv.thread_val = TO_NUMBER(:ps_thread_val)
GROUP BY tdh.item,
 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')-tdh.tran_date),
 -1,to_date(:ps_eow_date,'YYYYMMDD'),
 0,to_date(:ps_eow_date,'YYYYMMDD'),
 TO_DATE(tdh.tran_date,'YYYYMMDD')))/7))
ORDER BY tdh.item,
 to_date(:ps_eow_date,'YYYYMMDD')-(7*trunc((to_date(:ps_eow_date,'YYYYMMDD')-
 (DECODE(SIGN(TO_DATE(:ps_eow_date,'YYYYMMDD')-tdh.tran_date),
 -1,to_date(:ps_eow_date,'YYYYMMDD'),
 0,to_date(:ps_eow_date,'YYYYMMDD'),
 TO_DATE(tdh.tran_date,'YYYYMMDD')))/7));

```



**Driving cursor for historic data:**

```

SELECT tdh.item,
 DECODE(tdh.store,-1,'W','S'),
 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')- tdh.tran_date),
 -1,to_date(:is_to_date,'YYYYMMDD'),0, to_date(:is_to_date,'YYYYMMDD'),
 TO_DATE(tdh.tran_date,'YYYYMMDD')))/7)),

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

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

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

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

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

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

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

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

SUM(DECODE(tdh.tran_code,1,DECODE(tdh.sales_type,'C',NVL(tdh.units,0),0),0)),
 SUM(DECODE(:pi_stkldgr_vat_incl_retl_ind,2,0,
 DECODE(tdh.tran_code,1,NVL(tdh.total_retail,0),0))+
 DECODE(tdh.tran_code,2,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,4,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,4,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,4,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,24,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,24,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,24,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,34,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,34,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,34,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,36,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,36,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,36,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,13,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,15,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,16,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,14,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,11,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,12,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,22,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,22,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,22,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,20,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,20,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,20,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,30,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,30,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,30,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,32,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,32,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,32,NVL(tdh.units,0),0)),
 SUM(DECODE(tdh.tran_code,26,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,60,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,70,NVL(tdh.total_cost,0),0)),
 SUM(DECODE(tdh.tran_code,80,NVL(tdh.total_retail,0),0)),
 SUM(DECODE(tdh.tran_code,81,NVL(tdh.total_retail,0),0)),
 ':'||TO_CHAR(tdh.item)
FROM tran_data_history tdh,
 v_restart_dept rv

```

```

WHERE tdh.tran_date between (to_date(:is_from_date,'YYYYMMDD')-6) and
(to_date(:is_to_date,'YYYYMMDD'))
AND tdh.item > NVL(:ps_restart_item, ' ')
AND rv.driver_value = tdh.dept
AND rv.num_threads = TO_NUMBER(:ps_num_threads)
AND rv.thread_val = TO_NUMBER(:ps_thread_val)
GROUP BY tdh.item,
to_date(:is_to_date,'YYYYMMDD')-(7*trunc((to_date(:is_to_date,'YYYYMMDD')-
(DECODE(SIGN(TO_DATE(:is_to_date,'YYYYMMDD')- tdh.tran_date),
-1,to_date(:is_to_date,'YYYYMMDD'),
0,to_date(:is_to_date,'YYYYMMDD'),
TO_DATE(tdh.tran_date,'YYYYMMDD')))/7)),
DECODE(tdh.store,-1,'W','S'),
DECODE(tdh.store,-1,tdh.wh,tdh.store)
ORDER BY tdh.item;

```

## Output Specifications

### Output Files

The file outputted will be named stckldgr.dat.

### Output File Format:

| <i>Record Name</i> | <i>Field Name</i>          | <i>Field Type</i> | <i>Default Value</i> | <i>Description</i>                                  |
|--------------------|----------------------------|-------------------|----------------------|-----------------------------------------------------|
|                    | Item                       | Char(25)          |                      | Item                                                |
|                    | Location type              | Char(6)           |                      | Location type                                       |
|                    | Location                   | Char(20)          |                      | Location                                            |
|                    | Eow_Date                   | Char(8)           |                      | End of Week date of week for which data was derived |
|                    | Update ind                 | Char(1)           |                      | Update indicator                                    |
|                    | Regular_sales_retail       | Number(25)        |                      | Regular sales value (retail)                        |
|                    | Regular_sales_cost         | Number(25)        |                      | Regular sales value (cost)                          |
|                    | Regular_sales_units        | Number(17)        |                      | Regular sales value (units)                         |
|                    | Promo_sales_retail         | Number(25)        |                      | Promo sales value (retail)                          |
|                    | Promo_sales_cost           | Number(25)        |                      | Promo sales value (cost)                            |
|                    | Promo_sales_units          | Number(17)        |                      | Promo sales value (units)                           |
|                    | Clear_sales_retail         | Number(25)        |                      | Clearance sales value (retail)                      |
|                    | Clear_sales_cost           | Number(25)        |                      | Clearance sales value (cost)                        |
|                    | Clear_sales_units          | Number(17)        |                      | Clearance sales value (units)                       |
|                    | Sales_retail_excluding_vat | Number(25)        |                      | Sales value excluding vat (retail)                  |
|                    | Custom_returns_retail      | Number(25)        |                      | Custom returns value (retail)                       |
|                    | Custom_returns_cost        | Number(25)        |                      | Custom returns value (cost)                         |
|                    | Custom_returns_units       | Number(17)        |                      | Custom returns value (units)                        |
|                    | Rtv_retail                 | Number(25)        |                      | Return to Vendor value (retail)                     |
|                    | Rtv_cost                   | Number(25)        |                      | Return to Vendor value (cost)                       |

|                         |            |                                   |
|-------------------------|------------|-----------------------------------|
| Rtv_units               | Number(17) | Return to Vendor value (units)    |
| Reclass_in_retail       | Number(25) | Reclass In value (retail)         |
| Reclass_in_cost         | Number(25) | Reclass In value (cost)           |
| Reclass_in_units        | Number(17) | Reclass In value (units)          |
| Reclass_out_retail      | Number(25) | Reclass Out value (retail)        |
| Reclass_out_cost        | Number(25) | Reclass Out value (cost)          |
| Reclass_out_units       | Number(17) | Reclass Out value (units)         |
| Perm_markdown_value     | Number(25) | Permanent markdown value (retail) |
| Prom_markdown_value     | Number(25) | Promotion markdown value (retail) |
| Clear_markdown_value    | Number(25) | Clearance markdown value (retail) |
| Markdown_cancel_value   | Number(25) | Markdown cancel value             |
| Markup_value            | Number(25) | Markup value                      |
| Markup_cancel_value     | Number(25) | Markup cancel value               |
| Stock_adj_retail        | Number(25) | Stock adjustment value (retail)   |
| Stock_adj_cost          | Number(25) | Stock adjustment value (cost)     |
| Stock_adj_units         | Number(17) | Stock adjustment value (units)    |
| Received_retail         | Number(25) | Received value (retail)           |
| Received_cost           | Number(25) | Received value (cost)             |
| Received_units          | Number(17) | Received value (units)            |
| Tsf_in_retail           | Number(25) | Transfer In value (retail)        |
| Tsf_in_cost             | Number(25) | Transfer In value (cost)          |
| Tsf_in_units            | Number(17) | Transfer In value (units)         |
| Tsf_out_retail          | Number(25) | Transfer Out value (retail)       |
| Tsf_out_cost            | Number(25) | Transfer Out value (cost)         |
| Tsf_out_units           | Number(17) | Transfer Out value (units)        |
| Freight_cost            | Number(25) | Freight cost                      |
| Employee_disc_retail    | Number(25) | Employee disc (retail)            |
| Cost_variance           | Number(25) | Cost variance                     |
| Wkroom_other_cost_sales | Number(25) | Wkroom other sales (cost)         |
| Cash_disc_retail        | Number(25) | Cash disc (retail)                |

### 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.

#### format\_buffer():

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

**process():**

Will read the input file and Calls either fetch\_w\_process or fetch\_h\_process according to the input value from command line.

**fetch\_w\_process()**

The bulk work of traditional process function is done here. This function will employ the driving cursor specifically coded to fetch the most current end of week information from the TRAN\_DATA\_HISTORY table. This function will also loop through records returned by the driving cursor and write to the output file by calling write\_file(). Also within the for loop it calls the conversion function if the multi\_currency\_ind = 'Y'.

**fetch\_h\_process()**

Same as fetch\_w\_process, except this function will employ the driving cursor for historic data.

**get\_eow\_date()**

Fetches the most current end of week date from DATES\_SQL.GET\_EOW\_DATE.

**write\_file()**

This function will employ rtk\_print to write the information fetched from driving cursor to the output file.

**conversion()**

This function will call the function convert\_to\_primary(), located in the curconv.h library, and convert all 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 allocated 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().

**Scheduling Considerations**

This program can be run weekly in phase 4.

**Locking Strategy****Restart/Recovery**

The logical unit of work (LUW) in this program is item/location.

## Store add [storeadd]

### Design Overview

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

The batch program loops through each record on the store\_add table.

Also, it supports the replenishment system in RMS 9.0.

### Scheduling Constraints

Processing Cycle: Daily, Ad Hoc Phase

Scheduling Diagram: N/A

Pre-Processing: pccxt.pc pcdnld.pc

Post-Processing: slocrbld.pc

Threading Scheme: Table based processing, don't use multithreading.

### Restart/Recovery

Select ALL FIELDS from store\_add.

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

### Program Flow

N/A

### Function Level Description

#### init()

Declare restart variables

Get system variables (ELC indicator and pricing rule)

#### process()

Loop through store\_add table

Set "new" variable indicators

Insert into store table

Call Insert\_Pricing\_Zone

If elc\_ind = 'Y'

    Call Insert\_Cost\_Zones

end if;

If repl\_ind = 'Y'

    Call Copy\_Repl\_info

end if;

```
If copy_close_ind = 'Y'
 Call Copy_Close_Sched
End if;
If copy_dlvry_ind = 'Y'
 Call Copy_Dlvry_Sched
End if;
```

Call Insert\_Stock\_Loc\_Traits

Delete from store\_add

#### **Insert\_Pricing\_Zone()**

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

```
insert corporate pricing zone information
insert store pricing zone information
call Item_Zone_Price
if new_price_zone_ind = 'N'
 insert zone info for existing currency
else
 insert new zone info
 call Item_Zone_Price (to add appropriate record for the new zone)
```

#### **Insert\_Cost\_Zones()**

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

