Retek® Merchandising System 9.0.9

Addendum to Operations Guide

Retek

WWW.RETEK.COM

HELPING THE RETAIL INDUSTRY CREATE, MANAGE AND FULFILL CONSUMER DEMAND™

Corporate Headquarters:

Retek Inc.

Retek on the Mall

950 Nicollet Mall

Minneapolis, MN 55403

888.61.RETEK (toll free US)

+1 612 587 5000

European Headquarters:

Retek

110 Wigmore Street

London

W1U 3RW

United Kingdom

Switchboard:

+44 (0)20 7563 4600

Sales Enquiries:

+44 (0)20 7563 46 46

Fax: +44 (0)20 7563 46 10

Customer Support

Customer Support hours:

Customer Support is available 7x24x365 via e-mail, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance.

Contact Method	**Contact Information**
Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
E-mail	support@retek.com
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

Chapter 1 – Introduction 1

Chapter 2 – Customs entry download (cednld.doc) 3

Modification 3

Design overview 3

Scheduling constraints 3

Restart recovery 3

Program flow 4

Shared modules 4

Function level description 4

init 4
process 5
Process_shipments 5
Process_order_items 5
Process_bl_awb_id 5
Process_container 6
Process_license_visa 6
Process_charges 6
Process_missing_docs 6
Update_ce_head 6
Size_arrays 6
Init_buffers 6
Write_line 6
Final 6

I/O specification 7

Output file 7

Technical issues 12

Chapter 3 – Transportation upload (tranupld.doc) 13

Modification 13

Design overview 13

Upload files 13

Scheduling constraints 14

Pre/post logic description 14

Restart recovery 14

Program flow 14

Tables used 14

Shared modules 14

Function level description 15

init(): 15
file_process(): 15
format_record(): 15
format_dtran_record(): 15
format_dpoit_record(): 15
format_ftail_record(): 15
validate_record(): 15
validate_tran_record(): 16
validate_poit_record(): 16
process_record(): 17

I/O specification 21

Technical issues 26

Chapter 4 – P.O. receipt transactions upload (rcvupld.doc) 27

Modification 27

Design overview 27

Scheduling constraints 29

Restart recovery 29

Program flow 30

Shared modules 31

Function level description 32
init() 32
process() 32
validate_THEAD() 33
validate_TDETL() 34
process_TDETL() 34
create_shipment() 34
ON Fatal Error 34
ON Non-Fatal Error 34
I/O specification 35
Input file 35
Reject file 37
Error file 37
Technical issues 37
Chapter 5 – ReSA 9.0 RTLOG Layout (SA RTLOG.doc) 39
Modification 39
Chapter 6 – Fashion merchandise hierarchy download (fmednldf.doc) 71
Modification 71
Design overview 71
Scheduling constraints 71
Restart recovery 71
Department 72
Class 74
Subclass 76
Program flow 78
Shared modules 80
Function level description 81
I/O specification 81
Technical issues 81

Chapter 7 – Staple merchandise hierarchy download (fmednlds.doc) 83

Modification 83

Design overview 83

Scheduling constraints 83

Restart recovery 83

Department 84
Class 85
Subclass 87

Program flow 88

Shared modules 91

Function level description 91

I/O specification 92

Technical issues 92

Chapter 8 – Product security rebuild (sprdrbld.doc) 93

Modification 93

Design overview 93

Scheduling constraints 94

Restart recovery 94

Program flow 95

Shared modules 96

Driving Cursor for SKU 96
Driving Cursor for Dept 97

Function level description 97

Main() 97
Init() 97
Process() 98
Size_rule_array() 98
Set_null_to_field() 98
Process_record() 99
Logical_or_indicators() 99
Update_array() 100
Write_array() 100
final() 100

I/O specification 101

Technical issues 101

Chapter 9 – Transfer shipments upload (tsfoupld.doc)... 103

Modification 103

Design overview 103

Scheduling constraints 104

Restart recovery 105

Program flow 106

Shared modules 107

Function level description 108

init() 108
process() 108
validate_THEAD() 110
process_THEAD() 111
validate_TDETL() 112
process_TDETL() 113
post_transfers() 114
format_header_fields() 115
format_detail_fields() 115
process_carton() 115

I/O specification 116

Input file 116
Output file 120
Reject file 122
Error file 122

Technical issues 122

Chapter 1 – Introduction

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

- Customs entry download (cednld.doc)
- Transportation upload (tranupld.doc)
- P.O. receipt transactions upload (rcvupld.doc)
- Sales Audit Interface file (SA RTLOG.DOC) ?
- Items (fmednldf.doc)
- Items (fmednlds.doc)
- Security (sprdrbld.doc)
- Transfers (tsfoupld.doc)

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

Chapter 2 – Customs entry download (cednld.doc)

Modification

Changed the shipment number to char(20) and bl_awb_id to char(30).