```
insert corporate cost zone information
insert store cost zone information
if new_cost_zone_ind = 'N'
 insert cost zone detail records
else
 insert new zone
```

#### **Item\_Zone\_Price()**

This function inserts records into the item\_zone\_price table for a new pricing zone after it's been created.

#### **Copy\_Store\_Items()**

This function calls the like\_store\_execute\_sql.copy\_store\_items package function, which copies all item/store records from the like\_store and inserts them for the new store.

#### **Copy\_Repl\_Info()**

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

#### **Copy\_Close\_Sched()**

This function copies all the location closed information from the selected like\_store which the close\_date are greater or equal to current and copies them into location\_closed and company\_closed\_except tables for the new store.

**Copy\_Dlvry\_Sched()**

This function copies all the location delivery schedules from the selected like\_store and copies them into the loc\_dlvry\_sched, loc\_dlvry\_sched\_days, and loc\_dlvry\_sched\_exc tables for the new store.

**Insert\_Stock\_Loc\_Traits()**

This function calls the stkledgr\_sql.stock\_ledger\_insert and loc\_traits\_sql.new\_org\_hier package functions, which insert records into the stock ledger and hierarchy tables.

**final()**

This function stops restart recovery.

**I/O Specification**

N/A

**Technical Issues**

N/A

## Supplier constraint scaling [supcnstr.pc]

### Design Overview

This new batch program will process all orders eligible for scaling during the nightly replenishment run. This program will run nightly after the replenishment order build program (rplbld.pc) and before the replenishment purge program (rplprg.pc). The purpose of this program will be to select all of the orders created by the replenishment programs which are eligible for scaling. Once selected, the program will serve as a wrapper program and send each order number into the supplier constraint scaling library to actually perform the scaling on the order.

The orders which will be eligible for scaling are as follows:

If due order processing was used, only orders with a written date of today, origin type = 0 (replenishment order), due order processing indicator = 'Y', due order indicator = 'Y' and a scale order to constraint indicator = 'Y' will be processed. This encompasses all due orders created by replenishment which have constraints associated with them.

If due order processing was not used, only orders with a written date of today, origin type = 0 (replenishment order), ord\_approve\_ind = 'Y', status = 'W'orksheet, due order processing indicator = 'N', due order indicator = 'Y', and a scale order to constraint indicator = 'Y' will be processed. This encompasses all approved orders created by replenishment which have constraints associated with them.

### Scheduling Constraints

#### Pre/Post Logic Description

Processing Cycle: PHASE III (daily)

Scheduling Diagram: After rplbld and before rplprg. This is not a required program. It needs to be run if order scaling is required in the batch run. If scaling will only be done online, this program need not be run.

Pre-Processing: N/A

Post-Processing: None

Threading Scheme: SUPPLIER

### Restart Recovery

Logical Unit of Work ( recommended Commit check points)

Driving Cursor

The Logic Unit of Work for this program is an order number.

This program supports Retek standard intermittent commit based on a commit counter. The recommended commit counter is 1000. Once a commit counter number of records have been successfully processed, it will force a commit. If a given record fails, the error will be posted to the standard error log for the batch module. If the error is a non-fatal type, all subsequent records



will continue to be processed. If a fatal error has occurred, all transactions from the last commit will be rolled back.

The driving cursor will need to be a union. The first section of the union will handle orders that are using due order processing. It should bring back every order where the `due_order_processing_ind` is 'Y' and the `scale_constr_ind` is 'Y'. The second section of the union will handle orders that are not using due order processing. It should bring back every order where the `due_order_processing_ind` is 'N' and the order is in worksheet status with `ord_approve_ind` = 'Y' and the `scale_constr_ind` is 'Y'.

The program should be threaded by supplier.

## Program Flow

Structure Chart

| TABLE        | SELECT | INSERT | UPDATE | DELETE |
|--------------|--------|--------|--------|--------|
| ORDHEAD      | Yes    | No     | Yes    | No     |
| ORD_INV_MGMT | Yes    | No     | Yes    | No     |
| PERIOD       | Yes    | No     | No     | No     |

## Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

Order Scaling Library

## Function Level Description

All database interactions required and error handling considerations

### Main

Follows Retek standards for the `main()` function. Logs on to the database, calls `init()`, `process()`, and `final()`.

### Init

Calls `retek_init()` to start up restart recovery and bring back the bookmark string in case of a restart. Fetches `vdate` from the `PERIOD` table.

### Process

This function is the controller for most of the processing in the program. It opens and fetches from the driving cursor according to Retek standards, using array processing and handling restart/recovery.

For each order returned by the driving cursor call the order scaling (`scaling_cnstr`) library which will perform the actual scaling of the order.

The scaling call will return success, fatal, and non-fatal errors. On success, the program will move on the next order. On fatal errors, the program will abort. Non-fatal errors will require

extra processing. `Handle_reject` will be called for all orders with non-fatal errors that were in approved status. `Update_oim` will be called for all orders returning non-fatal errors from scaling.

**Update\_oim**

A 2000 character error message will be populated by scaling for non-fatal errors. This error message will be placed in `ORD_INV_MGMT.SCALE_ISSUES` for the order being processed. `Ord_approve_ind` will be set to 'N'.

**Handle\_reject**

If the order was in approved status, it will be updated to worksheet status and its reject reason will be updated to 'SO' (scaling issues). There orders `orig_approval_id` and `orig_approval_date` will also be set to NULL.

**I/O Specification**

All files layouts input and output  
None.

**Database Objects**

Listing of all new tables, table changes, indexes, sequences, check constraints and foreign keys

**Testing Scenarios**

1. Test to ensure the proper order numbers are being picked up by the driving cursor, the order number is being passed into the scaling library, and the order is being scaled.
2. Test that when due order processing is being performed for a supplier, all due orders created by replenishment with scaling constraints are being scaled.
3. Test the when due order processing is NOT being performed for a supplier, all approved order created by replenishment with scaling constraints are being scaled.

## Supplier data amount repository [supmth]

### Design Overview

Accumulates SUP\_DATA amounts by department/supplier/transaction type and creates or updates one SUP\_MONTH row for each department/supplier combination. Based on the transaction type on SUP\_DATA, the following fields on SUP\_MONTH are updated:

- type 1 - purchases at cost (written for consignment sales and orders received at POS or online)
- type 2 - purchases at retail (written for consignment sales and orders received at POS or online)
- type 3 - claims at cost (written for claim dollars refunded on RTV orders )
- type 10 - markdowns at retail (net amount based on markdowns, markups, markdown cancellations and markup cancellations)
- type 20 - order cancellation costs (written for all supplier order cancellations)
- type 30 - sales at retail (written for consignment stock sales)
- type 40 - quantity failed (written for QC shipments with failed quantities)
- type 70 - markdowns at cost (net amount based on supplier cost markdowns)

Tables Affected:

| TABLE     | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-----------|-------|--------|--------|--------|--------|
| PERIOD    | No    | Yes    | No     | No     | No     |
| SUP_DATA  | No    | Yes    | No     | No     | No     |
| SUP_MONTH | No    | No     | Yes    | No     | No     |

Indexes: none

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: supmth\_post(), records are truncated from the sup\_data table.

Threading Scheme: DEPT

### Restart Recovery

```
SELECT sup_data.dept,
 TO_CHAR(sup_data.supplier),
 sup_data.tran_type,
 SUM(sup_data.amount),
 '||TO_CHAR(sup_data.dept)||'
 '||TO_CHAR(sup_data.supplier)
FROM v_restart_dept rv,
```

```

 sup_data
WHERE rv.driver_value = sup_data.dept
 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 (sup_data.dept > NVL(:ora_restart_dept, -999) OR
 (sup_data.dept = :ora_restart_dept AND
 (sup_data.supplier > :ora_restart_supplier)))
GROUP BY sup_data.dept,
 sup_data.supplier,
 sup_data.tran_type
ORDER BY sup_data.dept,
 sup_data.supplier,
 sup_data.tran_type;

```

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

N/A

**I/O Specification**

N/A

**Technical Issues**

N/A

## Zone security rebuild [szonrbld]

### Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Szonrbld.pc will handle the maintenance for the zone security data. Zones will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/pricing zone to define whether a user can select or update every zone covered by the defined rules. The functional document describes the architecture of the security features and how it works. If there are conflicting security definitions for a zone because a user is associated with more than one group, the user is, as expected, granted the capability. It will be multi-threaded by zone, and use restart\_recovery.

| TABLE                   | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-------------------------|-------|--------|--------|--------|--------|
| SEC_USER_GROUP          | No    | Yes    | No     | No     | No     |
| SEC_GROUP_ZONE_MATRIX   | No    | Yes    | No     | No     | No     |
| V_RESTART_ZONE_GROUP_ID | No    | Yes    | No     | No     | No     |
| SYSTEM_VARIABLES        | No    | Yes    | No     | No     | No     |
| SEC_USER_ZONE_MATRIX    | No    | No     | Yes    | No     | No     |

### Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters *szonrbld pre*, *szonrbld.pc* and *prepost.pc* with parameters *szonrbld post* in series.

Pre-Processing: Run prepost with parameters: *szonrbld pre*

Post-Processing: Run prepost with parameters: *szonrbld post*

Threading Scheme: Zone\_group\_id

### Restart Recovery

The logical unit of work for zone security rebuild will be the user-functional area (column\_code)-zone\_group\_id-zone\_id. Restart/recovery will be based on the user-functional area (column\_code)-zone\_group\_id-zone\_id. The commit\_max\_ctr field should be set to prevent excessive rollback space usage.

### Program Flow

N/A

**Shared Modules**

N/A

**Driving Cursor:**

```

SELECT u.user_id,
 z.column_code,
 z.zone_group_id,
 z.zone_id,
 MAX(z.select_ind),
 MAX(z.update_ind)
FROM sec_user_group u,
 sec_group_zone_matrix z,
 v_restart_zone_group_id v
WHERE u.group_id = z.group_id
 AND v.driver_value = z.zone_group_id
 AND v.num_threads = :pi_restart_num_threads
 AND v.thread_val = :pi_restart_thread_val
 AND (u.user_id > NVL(:ps_restart_user, '-999')
 OR (u.user_id = :ps_restart_user
 AND (z.column_code > :ps_restart_column_code
 OR (z.column_code = :ps_restart_column_code
 AND (z.zone_group_id > :ps_restart_zone_group_id
 OR (z.zone_group_id = :ps_restart_zone_group_id
 AND (z.zone_id > :ps_restart_zone_id)))))))
GROUP BY u.user_id, z.column_code, z.zone_id, z.zone_group_id;

```

**Function Level Description**

Main

Init()

- **Check SYSTEM\_VARIABLES.update\_loc\_sec\_ind.** If the indicator is not set then the program exit normally without further processing.
- Call retek\_init() to get restart-recover variables.

Process()

- Call size\_rule\_array() to allocate memory for arrays that store security rules.
- Open the driving cursor in a while loop. Fetch the data into rule array.
- Insert records in rule array into sec\_user\_zone\_matrix.
- Call restart\_commit.
  - End of while loop.

Size\_rule\_array()

This function allocates memory for arrays that store security rules based on the maximum commit count set in table restart\_control table. The rule array includes user\_id, column\_code, zone\_group\_id, zone\_id, select\_ind and update\_ind.

Insert\_array\_to\_table()

This function uses array insert to insert the contents in the rule array to table sec\_user\_zone\_matrix.

final():  
restart/recovery close

#### **I/O Specification**

N/A

#### **Technical Issues**

N/A

## Ticket output file [tcktdnld]

### Design Overview

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

### Scheduling Constraints

Processing Cycle: Ad Hoc (Daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

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

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

Driving Cursor:

```
SELECT tr.ticket_type_id,
 tr.item,
 tr.quantity,
 tr.loc_type,
 tr.location,
 tr.country_of_origin,
 tr.unit_retail,
 tr.multi_units,
 tr.multi_unit_retail,
 ROWIDTOCHAR(tr.rowid)
FROM ticket_request tr
ORDER BY 1, 2, 5;
```

### Program Flow

N/A



## Shared Modules

N/A

### Function Level Description

`init()` –

Functional details:

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

Technical details:

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

`process()` –

Functional Details:

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

Technical Details:

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

`ticket_pack()`

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

`ticket_item()`

This function should select all of the records from the `ticket_type_detail` table with the `ticket_type` from the `ticket_request` record. Detail records should be written out to the output file for each

detail record retrieved. Either item information or attribute information should be written to the output file. If the ticket item is to be written (fetched ticket item is not null) `get_ticket_item` is called to retrieve this appropriate information. If the attribute information is to be written (fetched attribute column is not null) then a function should be called to get the appropriate attribute information (`get_UDA`).

`get_ticket_item()`–

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

| TICKET ITEM | OUTPUT FILE VALUE                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| UOM         | Price per unit of measure from item_master.                                                                                                              |
| ITEM        | Retek ITEM value                                                                                                                                         |
| ITDS        | ITEM description (from item_master)                                                                                                                      |
| ITSD        | ITEM short description from the item_master table                                                                                                        |
| VAR         | The primary variant (ref_item) from the item_master table                                                                                                |
| DIF1        | Diff 1 value from item_master                                                                                                                            |
| DIF2        | Diff 2 value from item_master                                                                                                                            |
| WGHT        | Case weight from item_supp_country_dim table                                                                                                             |
| DEPT        | Department from item_master & department name from deps table                                                                                            |
| CLAS        | Class value from item_master table & class name from class table                                                                                         |
| SBCL        | Subclass from item_master table & subclass name from subclass table                                                                                      |
| RTPC        | Selling retail price from driving cursor (if available), otherwise from item_zone_price for item/store (use base zone value for warehouses).             |
| SRTP        | Suggested retail price (mfg_rec_retail) from item_master                                                                                                 |
| MUPC        | Multi-units and multi-unit retail from driving cursor (if available), otherwise from item_zone_price for item/store (use base zone value for warehouses) |
| SUPR        | Supplier from ordhead for most recent PO for the SKU.                                                                                                    |
| SUP1        | Supplier diff 1, from item_supplier                                                                                                                      |
| SUP2        | Supplier dif 2, from item_supplier.                                                                                                                      |
| STRE        | Store from driving cursor                                                                                                                                |
| WHSE        | Warehouse from driving cursor                                                                                                                            |
| COOG        | Country of origin from driving cursor if available, else from the last PO (see supplier).                                                                |

`get_UDA()`

This function should fetch the user defined attribute (UDA) value and description associated with the attribute value selected from the ticket detail table. The UDA description will be selected for the UDA and the ITEM from either the UDA\_item\_lov and the UDA\_value tables, the UDA\_item\_ff table or the UDA\_item\_date table. The UDA value will be written to the output file in the “value” location of the detail line.

`final()`

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

### I/O Specification

Output File:

| Record Name           | Field Name                  | Field Type               | Default Value              | Description                                                                                                                                                                |
|-----------------------|-----------------------------|--------------------------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| File Header           | File Type Record Descriptor | Char(5)                  | FHEAD                      | Identifies file record type                                                                                                                                                |
|                       | File Line Sequence          | Number(10)               |                            | Line number of the current file                                                                                                                                            |
|                       | File Type Definition        | Char(4)                  | TCKT                       | Identifies file as 'Print Ticket Requests'                                                                                                                                 |
|                       | File Create Date            | Date                     | create date                | date file was written by external system                                                                                                                                   |
| Transaction Header    | File Type Record Descriptor | Char(5)                  | THEAD                      | Identifies file record type                                                                                                                                                |
|                       | File Line Sequence          | Number(10)               |                            | Line number of the current file                                                                                                                                            |
|                       | ITEM                        | Char(25)                 |                            | ID number of the transaction level, non-pack item or the pack item                                                                                                         |
|                       | Ticket Type                 | Char(4)                  |                            | ID which indicates the ticket type to be printed                                                                                                                           |
|                       | Location Type               | Char(1)                  | S - Store<br>W - Warehouse | Identifies the type of location for which tickets will be printed                                                                                                          |
|                       | Location                    | Char (10)                |                            | number of the store or warehouse for which tickets will be printed                                                                                                         |
| Transaction Component | Quantity                    | Number(12,4)             |                            | the quantity of tickets to be printed                                                                                                                                      |
|                       | File Type Record Descriptor | Char(5)                  | TCOMP                      | Identifies file record type                                                                                                                                                |
|                       | File Line Sequence          | Number(10)               |                            | Line number of the current file                                                                                                                                            |
|                       | ITEM Quantity               | Char(25)<br>Number(12,4) |                            | ID number of the ITEM<br>Quantity of the component ITEM as part of the whole; if ITEM on the header record is a transaction level ITEM, the value in this field will be 1. |
| Transaction Detail    | File Type Record Descriptor | Char(5)                  | TDETL                      | Identifies file record type                                                                                                                                                |
|                       | File Line Sequence          | Number(10)               |                            | Line number of the current file                                                                                                                                            |
|                       | Detail                      | Number(10)               |                            | Sequential number assigned to the detail                                                                                                                                   |

|                     |                               |            |                     |                                                                                         |
|---------------------|-------------------------------|------------|---------------------|-----------------------------------------------------------------------------------------|
|                     | Sequence Number               | )          |                     | records                                                                                 |
|                     | Ticket Item                   | Char(4)    |                     | ID indicating the detail to be printed on the ticket                                    |
|                     | Attribute Description         | Char(40)   |                     | Description of the attribute (from the UDA Table)                                       |
|                     | Value                         | Char(100)  |                     | Detail to be printed on the ticket (i.e. REF_ITEM, Department Number, ITEM description) |
|                     | Supplement                    | Char(300)  |                     | Supplemental description to the Value (i.e. Department Name)                            |
| Transaction Trailer | File Type Record Descriptor   | Char(5)    | TTAIL               | Identifies file record type                                                             |
|                     | File Line Sequence            | Number(10) |                     | Line number of the current file                                                         |
|                     | Transaction Detail Line Count | Number(6)  | sum of detail lines | sum of the detail lines within a transaction                                            |
| File Trailer        | File Type Record Descriptor   | Char(5)    | FTAIL               | Identifies file record type                                                             |
|                     | File Line Sequence            | Number(10) |                     | Line number of the current file                                                         |

### Technical Issues

The program could be sped up by outer joining ticket\_type\_detail into the driving cursor, and avoiding the c\_ttd cursor which must be opened for each record in our fetch array.

## Tax rate POS download [tifposdn]

### Design Overview

This batch program processes records off of the TIF\_EXPLODE table, calculate a tax rate total for every item/department/site combination and write this total to a POS flat file. The changes defined by this document are required to adapt tifposdn to the new item dialogue.

The process function drives the processing of the program. Data will be read from the tif\_explode table and each record on the tif\_explode table will be processed into a transaction record on the output file. This file will contain separate transactions for every store listed on the tif\_explode table.

### Stored Procedures / Shared Modules (Maintainability)

No changes.

### Input Specifications

No changes.

### Output Specifications

Output file: Format will be as in table given below.

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

| Record | Variable               | Field Type | Default Value    | Description                   |
|--------|------------------------|------------|------------------|-------------------------------|
| FHEAD  | File record descriptor | CHAR(5)    | FHEAD            | Identifies file record type.  |
|        | Line number            | NUMBER(10) | 0000000001       | Identifies the line number    |
|        | Program descriptor     | CHAR(8)    | tifposdn         | Identifies the program.       |
|        | Create date            | CHAR(14)   | File create date | YYYYMMDDH H24MISS format      |
| THEAD  | File record descriptor | CHAR(5)    | THEAD            | Identifies Transaction Header |
|        | Line Number            | NUMBER(10) |                  | Sequential Line Number        |
|        | Transaction Number     | NUMBER(10) |                  | Transaction Number            |
|        | Store                  | NUMBER(10) |                  | Identifies store number       |
| TDETL  | File Record descriptor | CHAR(5)    | TDETL            | Identifies transaction detail |

|       |                        |            |       |                                                       |
|-------|------------------------|------------|-------|-------------------------------------------------------|
|       | Line Number            | NUMBER(10) |       | Sequential Line Number                                |
|       | Transaction Number     | NUMBER(10) |       | Transaction Number                                    |
|       | Department             | NUMBER(4)  |       | Department number.                                    |
|       | item                   | NUMBER(25) |       | Identifies the item.                                  |
|       | Tax Rate               | NUMBER(8)  |       | A new sales tax associated with the item ( % = /5)    |
|       | Start Date             | CHAR(8)    |       | Start date for the new tax rate. (in YYYYMMDD format) |
|       |                        |            |       |                                                       |
| TTAIL | File Record Descriptor | CHAR(5)    | TTAIL | Identifies transaction trailer.                       |
|       | Line Number            | NUMBER(10) |       | Sequential line number                                |
|       | Transaction Number     | NUMBER(10) |       | Transaction Number                                    |
|       | Detail Counter         | NUMBER(6)  |       | Number of detail records in transaction.              |
|       |                        |            |       |                                                       |
| FTAIL | Line Number            | NUMBER(10) |       | Sequential Line Number                                |
|       | Detail Line Counter    | NUMBER(6)  |       | Total Number of detail records processed.             |
|       |                        |            |       |                                                       |

### Function Level Description

Add a `#include <intrface.h>`

Remove all “EXEC SQL VAR”s.

Remove the following #defines, as they are not used : LEN\_TAX\_JURIS, LEN\_TAX\_TYPE, LEN\_GEO\_LEVEL, NULL\_GEO\_LEVEL, LEN\_GEO\_ID, NULL\_TAX\_RATE, LEN\_REC\_NO, LEN\_REC\_TYPE, NULL\_REC\_NO.

All variables referring to “sku” need to be renamed and resized to “item” specifications.

The C\_TIF\_EXPLODE cursor need to be changed...where it selects SKU, it needs to select ITEM. Also, the NVL(ps\_restart\_sku, -999) needs to be changed to NVL(ps\_restart\_item, “”).

Bring all code up to the latest standards.

Change the following line, which can be found closely following the fetch of the c\_tif\_explode cursor, :

```

 Ll_num_records_fetched = NUM_RECORDS_PROCESSED - ll_prev_total;
To :
 If(!(ll_num_records_fetched = NUM_RECORDS_PROCESSED - ll_prev_total)) break;

```

This will break out of the while loop if 0 records are brought back by the cursor.

### Scheduling Considerations

Processing Cycle: Phase 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: txrposdn.pc

Post-Processing: prepost.pc (to TRUNCATE tif\_explode table)

Threading Scheme: N/A

### Locking Strategy

No changes.

### Restart/Recovery

LOGICAL UNIT OF WORK = item/department/store combination

### Performance Considerations

No changes.

### Security Considerations

No changes.

### Appendix

Program Flow :



## Transportation upload [tranupld]

### Design Overview

The purpose of the TRANUPLD.pc batch program is to update the TRANSPORTATION table. This will allow the users to view and implement the transportation data online instead of manually viewing and inserting information.

This upload process will write both fatal and non-fatal errors to the IF\_ERRORS table rather than a reject file. The IF\_ERRORS table holds the program that caused the error (tranupld.pc in this case), error date (vdate), unit of work (Vessel ID, Voyage ID, Estimated Departure Date, Order No., Item, Container ID) and the error text.

This upload program has two business rules that need to be enforced. First, a PO/Item combination will be associated with a single Invoice. Second, a Container will be associated with a single Bill of Lading. These business rules will allow batch driven updates of the transportation table to function more smoothly and in a more time efficient manner.

### Upload Files

Four record types will be used in this program's upload files: FTRAN, DTRAN, DPOIT, and FTAIL. The FTRAN record will contain general header information. All pieces of the unique key, other than PO/Item combinations, will be found within DTRAN records. The remaining DTRAN information represents the details pertaining to the current vessel. Each DTRAN record must be followed by at least one DPOIT record. DPOIT records will store the remaining portion of the key (mainly, the PO/Item combination). Finally, the file will end with an FTAIL record that has a field holding the total number of records in the file.

For the purpose of this design a unit of data will be defined as the collection all the information in a given DPOIT record as well as the data in that of the DTRAN record to which it is associated. Depending on the acd\_code found in each DPOIT record, these units of data will be either added, deleted, or used to update a record in the transportation table.

### Program Flow

Read current record into appropriate data structure

- This format process should include checks which verify that all date fields and all numeric fields contain data of the appropriate type.

Validate current record

- Certain fields must contain values that already appear in other locations in the database. Ensure that these fields contain valid data.

Process current record

- Processing will only occur when the current record is of type DPOIT. Use the acd\_code to determine what action to perform on the transportation table.

### Tables Used



| TABLE          | SELECT | INSERT | UPDATE | DELETE |
|----------------|--------|--------|--------|--------|
| TRANSPORTATION | Yes    | Yes    | Yes    | Yes    |
| IF_ERRORS      | No     | Yes    | No     | Yes    |
| PERIOD         | Yes    | No     | No     | No     |

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: PHASE 4 (may also be schedule ad hoc to run multiple times per day)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

### Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)  
A valid DTRAN record.

### Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage

TRANSPORTATION\_SQL.GET\_NEXT\_ID

This function will create a new Transportation ID for the TRANSPORTATION table.

### Function Level Description

All database interactions required and error handling considerations

*init()*:

- Open input file.
- Get vdate.
- Initialize restart/recovery.
- Get FTRAN and validate.

*file\_process()*:

- Format a record from the input file.
- Determine if the record is valid by calling the *validate\_record()* function.
- If the record is a valid DPOIT, call the *process\_record()* function to perform an insert, update or delete statement.
- Continue processing until an FTAIL record is found

*format\_record()*:

If the record is a 'DTRAN' record call the `format_dtran_record()` function.

If the record is a 'DPOIT' record call the `format_dpoit_record()` function.

If the record is a 'FTAIL' record call the `format_ftail_record()` function.

*format\_dtran\_record():*

Read dtran from input file. Update indicator values for null fields. Ensure that consolidator and candidate ind fields are not null. If the candidate field is null, populate it with an 'N'. Validate that all date fields and all numeric fields contain data of the appropriate type.

*format\_dpoit\_record():*

Read dpoit from input file. Update indicator values for null fields. Ensure that the rush ind, order and item are not null. If the rush ind is null, populate it with an 'N'. Validate that all date fields and all numeric fields contain data of the appropriate type.

*format\_ftail\_record():*

Read ftail from input file. Ensure that the actual number of records processed matched the stated number of records.

*validate\_record():*

If the record is a 'DTRAN' record call the `validate_dtran_record()` function.

If the record is a 'DPOIT' record call the `validate_dpoit_record()` function.

If the record is a 'FTAIL' record call the `validate_ftail_record()` function.

*validate\_tran\_record():*

- vessel\_id, voyage\_flt\_id and estimated\_depart\_date must be all have a value, or must all be null. If this condition is not met, reject the record.
- validate the consolidator by calling a function which will return true when the given consolidator = partner\_id on the partner table with partner\_type = 'CO'.
- validate the trans mode by calling a function which will return true when the given trans mode = code\_detail on the code\_detail table with code\_type = 'TRMO'.
- validate the vessel\_scac\_code by calling a function which will return true when the given vessel\_scac\_code = scac\_code on the scac table
- validate the lading port by calling a function which will return true when the given lading port = outloc\_id on the outloc table with outloc\_type = 'LP'.
- validate the discharge port by calling a function which will return true when the given discharge port = outloc\_id on the outloc table with outloc\_type = 'LP'.
- validate that the estimated departure date is <= actual arrival date <= delivery date.
- validate the container\_scac\_code by calling a function which will return true when the given container\_scac\_code = scac\_code on the scac table
- validate the freight type by calling a function which will return true when the given freight type = freight\_type on the freight\_type table
- validate the freight size by calling a function which will return true when the given freight size = freight\_size on the freight\_size table

*validate\_poit\_record():*

- validate the order number by calling a function which will return true when the given order\_no = order\_no on the ordhead table
- validate the item by calling a function which will return true when the given item = item on the item\_master table

- validate the order/item combination by first checking if the combination exists, as is, on the ordsku table. If it does not, then check that the given item is a parent or grandparent item for any item associated with the given order.
- validate the origin\_country\_id by calling a function which will return true when the origin\_country\_id = country\_id on the country table
- validate the consolidation\_country\_id by calling a function which will return true when the consolidation\_country\_id = country\_id on the country table
- validate the export\_country\_id by calling a function which will return true when the export\_country\_id = country\_id on the country table
- validate the status by calling a function which will return true when the given status = code\_type on the code table with code\_type = 'TRCO'.
- validate the packing type by calling a function which will return true when the given packing type = code on the code\_detail table with code\_type = 'PKMT'.
- validate the item quantity UOM by calling a function which will return true when the given item quantity UOM = UOM on the uom\_class table with uom\_class = 'QTY'.
- validate the carton quantity UOM by calling a function which will return true when the given carton quantity UOM = UOM on the uom\_class table with uom\_class = 'PACK'.
- validate the gross weight UOM by calling a function which will return true when the given gross weight UOM = UOM on the uom\_class table with uom\_class = 'MASS'.
- validate the net weight UOM by calling a function which will return true when the given net weight UOM = UOM on the uom\_class table with uom\_class = 'MASS'.
- validate the cubic uom by calling a function which will return true when the given cubic uom = uom on the uom\_class table with uom\_class = 'VOL'.
- validate the currency code by calling a function which will return true when the currency code = currencies on the currencies table.
- Check that the current record type is valid given the preceding record type. FTRAN records can only occur at the beginning of the file. DPOIT records may only follow DPOIT or valid DTRAN records. DTRAN records can only follow FTRAN or DPOIT records. If a DTRAN valid is rejected, all DPOIT records associated with it must also be rejected.
- Validate that if a carton quantity exists, then a carton quantity uom also exists. If a cubic exists, then a cubic uom exists. If an invoice amount exists, then a currency code exists. If a gross weight exists, then a gross weight uom exists. If a net weight exists, then a net weight uom exists. If any of these conditions are not satisfied then reject the record.

process\_record():

Since the TRANSPORTATION table is completely de-normalized we need to synthesize the multiple records on the input files into single records to insert or update into the table or find the proper where clause to delete from it. To determine whether or not a combination exists on the TRANSPORTATION table call the appropriate cursor (there are four of them, one with PO/Item, another with VVE,PO/Item, another Container,PO/Item and the last VVE,Container,PO/Item).

Validation must prevent the following unique key from being violated:

vessel\_id / voyage\_flt\_id / estimated\_depart\_date,  
order\_no / item,  
container\_id

If any value on one of the above lines is null, then all values on that line must also be null.