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

Shared modules

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

Initally 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.

RecordName	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	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	YYYYMMDDHH24 MISS 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	YYYYMMDDHH24 MISS 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	

RecordName	Field Name	Field Type	Default Value	Description
	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	YYYYMMDDHH24 MISS format
	Reliquidation Amount	Number(20,4)	ce_head.reliquidation_amt	
	Reliquidation Date	Char(14)	ce_head.reliquidation_date	YYYYMMDDHH24 MISS 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	YYYYMMDDHH24 MISS 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	

RecordName	Field Name	Field Type	Default Value	Description
	Export Date	Char(14)	ce_shipment.export_date	YYYYMMDDHH24 MISS
	Import Date	Char(14)	ce_shipment.import_date	YYYYMMDDHH24 MISS
	Arrival Date	Char(14)	ce_shipment.arrival_date	YYYYMMDDHH24 MISS
	Export Country	Char(3)	ce_shipment.export_country _id	
	Shipment Number	Char(20)	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	Number(8)	ce_ord_item.item	
	BL AWB ID	Char(30)	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	YYYYMMDDHH24 MISS 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	

RecordName	Field Name	Field Type	Default Value	Description
	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	YYYYMMDDHH24 MISS 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	YYYYMMDDHH24 MISS 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.

RecordName	Field Name	Field Type	Default Value	Description
	BL AWB ID	Char(30)	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_sc ac_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_typ e	
	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	Number(3)	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_uo m	
	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(8)	ce_charges.pack_item	
	HTS	Char(10)	ce_charges.hts	

RecordName	Field Name	Field Type	Default Value	Description
	Effect From Date	Char(14)	ce_charges.effect_from	YYYYMMDDHH24 MISS format
	Effect To Date	Char(14)	ce_charges.effect_to	YYYYMMDDHH24 MISS 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)	DeterminedInternally	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

Chapter 3 – Transportation upload (tranupld.doc)

Modification

Changed the shipment number to char(20) and bl_awb_id to char(30).

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 diverge from the current RMS 8.0 upload programs in that it 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 (trnupld.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 singe 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.

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.

Program flow

Read current record into appropriate data structure

- This format process should include checks that 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

Shared modules

TRANSPORTATION_SQL.GET_NEXT_ID

This function will create a new Transportation ID for the TRANSPORTATION table.

Function level description

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 desc_look 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 style 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_detail table with code_desc = ‘TRCO’.

- validate the packing type by calling a function which will return true when the given packing type = code_type on the code_detail table with code_desc = ‘PKMT’.
- validate the item quantity by calling a function which will return true when the given item quantity = uom on the uom_class table with uom_class = QTY’.
- validate the carton quantity by calling a function which will return true when the given carton quantity = uom on the uom_class table with uom_class = ‘PACK’.
- validate the gross weight by calling a function which will return true when the given gross weight = uom on the uom_class table with uom_class = ‘MASS’.
- validate the net weight by calling a function which will return true when the given net weight = 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 he 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’hange 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’elete 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(8)	TRANUPLD	Identifies program to use
	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

Record Name	Field Name	Field Type	Default Value	Description
	Consolidator	*Char(10)*		*Identifies the Consolidator. Validated against PARTNER table with type = 'CO'*
	Vessel ID	**Char(20)**		**Identifies the Vessel**
	Voyage ID	**Char(10)**		**Identifies the Voyage or Flight ID**
	Estimated Depart Date	**Char(8)**		**YYYYMMDD format**
	Shipment Number	Char(20)		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'

Record Name	Field Name	Field Type	Default Value	Description
	Service Contract Number	Char(15)		Identifies the outside Service Contract Number
	Container id	**Char(20)**		**Identifies the Container**
	Container SCAC code	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(30)		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'