```

if the DPOIT record does exist then
 if the acd_code = 'A' then
 validate that if candidate indicator is "Y" then invoice id, invoice amt, invoice date,
 currency code, exchange rate, item quantity, and item quantity uom are not null.
 to determine if the unique key combination including PO/Item already exists in
 the TRANSPORTATION table.
 if the combination does not exist on the TRANSPORTATION table then
 build an insert statement with DTRAN and DPOIT data
 if the combination does already exist on the TRANSPORTATION table then
 write error
 continue on to the next DPOIT, DTRAN or FTAIL section of the file

if the acd_code = 'C' then
 validate that if candidate indicator is "Y" then invoice id, invoice amt, invoice
 date, currency code, exchange rate, item quantity, and item quantity uom are not
 null.

 if the DTRAN Vessel_id, Voyage_flt_id, Estimated_depart_date and
 Container_id fields are NULL then
 that will validate if the PO/Item combination already exists on the
 TRANSPORTATION table
 if the PO/Item combination already exists on the TRANSPORTATION
 table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item combination with all others NULL does not exist on the
 TRANSPORTATION table then
 write error
 continue on to the next DPOIT, DTRAN or FTAIL section of the
 file
 if the DTRAN Vessel_id, Voyage_flt_id, Estimated_depart_date fields are
 NULL and the Container_id field is not NULL then
 that will validate if the PO/Item, Container combination already exists
 on the TRANSPORTATION table with VVE NULL
 if the PO/Item, Container combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item, Container combination does not exist on the
 TRANSPORTATION table then
 that will validate if the PO/Item combination already exists on
 the TRANSPORTATION table with all others NULL
 if the PO/Item combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item combination does not exist on the
 TRANSPORTATION table then
 write error
 continue on to the next DPOIT, DTRAN or FTAIL
 section of the file
 if the DTRAN Container_id field is NULL and Vessel_id, Voyage_flt_id,
 Estimated_depart_date fields are not NULL then

```

```

 that will validate if the PO/Item, VVE combination with Container
 NULL, already exists on the TRANSPORTATION table
 if the PO/Item, VVE combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item, VVE combination does not exist on the
 TRANSPORTATION table then
 that will validate if the PO/Item with all others NULL,
 combination already exists on the TRANSPORTATION table
 if the PO/Item combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item combination does not exist on the
 TRANSPORTATION table then
 write error
 continue on to the next DPOIT, DTRAN or FTAIL section of the
 file
if the DTRAN Vessel_id, Voyage_flt_id, Estimated_depart_date fields are not
NULL and the Container_id field is not NULL then
 that will validate if the PO/Item, VVE, Container combination already
 exists on the TRANSPORTATION table
 if the PO/Item, VVE, Container combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item, VVE, Container combination does not exist on the
 TRANSPORTATION table then
 that will validate if the PO/Item, Container, VVE NULL,
 combination already exists on the TRANSPORTATION table
 if the PO/Item, Container combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN and DPOIT data
 if the PO/Item, Container, VVE NULL combination does not
 exist on the TRANSPORTATION table then
 that will validate if the PO/Item, VVE, Container NULL
 combination already exists on the TRANSPORTATION
 table
 if the PO/Item, VVE, Container NULL combination
 already exists on the TRANSPORTATION table then
 build an update statement with DTRAN and
 DPOIT data
 if the PO/Item, VVE, Container NULL combination
 does not exist on the TRANSPORTATION table then
 that will validate if the PO/Item, VVE NULL,
 Container NULL combination already exists on
 the TRANSPORTATION table
 if the PO/Item, VVE NULL, Container NULL
 combination already exists on the
 TRANSPORTATION table then
 build an update statement with DTRAN
 and DPOIT data

```

if the PO/Item, VVE NULL, Container NULL  
combination does not exist on the  
TRANSPORTATION table then  
write error  
continue on to the next DPOIT, DTRAN  
or FTAIL section of the file

if the acd\_code = 'D'elele then  
to see if the Vessel/Voyage/ETD/PO/Item/Container/BL/Invoice combination  
exists on the TRANSPORTATION table  
if the Vessel/Voyage/ETD/PO/Item/Container/BL/Invoice combination exists on  
the TRANSPORTATION table.  
build a delete statement with DTRAN and DPOIT data  
if the combination does not exist on the TRANSPORTATION table then  
write error  
continue on to next DTRAN or FTAIL section of the file

### I/O Specification

All files layouts input and output

The following file formats will be used:

Header - DTRAN - Vessel/Voyage/ETD/Container/BL/Invoice  
Detail - PO/Item

File Format – Vessel/Voyage/ETD/Container/BL/Invoice

### Key

*Italicized field names must be included in the input file.*

**Bold field names are part of the primary key**

| Record Name | Field Name           | Field Type      | Default Value | Description                                                                          |
|-------------|----------------------|-----------------|---------------|--------------------------------------------------------------------------------------|
| FTRAN       | Record descriptor    | Char(5)         | FTRAN         | File head marker                                                                     |
|             | Line id              | Char(10)        | 0000000001    | Unique line id                                                                       |
|             | File type definition | Char(4)         | <b>TRUP</b>   | Identifies program as tranupld                                                       |
|             | File create date     | Char(8)         | Current date  | YYYYMMDD format                                                                      |
| DTRAN       | Record descriptor    | Char(5)         | DTRAN         | Vessel, Voyage, ETD, Container, BL, Invoice File head                                |
|             | Line id              | Char(10)        |               | Unique line id                                                                       |
|             | <i>Consolidator</i>  | <i>Char(10)</i> |               | <i>Identifies the Consolidator. Validated against PARTNER table with type = 'CO'</i> |
|             | <b>Vessel ID</b>     | <b>Char(20)</b> |               | <b>Identifies the Vessel</b>                                                         |

|  |                              |                 |                  |                                                                                                                         |
|--|------------------------------|-----------------|------------------|-------------------------------------------------------------------------------------------------------------------------|
|  | <b>Voyage ID</b>             | <b>Char(10)</b> |                  | <b>Identifies the Voyage or Flight ID</b>                                                                               |
|  | <b>Estimated Depart Date</b> | <b>Char(8)</b>  |                  | <b>YYYYMMDD format</b>                                                                                                  |
|  | Shipment Number              | Number(10)      |                  | Identifies an outside Shipment number                                                                                   |
|  | Actual Arrival Date          | Char(8)         |                  | YYYYMMDD format                                                                                                         |
|  | Trans Mode                   | Char(6)         |                  | Identifies the type of transportation being used. Valid values are found in the TRMO Code Type on the CODE_DETAIL table |
|  | Vessel SCAC Code             | Char(6)         |                  | Customs defined ID for the Vessel. Validated against SCAC table.                                                        |
|  | Estimated Arrival Date       | Char(8)         |                  | YYYYMMDD format                                                                                                         |
|  | Lading Port                  | Char(5)         |                  | Identifies the Lading Port. Validated against OUTLOC with type = 'LP'                                                   |
|  | Discharge Port               | Char(5)         |                  | Identifies the Discharge Port. Validated against OUTLOC with type = 'DP'                                                |
|  | Service Contract Number      | Char(15)        |                  | Identifies the outside Service Contract Number                                                                          |
|  | <b>Container id</b>          | <b>Char(20)</b> |                  | <b>Identifies the Container</b>                                                                                         |
|  | <b>Container SCAC code</b>   | Char(6)         |                  | Customs defined id for the container. Validated against SCAC table                                                      |
|  | Delivery Date                | Char(8)         |                  | YYYYMMDD format                                                                                                         |
|  | Seal id                      | Char(15)        |                  | Customs defined id for the container's seal                                                                             |
|  | Freight Type                 | Char(6)         |                  | Code that identifies the container type. Validated against the FREIGHT_TYPE table.                                      |
|  | Freight Size                 | Char(6)         |                  | Code that identifies the container size. Validated against the FREIGHT_SIZE table.                                      |
|  | In Transit No.               | Char(15)        |                  | External transit number                                                                                                 |
|  | In Transit Date              | Char(8)         |                  | YYYYMMDD format                                                                                                         |
|  | BL/AWB id                    | Char(15)        |                  | Identifies the Bill of Lading or Air Way Bill                                                                           |
|  | Candidate Ind                | Char(1)         | Defaulted to 'N' | Identifies a complete Transportation record. Valid values are 'Y' and 'N'                                               |

| DPOIT | Record descriptor        | Char(5)           | DPOIT                   | Order/Item detail info                                                                                 |
|-------|--------------------------|-------------------|-------------------------|--------------------------------------------------------------------------------------------------------|
|       | Line id                  | Char(10)          |                         | Unique file line id                                                                                    |
|       | ACD_Code                 | Char(1)           |                         | Determines which process to perform 'A'dd, 'C'hange, 'D'elete.                                         |
|       | <i>Rush Ind</i>          | <i>Char(1)</i>    | <i>Defaulted to 'N'</i> | <i>Identifies whether or not the item should be on a 'Rush' delivery. Valid values are 'Y' and 'N'</i> |
|       | <b>Order number</b>      | <b>Number(8)</b>  |                         | <b>Internal Retek order no</b>                                                                         |
|       | <b>Item</b>              | <b>char(25)</b>   |                         | <b>Internal Retek Item</b>                                                                             |
|       | Invoice id               | Char(30)          |                         | Identifies the Commercial Invoice                                                                      |
|       | Invoice date             | Char(8)           |                         | YYYYMMDD format                                                                                        |
|       | Currency Code            | Char(3)           |                         | Currency that the Currency Amount is reported in. Validated against CURRENCIES table.                  |
|       | Exchange Rate            | Number(20)        |                         | The exchange rate back to the primary currency (10 implied decimals)                                   |
|       | Invoice amt              | Number(20)        |                         | Amount charged by supplier for the PO/Item. (4 implied decimal places)                                 |
|       | Origin Country id        | Char(3)           |                         | Identifies where the PO/Item was made                                                                  |
|       | Consolidation Country id | Char(3)           |                         | Identifies where the PO/Items were consolidated                                                        |
|       | Export Country id        | Char(3)           |                         | Identifies where the PO/Items where shipped from                                                       |
|       | Status                   | Char(6)           |                         | Identifies the PO/Item status. Valid values are found in the TRCO Code Type on CODE_DETAIL             |
|       | Receipt ID               | Char(30)          |                         | Identifies the external receipt number                                                                 |
|       | FCR id                   | Char(15)          |                         | Identifies the Freight Cargo Receipt id                                                                |
|       | FCR date                 | Char(8)           |                         | YYYYMMDD format                                                                                        |
|       | Packing Method           | Char(6)           |                         | Identifies the Packing Type (Hanging or Flat). Valid values are 'HANG' or 'FLAT'                       |
|       | Lot Number               | Char(15)          |                         | Identifies the Lot Number of the PO/Item                                                               |
|       | <b>Item Qty</b>          | <b>Number(12)</b> |                         | <b>Qty of Items (4 implied decimals)</b>                                                               |
|       | Item QTY UOM             | Char(4)           |                         | Identifies the UOM associated with the item quantity                                                   |
|       | <b>Carton QTY</b>        | <b>Number(12)</b> |                         | <b>Qty of Cartons (4 implied</b>                                                                       |



|       |                   |            |       |                                                                                 |
|-------|-------------------|------------|-------|---------------------------------------------------------------------------------|
|       |                   |            |       | decimals)                                                                       |
|       | Carton QTY<br>UOM | Char(4)    |       | Identifies the UOM associated<br>with the carton quantity                       |
|       | Gross WT          | Number(12) |       | Gross weight (4 implied<br>decimals)                                            |
|       | Gross WT<br>UOM   | Char(4)    |       | Identifies the UOM associated<br>with the gross weight                          |
|       | Net WT            | Number(12) |       | Net Weight (4 implied<br>decimals)                                              |
|       | Net WT UOM        | Char(4)    |       | Identifies the UOM associated<br>with the net weight                            |
|       | Cubic             | Number(12) |       | Cubic size (4 implied decimals)                                                 |
|       | Cubic UOM         | Char(4)    |       | Identifies the UOM associated<br>with the cubic size                            |
|       | Comments          | Char(256)  |       | User Comments                                                                   |
| FTAIL | Record type       | Char(5)    | FTAIL |                                                                                 |
|       | Line id           | Char(10)   |       | Unique file line id                                                             |
|       | #lines            | Number(10) |       | Total number of transaction<br>lines in file (not including<br>FHEAD and FTAIL) |

### Technical Issues

\*\* The ACD\_Code places the responsibility of the upload process on the Consolidator. They must know how each record should be processed. The ACD\_Code allows them to identify an 'A'dd, 'C'hange or 'D'etele.

## Transfer purge [tsfprg]

### Design Overview

The purpose of this module is to purge transfer records, deleting all rows from the transfer header and detail table based on the number of months of transfer history to be retained. The number of transfer history months to be retained is specified on system options. If the difference in the number of months between today and the date on which the transfer was closed is greater than or equal to the number of transfer history months, the header and detail record are purged.

To keep the warehouse management system and the host system aligned, a file containing all of the deleted transfers will be created. These transfers can then be sent to the warehouse management system to be deleted as well. The file will contain the shipment number, Location (DC), and an action type of 'D'elele. The shipment number will correspond to the Nautilus' distribution number.

Tables Affected:

| TABLE             | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-------------------|-------|--------|--------|--------|--------|
| PERIOD            | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS    | No    | Yes    | No     | No     | No     |
| TSFHEAD           | No    | Yes    | No     | No     | Yes    |
| TSFDETAIL         | No    | No     | No     | No     | Yes    |
| TSFDETAIL_CHRG    | No    | No     | No     | No     | Yes    |
| SHIPMENT          | No    | No     | No     | No     | Yes    |
| SHIPSKU           | No    | No     | No     | No     | Yes    |
| ORDCUST           | No    | No     | No     | No     | Yes    |
| SHIPITEM_INV_FLOW | No    | No     | No     | No     | Yes    |
| CARTON            | No    | No     | No     | No     | Yes    |
| APPT_HEAD         | No    | Yes    | No     | No     | Yes    |
| APPT_DETAIL       | No    | Yes    | No     | No     | Yes    |
| DOC_CLOSE_QUEUE   | No    | No     | No     | No     | Yes    |

### Function Level Description

init()

Call restart\_init().

Initialize arrays

Fetch tsf\_history\_months from system\_options and vdate from period.

process()

initialize tsfhead\_rec\_ctr = 0

tsfhead\_total\_rec = 0

shipment\_rec\_ctr = 0

shipment\_total\_rec = 0

tsfhead\_flag = 1

shipment\_flag = 1

open driving cursor  
while

- Loop through the records in the driving cursor feeding them into an array as they are fetched.

for :num fetch into tsfhead array

- When no more records are found, set a flag to false so that the loop will be exited at the end of the processing.

if (not found) tsfhead\_flag = 0;

- Determine the number of records processed this time through the loop.

tsfhead\_rec\_ctr = NUM\_RECORDS\_PROCESSED -

tsfhead\_total\_rec;

tsfhead\_total\_rec += tsfhead\_rec\_ctr;

- Delete from shipitem\_inv\_flow where for all of the tsf\_no fetched into the array

EXEC SQL FOR :pi\_tsf\_to\_process

delete from shipitem\_inv\_flow where tsf\_no = :tsf\_recs.tsf\_no;

- For all of the transfer numbers which were fetched into the array in the previous fetch, find the associated shipment numbers

for i = 0..tsfhead\_rec\_ctr loop

call del\_appts() to delete associated appointments

open cursor fetching all shipment numbers for the transfer  
number

while(1)

for :num fetch into shipment array

- When no more records are found, set a flag to false so that the loop will be exited at the end of the processing.

if (not found) shipment\_flag = 0;

- Determine the number of records processed this time through the loop.

shipment\_rec\_ctr =

NUM\_RECORDS\_PROCESSED -

shipment\_total\_rec;

shipment\_total\_rec += shipment\_rec\_ctr;

if !(shipment flag) break;

- Delete from shipsku and shipment for all of the shipment numbers in the array

EXEC SQL FOR :shipment\_rec\_ctr

delete from shipsku where shipment =

:shipment and distro\_no = tsf\_no

```

EXEC SQL FOR :shipment_rec_ctr
 delete from shipment where shipment =
 :shipment and not exists (select 1 from shipsku where
shipment = :shipment)

 if the to or from location type is a warehouse
 write_detail_to_file()
 end if;
end for

 if !(tsfhead flag) break;
end while

- Delete from ordcust, tsfdetail_chrg, tsfdetail, and tsfhead for all of the
 transfer number in the array.
EXEC SQL FOR :tsfhead_rec_ctr
 delete from ordcust where tsf_no = :tsf_no;

EXEC SQL FOR :tsfhead_rec_ctr
 delete from tsfdetail where tsf_no = :tsf_no;

EXEC SQL FOR :tsfhead_rec_ctr
 delete from tsfhead where tsf_no = :tsf_no;
end for
end while
close cursor

```

#### write\_detail\_to\_file()

This function will write the 2 files to download to the warehouse management system. One for stock orders (ie. transfers from the warehouse) and another file for transfers coming into the warehouse (ie. PO). If the from location of the transfer is a warehouse, then a stock order record will be written to the stock order file. If the to location of the transfer is a warehouse, then a record is written to the PO file.

#### final()

Call restart\_close function

#### Del\_appts()

Deletes records from appt\_detail, first saving distinct appt/loc combination into a local array that is dynamically sized based on the number of records to be deleted from appt\_head. Then array deletes records based on the array from appt\_head. Also deletes from doc\_close\_queue. Calls size\_appt\_array() to size the appt\_head delete array.

#### Size\_appt\_array()

Sizes the array used to hold appt\_head appt/loc info between deletes from appt\_detail and appt\_head.

## Input Specifications

### Driving Cursor:

```
SELECT /*+ INDEX(tsfhead pk_tsfhead) */
 th.tsf_no,
 th.to_loc,
 th.to_loc_type,
 th.from_loc,
 th.from_loc_type
FROM tsfhead th,
 v_restart_transfer rv
WHERE (th.status = 'D'
 OR (th.status = 'C'
 AND th.tsf_type <> 'SA'
 AND MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),th.close_date)
 >= :ol_tsf_history_mths))
 AND rv.driver_value = th.tsf_no
 AND rv.num_threads = :ora_restart_num_threads
 AND rv.thread_val = :ora_restart_thread_val
 AND th.tsf_no > NVL(:ora_restart_tsf_no,-999)
ORDER BY th.tsf_no;
```

## Output Specifications

### Stock Order 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 Identifier           | Number(10) | Sequential number Created by program. | ID of current line being created for output file. |
|                    | File Type Definition           | Char(4)    | TSFD                                  | Identifies the program                            |
|                    | File Create Date               | Char(14)   | create date                           | current date, formatted to 'YYYYMMDDHH24MISS'.    |
| 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            |
|                    | Action Type                    | Char(1)    | 'D' (hardcode) +1 for space           | 'A'dd, 'D'elete, 'M'odify                         |

|                      |                                   |                   |                                                |                                                                                                                                                                                |
|----------------------|-----------------------------------|-------------------|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                      | Location (DC)                     | Number (4)        | Tsfhead.from_lo<br>c +1 for space              | Code for the DC (future use).<br>This field will not be used by<br>Logistics, however it will allow<br>RMS to create a separate file for<br>each warehouse.                    |
|                      | Transaction<br>Date/Time          | YYYYMMDD<br>DHHMI | Period.vdate +1<br>for space                   | Date/Time created in RMS                                                                                                                                                       |
|                      | Distribution<br>Number            | Char(9)           | Shipment.shipm<br>ent +1 for space             | Unique identifier of the<br>distribution.                                                                                                                                      |
|                      |                                   |                   | 408 spaces                                     | Do not need to send whole file.<br>Just the shipment number, action<br>type, and Location(DC), which<br>will be used for sorting by<br>warehouse in the conversion<br>program. |
| Transactio<br>n Tail | File type record<br>descriptor    | Char(5)           | TTAIL                                          | Identifies file record type                                                                                                                                                    |
|                      | File Line<br>sequence             | Number(10)        | Specified by<br>external system                | Line number of the current file                                                                                                                                                |
|                      | Transaction<br>detail line count  | Number(10)        | Sum of detail<br>lines                         | Sum of the detail lines within a<br>transaction                                                                                                                                |
| File Trailer         | File Type<br>Record<br>Descriptor | Char(5)           | FTAIL                                          | Identifies file record type                                                                                                                                                    |
|                      | File Line<br>Identifier           | Number(10)        | Sequential<br>number<br>Created by<br>program. | ID of current line being created<br>for output file.                                                                                                                           |
|                      | File Record<br>Counter            | Number(10)        |                                                | Number of records/transactions<br>processed in current file (only<br>records between head & tail)                                                                              |

## Transfers to the DC file (ie. PO)

| Record<br>Name | Field Name              | Field Type | Default Value                         | Description                                             |
|----------------|-------------------------|------------|---------------------------------------|---------------------------------------------------------|
| File Header    | File Type<br>Descriptor | Char(5)    | FHEAD                                 | Identifies file<br>record type                          |
|                | File Line<br>Identifier | Number(10) | Nine leading<br>zeroes:<br>0000000001 | ID of current line<br>being processed by<br>input file. |
|                | File Type<br>Definition | Char(4)    | TDCH                                  | Identifies file as<br>'DC Header<br>download'           |
|                | File Create<br>Date     | Date       | create date                           | date file was<br>written by external<br>system          |

|                    |                             |              |                                       |                                                                                                                                                                 |
|--------------------|-----------------------------|--------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Transaction 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.                                                                                                               |
|                    | Action Type                 | Char (1)     | default to 'D' + 1 for space          | 'A'dd, 'D'ele, 'M'odify                                                                                                                                         |
|                    | Location (DC)               | NUMBER (4)   | shipment.location + 1 for space       | Code for the DC (future use)                                                                                                                                    |
|                    | Transaction Date/Time       | YYYYMMDDHHMI | Sysdate + 1 for space                 | Date/Time created on host                                                                                                                                       |
|                    | PO Number                   | VARCHAR (9)  | shipment + 1 for space                | Unique identifier of the PO.                                                                                                                                    |
|                    |                             |              | 48 spaces                             | Do not need to send whole file. Just the shipment number, action type, and Location(DC), which will be used for sorting by warehouse in the conversion program. |
| File Trailer       | File Type Record Descriptor | Char(5)      | FTAIL                                 | Identifies file record type                                                                                                                                     |
|                    | File Line Identifier        | Number(10)   | Sequential number Created by program. | ID of current line being created for output file.                                                                                                               |
|                    | File Record Counter         | Number(10)   |                                       | Number of records/transactions processed in current file (only records between head & tail)                                                                     |

### Scheduling Considerations

Processing Cycle: PHASE AD-HOC

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

## Tax rate POS download [txrposdn]

### Design Overview

This batch program processes records off of the geocode\_txcde, product\_taxcode, and tax\_rates tables, and writes all item/site combinations to the TIF\_EXPLODE table for tax rate processing to be sent to the POS flat file (created by tifposdn.pc). This program writes all item/category/store combinations that are affected by changes to these three tables by pulling the records from the tables where the download indicator is 'Y' and the start date less than or equal to (current date +2), where the end date is equal to (current date +2), or where the download indicator is 'Y', and the end date is equal to current date or (current date +1). After each record is processed, the download\_ind flag will be set to 'N'.

The changes made to this program are necessary to account for changes in the item dialogue.

### Stored Procedures / Shared Modules (Maintainability)

No changes.

### Input Specifications

No changes.

### Output Specifications

No changes.

### Function Level Description

Remove all "EXEC SQL VAR"s.

All variables referring to "sku" need to be renamed and redefined to "item" specifications.

All code needs to be brought up to current batch standards.

Remove the resize\_tif\_insert\_array\_function.

### Init()

Remove the c\_get\_seq\_no cursor and all associated functionality. Keep the pl\_seq\_no variable, however.

### Process\_ptc\_data() :

The C\_PTC\_DATA cursor needs to be changed to something similar to the following :

```
SELECT loc,
 Ptc.Item,
 Dept,
 Start_date - :ps_vdate,
 NVL((End_date - vdate), -9999),
 Decode(download_ind, 'Y', 1, 'N', 0)
FROM product_tax_code ptc,
 Item_loc,
```



```

 Item_master
WHERE loc_type = 'S'
 AND status = 'A'
 AND tran_level = item_level
 AND item_master.item = item_loc.item
 AND (item_master.item = ptc.item
 OR (item_master.item_parent = ptc.item
 AND item_level >= tran_level))
 AND ptc.dept IS NULL
 AND ((download_ind = 'Y'
 AND (start_date >= vdate
 OR end_date = vdate
 OR end_date = vdate + 1)
 OR (download_ind = 'N'
 AND end_date = vdate + 2))
 AND ((loc > NVL(ps_restart_store, -9999)
 OR (item_loc.loc = ps_restart_store
 AND ptc.item > ps_restart_item)))
UNION ALL
SELECT store,
 Ptc.item,
 Dept,
 Start_date - :ps_vdate,
 NVL((End_date - vdate), -9999),
 Decode(download_ind, 'Y', 1, 'N', 0)
FROM product_tax_code ptc,
 Store
WHERE ptc.item IS NULL
 AND ((download_ind = 'Y'
 AND (start_date >= vdate
 OR end_date = vdate
 OR end_date = vdate + 1)
 OR (download_ind = 'N'
 AND end_date = vdate + 2))
 AND ((loc > NVL(ps_restart_store, -9999)
 OR (store = ps_restart_store
 AND item > ps_restart_item)))
ORDER BY 1, 2, 3;

```

Currently, if the c\_ptc\_data cursor brings back 0 records, this program will access memory outside of the pa\_data arrays (ex. : if ll\_ctr = 0, then ps\_data.item[0 - 1] is accessed. This is bad.) Add functionality to prevent this from happening.

#### **Process\_gt\_data() :**

The C\_GT\_DATA cursor needs to be changed to something similar to the following :

```

SELECT loc,
 Item,
 Start_date - vdate,
 NVL((end_date - vdate), -9999),
 Decode(download_ind, 'Y', 1, 'N', 0)

```

```

FROM item_loc,
 Geocode_txcode gt,
 Geocode_store gs,
 Item_master
WHERE loc_type = 'S'
 AND loc = gs.store
 AND status = 'A'
 AND (item_master.item = item_loc.item
 OR (item_parent = item_loc.item
 AND item_level >= tran_level))
 AND gt.country_geocode_id = gs.country_geocode_id
 AND (gt.state_geocode_id = gs.state_geocode_id
 OR gs.state_geocode_id IS NULL)
 AND (gt.county_geocode_id = gs.county_geocode_id
 OR gs.geocode_id IS NULL)
 AND (gt.city_geocode_id = gs.city_geocode_id
 OR gs.city_geocode_id IS NULL)
 AND (gt.district_geocode_id = gs.district_geocode_id
 OR gs.district_geocode_id IS NULL)
 AND ((download_ind = 'Y'
 AND (start_date >= vdate
 OR end_date = vdate
 OR end_date = vdate + 1)
 OR (download_ind = 'N'
 AND end_date = vdate + 2))
 AND ((loc > NVL(ps_restart_store, -9999)
 OR (store = ps_restart_store
 AND item > ps_restart_item)));

```

Currently, if the c\_ptc\_data cursor brings back 0 records, this program will access memory outside of the pa\_data arrays (ex. : if ll\_ctr = 0, then ps\_data.item[0 - 1] is accessed. This is bad.) Add functionality to prevent this from happening.

#### **Process\_tr\_data() :**

The C\_TR\_DATA cursor needs to be changed to something similar to the following :

```

SELECT loc,
 Item,
 Start_date - vdate,
 NVL((end_date - vdate), -9999)
 Decode(download_ind, 'Y', 1, 'N', 0)
FROM item_loc,
 Tax_rates tr,
 Product_tax_code ptc,
 Item_master
WHERE loc_type = 'S'
 AND status = 'A'
 AND item_loc.item = ptc.item
 AND (item_master.item = ptc.item
 OR (item_parent = ptc.item
 AND item_level >= tran_level))

```

```

AND ptc.tax_jurisdiction_id = tr.tax_jurisdiction_id
AND ptc.tax_type_id = tr.tax_type_id
AND ((tr.download_ind = 'Y'
 AND tr.start_date >= :ps_vdate
 OR tr.end_date = :ps_vdate
 OR tr.end_date = :ps_vdate + 1))
OR (tr.download_ind = 'N'
 AND tr.end_date = :ps_vdate + 2))
AND ((ws.store > NVL(:ps_restart_store, -9999)
 OR (ws.store = :ps_restart_store
 AND ws.sku > :ps_restart_sku)))

```

Currently, if the c\_ptc\_data cursor brings back 0 records, this program will access memory outside of the pa\_data arrays (ex. : if ll\_ctr = 0, then ps\_data.item[0 -1] is accessed. This is bad.) Add functionality to prevent this from happening.

#### Fill\_insert\_array()

The C\_record\_exists cursor needs to have the WHERE clause changed from “WHERE sku = :is\_sku” to “WHERE item = :is\_item”. Also change “UNION” to “UNION ALL”.

Immediately inside the “if(!li\_record\_exists)” statement, the call to the resize\_tif\_insert\_array() function and the surrounding if statement need to be removed. Replace with a call to the NEXT\_TIFXPLD\_SEQ\_NO function to get the next seq\_no, passing in the pl\_seq\_no variable.

#### Post\_tif\_explode()

The INSERT into TIF\_EXPLODE needs to be changed to insert ‘item’ instead of ‘sku’.

#### Final()

Add functionality to free the memory that was allocated for the arrays.

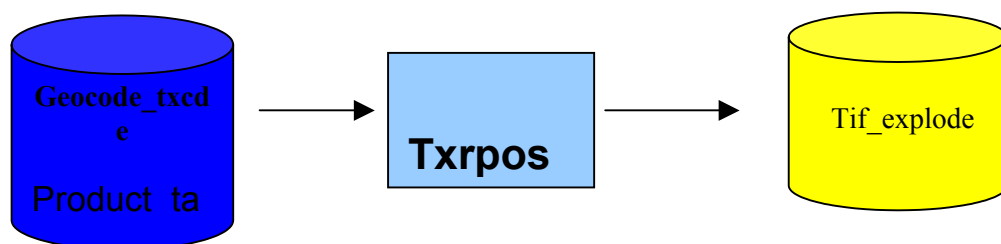
#### Scheduling Considerations

|                     |                                        |
|---------------------|----------------------------------------|
| Processing Cycle:   | Phase 4 (daily)                        |
| Scheduling Diagram: | N/A                                    |
| Pre-Processing:     | N/A                                    |
| Post-Processing:    | Run tifposdn.pc to create a flat file. |
| Threading Scheme:   | N/A                                    |

#### Restart/Recovery

LOGICAL UNIT OF WORK = item/store combinationAppendix

Program Flow :



## Tax rate upload [txrtupld]

### Design Overview

This program will be used to upload tax rates from an outside source into the tables into the RMS. The upload will provide the means to create new tax jurisdictions, tax codes, and tax rates. In addition to the ability to insert new tax codes and rates it will also be able to update the rates for existing tax codes.

Tax rates contain a rate, start date and end date(not required). Tax rates are associated to tax codes. A tax code can have many rates associated to it, however not more than one rate can be effective at one time. Thus when new rates are uploaded into the system for existing records, validation will occur to ensure that the start and end dates of the tax rates are not overlapping. If an overlap is found then the record will be written out to a reject file where it can be investigated further.

The input file will contain the tax jurisdiction, tax type, level and tax rates which will allow the batch programs to correctly update the tax rates to reflect any changes that have occurred.

Assumption: The tax codes that are in the input file will be the same tax codes in the Retek system. This program will not be responsible for matching, mapping, and/or converting the data from an outside source to the data that is present in the Retek system. Delete functionality will be carried out using the functionality of the end dates on the tax rates. This program will not physically delete any information off of the tables.

### Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: ad hoc

Scheduling Diagram:

Pre-Processing:

Post-Processing:

Threading Scheme:

### Restart Recovery

Logical Unit of Work ( recommended Commit check points )

Driving Cursor

### Program Flow

Structure Chart

### Main()

This function provided the structure for the program and outlines the flow of the program depending on when what function is being called.

- Init()
- Process()
- Final()

### **Init()**

- Initialize the restart variables that will be used from restart recovery through out the program.
- Open a temporary file that will be used to create the reject file when the program is completed.

### **Process()**

- In the beginning of this function create a loop to obtain all of the THEAD records that exist on the input file. Once the FTAIL is reached then exit out of the loop and continue on with the remaining processing.
- Call the function validate\_THEAD() this function will store all of the information in the THEAD line into local variables.
- Call the function tax\_jurisdiction ()
- Call the function tax\_codes()
- Call the function tax\_rates()

### **Validate\_record()-**

This function will check to ensure that the dates entered are valid dates on the calendar. Additional validation will occur to ensure that the date passed in are greater than or equal to today's date. If an invalid date is passed in , the record will be written out the reject file for further investigation.

### **tax\_jurisdiction()-**

This function checks to see if the current jurisdiction being processed exists currently on the tax\_jurisdictions table.

- If the tax\_jursidiction already exists in the system and does not need to be added.
- If the tax\_jurisdiction does not exist in the system ,it is a new tax\_jurisdiction and it needs to be added. This will be done by filling the insert\_tax\_jurisdictions array.

### **tax\_codes()-**

This function checks to see if the current tax code being processed exists currently on the tax\_codes table.

- If the tax code already exists in the system and does not need to be added.
- If the tax code does not exist in the system, it is a new tax code and it needs to be added into the system. This will be done by filling the insert\_tax\_codes array.

**tax\_rates()-**

This function checks to see if the current tax rate being process exists currently on the tax\_rates table.

- If it does exist then the tax rate already exists in the system and does not need to be added.
  - Call the function update\_tax\_rates
- If it does not exist in the system it is a new tax rate and it needs to be added into the system. This will be done by filling the insert\_tax\_rates array. This is done by calling the function fill\_tax\_rates\_array

**Update\_tax\_rates()**

- This function should check to see if there currently is an ongoing tax rate in the system.
  - If there is an ongoing rate then the first rate must be ended the day before the new rate is to start.
- Validate that the current tax rate's date ranges do not conflict with the other date ranges that are currently set up for the current tax code. If conflicts are found then the record will be written out to the reject file for further investigation.
- Call the function fill\_tax\_rates\_array

**Process\_record()**

- This function will insert the arrays the were created in the functions above. The arrays should be inserted into the tables in the following order:
  - Insert\_tax\_jurisdiction
  - Insert\_tax\_code
  - Insert\_tax\_rate

**Final()**

Close the reject file and exit out of the program successfully.

**Shared Modules**

Listing of all externally referenced functions and Stored procedures and description of usage

**Function Level Description**

All database interactions required and error handling considerations

**I/O Specification**

All files layouts input and output

| Record Name | Field Name           | Field Type | Default Value | Description              |
|-------------|----------------------|------------|---------------|--------------------------|
| FHEAD       | File head descriptor | Char(5)    | FHEAD         | Describes file line type |

|       |                              |             |            |                                                      |
|-------|------------------------------|-------------|------------|------------------------------------------------------|
|       | Line id                      | Char(10)    | 0000000001 | Sequential file line number                          |
|       | Gentran ID                   | Char(5)     | 'TXUP'     | Identifies which translation Gentran uses            |
|       | Current date                 | Char(14)    |            | File date in YYYYMMDDHH24 MISS format                |
| FDETL | File record descriptor       | Char(5)     | THEAD      | Start of an invoice transaction                      |
|       | Line id                      | Char(10)    |            | Sequential file line number                          |
|       | Tax Jurisdiction             | Number(8)   |            | Tax Jurisdiction                                     |
|       | Tax Jurisdiction Description | Char(40)    |            | Tax Jurisdiction description                         |
|       | Tax Level                    | Char(6)     |            | Tax level of the Jurisdiction                        |
|       | Tax Type                     | Char(6)     |            | Tax Type                                             |
|       | Tax Rate                     | Number(8,5) |            | Tax rate of the tax code                             |
|       | Start Date                   | Char (14)   |            | Start Date of the tax rate                           |
|       | End Date                     | Char (14)   |            | End Date of the tax rate                             |
| FTAIL | File record descriptor       | Char(5)     | FTAIL      | Marks end of file                                    |
|       | Line id                      | Char(10)    |            | Sequential file line number                          |
|       | Number of lines              | Number(10)  |            | Number of lines in file not counting FHEAD and FTAIL |

Mapping of the fields:

| Comets                                      | Retek                        |
|---------------------------------------------|------------------------------|
| Tax jurisdiction                            | Tax Jurisdiction             |
| Geocode Description                         | Tax Jurisdiction Description |
| Level                                       | Tax Level                    |
| 3 <sup>rd</sup> digit of the Tax item field | Tax Type                     |

## VAT rate maintenance [vatdixpl]

### Design Overview

Value Added Tax (VAT) is a tax imposed by some governments on goods that have realized an increase in value.

As with price zones and cost zones, individual stores belong to a particular VAT region. VAT regions are areas that contain VAT code and VAT rates information that applies to stores in that region and are based on VAT rates defined by the government. VAT codes can be set up by Retek users to identify a particular VAT rate percentage. Only one VAT rate can be active for any one VAT code.

VAT rates are stored at the ITEM level in Retek on the VAT\_ITEM table. On this table, records with a past active date serve as an audit trail to track what VAT code and VAT rate a particular ITEM has had or currently has. The VAT\_ITEM record with the most recent active date holds the ITEM's current VAT code. The VAT rate related to a particular VAT code is found on the VAT\_CODE\_RATES table. Records on VAT\_ITEM with a future active date functionally represent a pending change in the VAT code for that particular ITEM, effective on that date and reflecting the corresponding VAT rate associated with that VAT code on the VAT\_CODE\_RATES table.

Records on the VAT\_CODE\_RATES table functionally represent a change in the VAT rate for a particular VAT code. When a record exists on this table with an active date of tomorrow, this program updates the VAT rate on any future VAT\_ITEM records associated with that VAT code to the new VAT rate defined in the VAT\_CODE\_RATES table. Also, if the latest VAT\_ITEM record for any ITEM contains the VAT code that is changing, a new VAT\_ITEM record is inserted for that ITEM with an active date of tomorrow and the new VAT rate associated with the VAT code.

Tables Affected:

| TABLE          | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------|-------|--------|--------|--------|--------|
| VAT_CODE_RATES | Yes   | Yes    | No     | No     | No     |
| VAT_ITEM       | Yes   | Yes    | Yes    | Yes    | No     |

### Scheduling Constraints

Processing Cycle: Phase I

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

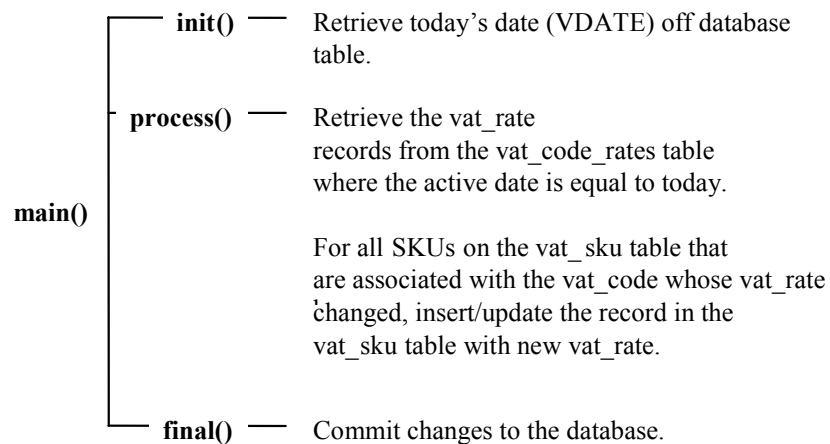
Threading Scheme: N/A



## Restart Recovery

The logical unit of work will be based on VAT code. Restart/recovery will be based on VAT code. It is recommended that records be committed after a set of 10,000 rows has been processed.

## Program Flow



## Shared Modules

N/A

## Function Level Description

**init():**

Retrieve tomorrow's date off the period table.

**process():**

This function will retrieve all records off of the VAT\_CODE\_RATES table with an active date of tomorrow. If any records exist on the VAT\_ITEM table with an active date on or after tomorrow for the VAT code fetched off of the VAT\_CODE\_RATES table, their VAT rate is updated with the VAT rate fetched off of the VAT\_CODE\_RATES table. Also, if an ITEM is currently assigned the VAT code with the changing VAT rate, a new record is inserted into VAT\_ITEM for the new VAT code/VAT rate combination with an active date of tomorrow.

**update\_vat\_item():**

This function updates any future VAT\_ITEM records with the new VAT rate fetched off the VAT\_CODE\_RATES table. It utilizes array processing to handle the update.

**insert\_vat\_item():**

Inserts a new record into the VAT\_ITEM table for any ITEMS that are currently associated with the VAT code fetched from the VAT\_CODE\_RATES table. A new record is inserted into VAT\_ITEM for the new VAT code/VAT rate combination with an active date of tomorrow.

**Technical Issues**

This program implements the use of array processing to handle the inserts/updates of the VAT\_ITEM table.

## Vendor replenished order build [vrplbld]

### Design Overview

| TABLE                 | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-----------------------|-------|--------|--------|--------|--------|
| EDI_ORD_TEMP          | No    | Yes    | No     | No     | No     |
| ORDHEAD               | Yes   | Yes    | Yes    | No     | No     |
| ORDSKU                | No    | No     | Yes    | Yes    | No     |
| ORDLOC                | No    | No     | Yes    | No     | No     |
| PERIOD                | Yes   | Yes    | No     | No     | No     |
| STORE                 | Yes   | Yes    | No     | No     | No     |
| WH                    | Yes   | Yes    | No     | No     | No     |
| UNIT_OPTIONS          | No    | Yes    | No     | No     | No     |
| ITEM_SUPP_COUNT<br>RY | Yes   | Yes    | No     | No     | No     |
| SUPP_IMPORT_ATT<br>R  | Yes   | Yes    | No     | No     | No     |
| SUPS                  | Yes   | Yes    | No     | No     | No     |
| ORDSKU HTS            | Yes   | Yes    | Yes    | No     | No     |
| SYSTEM_OPTIONS        | No    | Yes    | No     | No     | No     |
| DEAL_CALC_QUEUE       | No    | Yes    | Yes    | Yes    | No     |
| ORD_INV_MGMT          | No    | No     | Yes    | No     | No     |

Note: ORDHEAD (select via index) is referenced via the function  
ORDER\_NUMBER\_SQL.NEXT\_ORDER\_NUMBER.

**Indexes:** ORDHEAD(order\_no),  
ORDSKU(order\_no,item)

Function: To continue the process started by the program ediupack of building RETEK orders that reflect the vendor-generated orders as received through the EDI 855.

The program ediupack.pc processes the EDI 855's that have been received from vendors. Rows have been inserted onto the table EDI\_ORD\_TEMP for use by this program. The EDI\_ORD\_TEMP table contains all items which were included on the EDI 855, along with the vendor order number with which they were originally associated.

The items are then consolidated so that one RETEK order is created for each vendor-order-number/department/supplier or vendor-order-number/supplier or depending on UNIT\_OPTIONS.dept\_level\_orders (the latter occurs when UNIT\_OPTIONS.dept\_level\_orders = 'N').

Some details for each item are retrieved from the item/location tables and ITEM\_SUPPLIER\_COUNTRY and stored in ORDSKU and ORDLOC so that subsequent changes to the item do not affect the outstanding order.

A\_order\_amt is updated on the OTB table. If the supplier is foreign and OTB is calculated at cost, the added amount is total\_duty + total\_cost. If the supplier is not foreign and OTB is calculated at cost, the added amount is total\_cost. Finally, if OTB is calculated at retail, the added amount is total\_retail.

In records written to the ORDHEAD table:

- FREIGHT\_TERMS is given the value fetched from SUPS.FREIGHT\_TERMS
- ORIG\_APPROVAL\_DATE is assigned the value of the PERIOD.VDATE
- ORIG\_APPROVAL\_ID is assigned the value 'EDI855'

If an item is ordered for a location at which it does not have an item/location record, a record is built by calling the stored procedure NEW\_ITEM\_LOC.

The order numbers are allocated automatically on a 'next available' basis.

Note: The vendor minimum order quantity is not considered when writing to the order tables because the quantity is determined by vendor. Thus, this quantity is not stored in the edi\_ord\_temp table.

Re-run: If this program aborts and terminates normally or terminates abnormally restart after setting restart\_flag = 'Y' on restart\_program\_status table for current restart\_name, schema, & thread\_val.

### Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Vrplbld\_post in prepost.pc truncates edi\_ord\_temp table

Threading Scheme: SUPPLIER

### Restart Recovery

The logical unit of work (LUW) for vrplbld.pc will be a count of records pulled from the edi\_temp\_ord table. The number of records will be determined by the commit\_max\_ctr field on the restart\_control table. Further, one or more of the key values (vendor\_order\_no, supplier, dept) must change before a commit event can take place. This will ensure that only whole order will be saved, i.e. that a commit will never take place before all of the order detail records have been processed.

The commit\_max\_ctr field should be set to prevent excessive rollback space usage. Given the use of multiple threading, the recommended commit counter setting is 5000 records (subject to change based on experimentation).

Two cursors are defined, but only one cursor is opened for fetching (the criteria is based on whether or not departmental level ordering is allowed.)

```

EXEC SQL DECLARE c_items_d CURSOR FOR
 SELECT isc.origin_country_id,
 isc.supp_pack_size,
 NVL(isc.lead_time, 0),
 sia.agent,
 sia.lading_port,
 sia.discharge_port,
 s.currency_code,
 e.vendor_order_no,
 e.supplier,
 e.dept,
 e.wh_or_store_c,
 e.wh_or_store,
 e.item,
 e.item_desc,
 e.ref_item,
 e.unit_retail,
 DECODE(:edi_cost_override_ind, 'Y', iscl.unit_cost, e.unit_cost),
 e.qty_ordered,
 TO_CHAR(e.written_date, 'YYYYMMDD'),
 TO_CHAR(e.not_before_date, 'YYYYMMDD'),
 TO_CHAR(e.not_after_date, 'YYYYMMDD'),
 TO_CHAR(e.not_after_date, 'DD'),
 TO_CHAR(e.not_after_date, 'MM'),
 TO_CHAR(e.not_after_date, 'YYYY'),
 ';' || e.vendor_order_no ||
 ';' || TO_CHAR(e.dept) ||
 ';' || TO_CHAR(e.supplier)
 FROM item_supp_country isc,
 item_supp_country_loc iscl,
 sup_import_attr sia,
 sups s,
 edi_ord_temp e,
 v_restart_supplier rv
 WHERE isc.item = e.item
 AND isc.primary_country_ind = 'Y'
 AND isc.supplier = e.supplier
 AND sia.supplier(+) = e.supplier
 AND s.supplier = e.supplier
 AND rv.driver_value = e.supplier
 AND rv.driver_name = :os_restart_driver_name
 AND rv.num_threads = :oi_restart_num_threads
 AND rv.thread_val = :oi_restart_thread_val
 AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no, -999) OR
 (e.vendor_order_no = :os_restart_vndr_ord_no AND
 (e.dept > :os_restart_dept OR
 (e.dept = :os_restart_dept AND
 e.supplier > :os_restart_supplier)
)
)
)

```

```
)
ORDER BY e.vendor_order_no,
 e.dept,
 e.supplier,
 e.wh_or_store_c,
 e.wh_or_store,
 e.item;
```

```
EXEC SQL DECLARE c_items_s CURSOR FOR
SELECT isc.origin_country_id,
 isc.supp_pack_size,
 NVL(isc.lead_time, 0),
 sia.agent,
 sia.lading_port,
 sia.discharge_port,
 s.currency_code,
 e.vendor_order_no,
 e.supplier,
 e.dept,
 e.wh_or_store_c,
 e.wh_or_store,
 e.item,
 e.item_desc,
 e.ref_item,
 e.unit_retail,
 DECODE(:edi_cost_override_ind, 'Y', iscl.unit_cost, e.unit_cost),
 e.qty_ordered,
 TO_CHAR(e.written_date, 'YYYYMMDD'),
 TO_CHAR(e.not_before_date, 'YYYYMMDD'),
 TO_CHAR(e.not_after_date, 'YYYYMMDD'),
 TO_CHAR(e.not_after_date, 'DD'),
 TO_CHAR(e.not_after_date, 'MM'),
 TO_CHAR(e.not_after_date, 'YYYY'),
 ';' || e.vendor_order_no ||
 ';' || TO_CHAR(e.dept) ||
 ';' || TO_CHAR(e.supplier)
FROM item_supp_country isc,
 item_supp_country_loc iscl,
 sup_import_attr sia,
 sups s,
 edi_ord_temp e,
 v_restart_supplier rv
WHERE isc.item = e.item
AND isc.primary_country_ind = 'Y'
AND isc.supplier = e.supplier
AND sia.supplier (+) = e.supplier
AND s.supplier = e.supplier
AND rv.driver_value = e.supplier
AND rv.driver_name = :os_restart_driver_name
AND rv.num_threads = :oi_restart_num_threads
```

```

AND rv.thread_val = :oi_restart_thread_val
AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
 (e.vendor_order_no = :os_restart_vndr_ord_no AND
 e.supplier > :os_restart_supplier)
)
ORDER BY e.vendor_order_no,
 e.supplier,
 e.wh_or_store_c,
 e.wh_or_store,
 e.item;

```

### Program Flow

N/A

### Shared Modules

```

NEW_ITEM_LOC
ORDER_NUMBER_SQL.GET_NEXT_ORDER_NUMBER
CURRENCY_SQL.GET_RATE
CAL_TO_454_LDOW
ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS
ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS
ORDBA_CALC_SQL.ORDER_COST
ORDER_EXPENSE_SQL.INSERT_COST_COMP
ORDER HTS_SQL.DEFAULT_CALC HTS
OTB_SQL.ORD_APPROVE
ORDER_SETUP_SQL.DEFAULT_ORDER_INV_MGMT_INFO
ORDER_ATTRIB_SQL.MULTIPLE_LOCS_EXIST
SUP_INV_MGMT_SQL.GET_PURCHASE_PICKUP

```

### Function Level Description

#### init()

select department level orders flag from unit options  
select base country, latest ship days, FOB information, ELC indicator, EDI cost override  
indicator, calendar 454 indicator, bill-to location, and multi-channel indicator from system  
options  
select vdate from period

call restart initialization logic

#### process()

call open\_cursor to open appropriate cursor (department level or multi-  
department orders)

call fetch\_cursor for prime fetch of driving cursor

while

```

 if supplier, department or vendor order number changed then
 call create_header
 if item changed
 call create_item

```

```

 if multi-channel = 'N' or the location fetched is a store
 call add_loc
 If multi-channel is on and the location is a warehouse, then distribute:
 call dist_init
 call dist_set_distribution_rule
 for every replenishable warehouse
 call dist_expected_quantity
 call dist_distributed_quantity
 call add_loc
 call dist_final

 call add_loc

 call fetch_cursor

 if supplier, department or vendor order number changed then
 if the ELC indicator is Y, call add_cost_comp
 call item_defaults
 call apply_disc
 call header_defaults
 call update_header
 call update_order
 call check_vmi_order
 else if the item changed
 if the ELC indicator is Y, call add_cost_comp
 call item_defaults

 call restart commit logic
end while
 call item_defaults
 call apply_disc
 call header_defaults
 call update_header
 call update_order
 call check_vmi_order

```

**open\_cursor()**

open appropriate cursor depending on what type of orders are allowed.

---

**fetch\_cursor()**

fetch records from appropriate cursor

**get\_min\_max()**

finds the minimum and maximum lead times for the current order

**item\_active ()**

returns 1 if item is active (0 if inactive) on item\_loc

**create\_header()**

call function to get the next order number



(ORDER\_NUMBER\_SQL.NEXT\_ORDER\_NUMBER)  
 call CAL\_TO\_454\_LDOW procedure to convert order's not\_after\_date to the end-of-week date  
 to update otb\_eow\_date field  
 call CURRENCY\_SQL.GET\_RATE to get the exchange rate for the supplier's currency.  
 insert record into ordhead

#### **header\_defaults()**

Call ORDER\_ATTRIB\_SQL.MULTIPLE\_LOCS\_EXIST to retrieve the location if the order is a single location order. Call ORDER\_SETUP\_SQL.DEFAULT\_ORDHEAD\_DOCS to default the necessary documents for the order header. Also call ORDER\_SETUP\_SQL.DEFAULT\_ORDER\_INV\_MGMT to appropriate supplier inventory management information to the order with NULL inserted for the scaling constraints, since a vendor generated order is not eligible for scaling. If no values are found for the purchase type or pickup location, call SUP\_INV\_MGMT\_SQL.GET\_PURCHASE\_PICKUP to retrieve the default values. If no value is found for the pickup date, use the longest lead time to calculate the pickup date.

#### **update\_header()**

Check to see if any items on the order come from a country other than the ordhead import country. If so, update the import\_order\_ind to 'Y' on ordhead.

#### **update\_order()**

Update the Order Header table with the following values

- a. Purchase Type
- b. Pickup Location (only enter if the Purchase Type is 'FOB' or 'BACK')
- c. Pickup Date (only enter if the Purchase Type is 'FOB' or 'BACK') – **if value is later than the Not After Date, set equal to the Not After Date**
- d. Earliest Ship Date – minimum of Order/Item ESDs
- e. Latest Ship Date – maximum of Order/Item LSDs

Update Order Item table. Set the pickup location if the purchase type is 'FOB' or 'BACK'.

#### **item\_defaults()**

Call ORDER\_SETUP\_SQL.DEFAULT\_ORDSKU\_DOCS to default necessary documents for items in order.

#### **apply\_disc()**

if edi\_cost\_override\_ind = 'Y' then insert a record into the DEAL\_CALC\_QUEUE table if the current order number is not in the table, or update the order\_appr\_ind field to 'Y' if the order number is already in the table.

call update\_OTB

#### **add\_cost\_comp()**

Call ORDER\_EXPENSE\_SQL.INSERT\_COST\_COMP,  
 ORDER HTS\_SQL.DEFAULT\_CALC HTS and  
 ELC\_CALC\_SQL.CALC\_COMP.

#### **create\_item ()**

if edi\_cost\_override\_ind = 'Y' insert record into ordsku with the cost\_source = 'MANL'. If edi\_cost\_override\_ind != 'Y', insert a record into ordsku with the cost\_source = 'NORM';

#### **add\_loc ()**

insert into ordloc

**set\_worksheet\_status()**

For a given order number and reject code, update the status on the ordhead table to 'W'orksheet and update the reject code to whatever was passed in.

**order\_exists()**

For a given order number, check to see whether or not the order number is on the DEAL\_CALC\_QUEUE table.

**final()**

Call restart/recovery close logic.

**I/O Specification**

N/A

**Technical Issues**

N/A

## Wastage adjustment [wasteadj]

### Design Overview

This program will reduce inventory of spoilage type wastage items to account for natural wastage that occurs over the shelf life of the product. Only items with spoilage type wastage will be affected by this program. Sales type wastage will be accounted for at the time of sale.

Spoilage type wastage is due to the natural loss of a product over its shelf life. For example, as sausage sits on the shelf it loses a certain percentage of its weight each day due to evaporation of the water that is in the sausage. Therefore, a sausage that weighs one pound today may only weight .9 pounds tomorrow due to the loss of water from the sausage. This new batch program will reduce the physical on hand inventory for a store based on the daily wastage percentage which will be entered by the retailer at the item location level:

$$\text{New Stock on Hand} = \text{Actual Stock on Hand} * (1 - \text{Daily Wastage \%})$$

When stock is reduced, Merchandising will process these records just as any other stock adjustment.

By automatically reducing stock on hand based on the wastage percentage and the shelf life of an item, the on hand inventory in Merchandising will be more accurate than if these adjustments were not made.

| TABLE          | INDEX | SELECT | INSERT | UPDATE | DELETE |
|----------------|-------|--------|--------|--------|--------|
| ITEM_MASTER    | No    | Yes    | No     | No     | No     |
| ITEM_LOC_SOH   | No    | Yes    | No     | Yes    | No     |
| ITEM_LOC       | No    | Yes    | No     | No     | No     |
| INV_ADJ        | No    | No     | Yes    | No     | No     |
| TRAN_DATA      | No    | No     | Yes    | No     | No     |
| PERIOD         | No    | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS | No    | Yes    | No     | Yes    | No     |

### Scheduling Constraints

Processing Cycle: Phase III (Before stock ledger processing)

Scheduling Diagram:

Pre-Processing: Run this program before the stock ledger roll up programs to make sure that the stock adjustments taken during the current day are credited to the appropriate day.

Post-Processing: N/A

Threading Scheme: N/A

**Restart Recovery**

Restart recovery is based on last item processed. The program will commit only when the commit max counter is reached.

**Program Flow**

N/A

**Shared Modules**

N/A

**Function Level Description**

init

- Initialize restart variables.
- Allocate memory to arrays for array processing
- Collect system information

process

- Fetch ITEMS with wastage type spoilage ('SP')
- Retrieve single ITEM from fetch array for processing
- If a the ITEM is a transaction level item:
  - retrieve appropriate information from ITEM\_LOC and ITEM\_LOC\_SOH
  - add information to insert/update arrays

create\_inv\_adj

- Adds a record to the inv\_adj insert array

create\_tran\_data

- Adds a record to the tran\_data insert array

update\_stock

- Performs array updates of ITEM\_LOC\_SOH records

write\_arrays

- Performs array inserts to inv\_adj and tran\_data

final

- Closes restart recovery

**I/O Specification**

N/A

**Technical Issues**

N/A

**Customization Issues**

N/A

## Warehouse store assignment [whstrasg]

### Design Overview

The whstrasg program is used to update the default warehouse attribute on the Store table and the sourcing warehousing replenishment attribute on the Replenishment Item Location table based on the warehouse-store assignments on the Warehouse Store Assignment table. The program will run nightly and on the night before the assignment date for a warehouse-store assignment, it will update the relevant tables with the new warehouse-store relationship. If the warehouse is to be assigned to an item on the Replenishment Item Location and the item is not associated to the warehouse, a non-fatal error message should be output indicating that the item is not associated to the warehouse. When updating the sourcing warehouse on the Replenishment Item Location table, a system indicator will determine whether the sourcing warehouse will be updated for warehouse-store replenishment records, cross-docked replenishment records, WH/Cross Link records, or all .

### Affected Tables:

| TABLE           | INDEX | SELECT | INSERT | UPDATE | DELETE |
|-----------------|-------|--------|--------|--------|--------|
| WH_STORE_ASSIGN | Yes   | Yes    | No     | Yes    | No     |
| STORE           | Yes   | No     | No     | Yes    | No     |
| REPL_ITEM_LOC   | Yes   | Yes    | No     | Yes    | No     |
| ITEM_LOC        | Yes   | Yes    | No     | No     | No     |
| SYSTEM_OPTIONS  | No    | Yes    | No     | No     | No     |
| PERIOD          | No    | Yes    | No     | No     | No     |

### Scheduling Constraints

Processing Cycle: PHASE 3 (daily)

Scheduling Diagram: After all replenishment batch programs since the assignments are meant to be effective the next day, not the current day.

Pre-Processing: N/A

Post-Processing: Clear off the WH\_STORE\_ASSIGN table (whstrasg\_post)

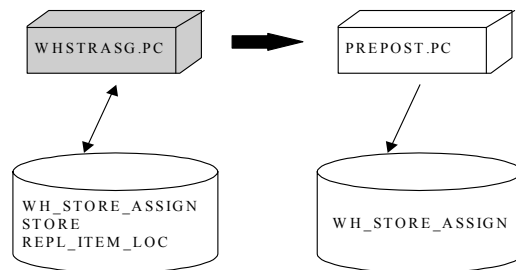
Threading Scheme: Store.

### Restart Recovery

Restart/recovery for this program is set up at the store level.

The recommended commit counter setting is 2000 records (subject to change based on experimentation).

## Program Flow



## Shared Modules

N/A

## Function Level Description

### Init():

This function will call `retex_init()` to start up restart recovery and bring back the bookmark string in case of a restart. Fetches `vdate` from the `PERIOD` table; fetches `wh_store_assign_type` from the `SYSTEM_OPTIONS` table.

### Process():

This function is the controller for most of the processing in the program. It opens and fetches from the driving cursor according to Retek standards, using array processing and handling restart/recovery

Once the records are fetched, call `update_repl_item_loc()` to fill the update array of the `REPL_ITEM_LOC` table then call `update_repl()` to update the `source_wh` field of the `REPL_ITEM_LOC` table.

For every for fetch loop, call `update_store()` to update the `default_wh` field of the `STORE` table and `processed_ind` of the `WH_STORE_ASSIGN` table.

### Final():

This function will call `retex_close()` to wrap up restart-recovery and populate the final error message, if necessary.

### Update\_repl\_item\_loc():

This function will check if ITEM\_LOC exists for the item/warehouse. If not, return a non-fatal error and write an error message to the error log. If there is no non-fatal error, fill the update array of the REPL\_ITEM\_LOC table.

#### Init\_arrays():

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart\_control.

#### Update\_repl():

This function will update the source\_wh field of the REPL\_ITEM\_LOC table to the warehouse of the driving cursor using array update. This is by rowid.

#### Update\_store():

This function will update the default\_wh field of the STORE table using array update, this is not by rowid. And update the process\_ind field of the WH\_STORE\_ASSIGN table to 'Y' for all records that have been successfully processed, this is by rowid.

### **I/O Specification**

N/A

### **Technical Issues**

N/A