Record Name	Field Name	Field Type	Default Value	Description
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.
	Rush Ind	*Char(1)*	*Defaulted to ‘N’*	*Identifies whether or not the item should be on a ‘Rush’ delivery. Valid values are ‘Y’ and ‘N’*
	Order number	***Number(8)***		***Internal Retek order no***
	Item	***Number(8)***		***Internal Retek Item (style, SKU or Pack)***
	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

Record Name	Field Name	Field Type	Default Value	Description
	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
	Item Qty	Number(12)		Qty of Items (4 implied decimals)
	Item QTY UOM	Char(4)		Identifies the UOM associated with the item quantity
	Carton QTY	Number(12)		Qty of Cartons (4 implied decimals)
	Carton QTY UOM	Char(4)		Identifies the UOM associated with the carton quantity

Record Name	Field Name	Field Type	Default Value	Description
	Gross WT	Number(12)		Gross weight (4 implied decimals)
	Gross WT UOM	Char(4)		Identifies the UOM associated with the gross weight
	Net WT	Number(12)		Net Weight (4 implied decimals)
	Net WT UOM	Char(4)		Identifies the UOM associated with the net weight
	Cubic	Number(12)		Cubic size (4 implied decimals)
	Cubic UOM	Char(4)		Identifies the UOM associated 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 lines in file (not including 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’elete.

Chapter 4 – P.O. receipt transactions upload (rcvupld.doc)

Modification

Modified the description of Inventory status column.

Design overview

The purpose of this batch module is to accept receipt details from an external system. The receipt transactions will provide feedback on orders existing in the Retek system, and will cross-reference shipments if they exist. The receipt detail information can be processed using any of the supported item types (i.e. SKU, UPC, or VPN) within the RMS system. It also processes receiver unit adjustment, as well as accepts item that does not exist on the purchase order.

The following functions will be performed for each item received:

- Create/update shipment and shipment/SKU records
- Create/update order/SKU and order/SKU/location records
- Update order header status to complete if all units received against the order
- Create item/location relation for receiving location (if it doesn’t exist)
- Update perpetual inventory
- Update average cost of item at receiving location (if stock on hand not = 0 or negative)
- Update unit cost through package call to RECEIVE_SQL.ITEM if the cost change is a base cost change (supplier is the primary supplier for the item)
- Write stock ledger financial transactions
- Update the snapshot stock on hand quantity for received item/location if a stock count is in progress
- Create transfers based on allocations tied to the order
- Update OTB table’s receipt amount
- Write supplier data record for vendor analytics
- Update contract_detail and contract_header if PO is for a contract
- Write ticket_request if item is received or allocated into store and if the ticket is generated at PO receipt time
- Write to variable asn_exists if the supplier is using “Evaluated receipt settlement” as payment process method and a matched shipment is found for the receipt

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	Yes	No	Yes	No
ALLOC_DETAIL	Yes	Yes	No	Yes	No
PACKWH	Yes	No	Yes	Yes	No
ORDLOC	No	Yes	Yes	Yes	No
ORDSKU	No	No	Yes	Yes	No
ORDHEAD	No	No	No	Yes	No
OTB	No	No	No	Yes	No
CONTRACT_HEADER	No	Yes	No	Yes	No
CONTRACT_DETAIL	No	Yes	No	Yes	No
RAG_SKUS_ST	Yes	Yes	Yes	Yes	No
RAG_SKUS_WH	Yes	Yes	Yes	Yes	No
WIN_STORE	Yes	Yes	Yes	Yes	No
WIN_WH	Yes	Yes	Yes	Yes	No
SHIPMENT	No	Yes	Yes	Yes	No
SHIPSKU	No	Yes	Yes	Yes	No
STAKE_SKU_LOC	No	No	No	Yes	No
SUP_DATA	No	No	Yes	Yes	No
TRAN_DATA	No	No	Yes	No	No
TSFHEAD	Yes	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	Yes	Yes	No
V_SKU_INFO	Yes	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	Yes	No
RAG_SKUS	Yes	Yes	No	Yes	No
PACKHEAD	Yes	Yes	No	Yes	No

Scheduling constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program must run after tsfparse.pc. This program will likely be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of he batch run if pending warehouse transactions. 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: STORE and WH – Driven by distinct files by location, or consolidated receipt files.

Restart recovery

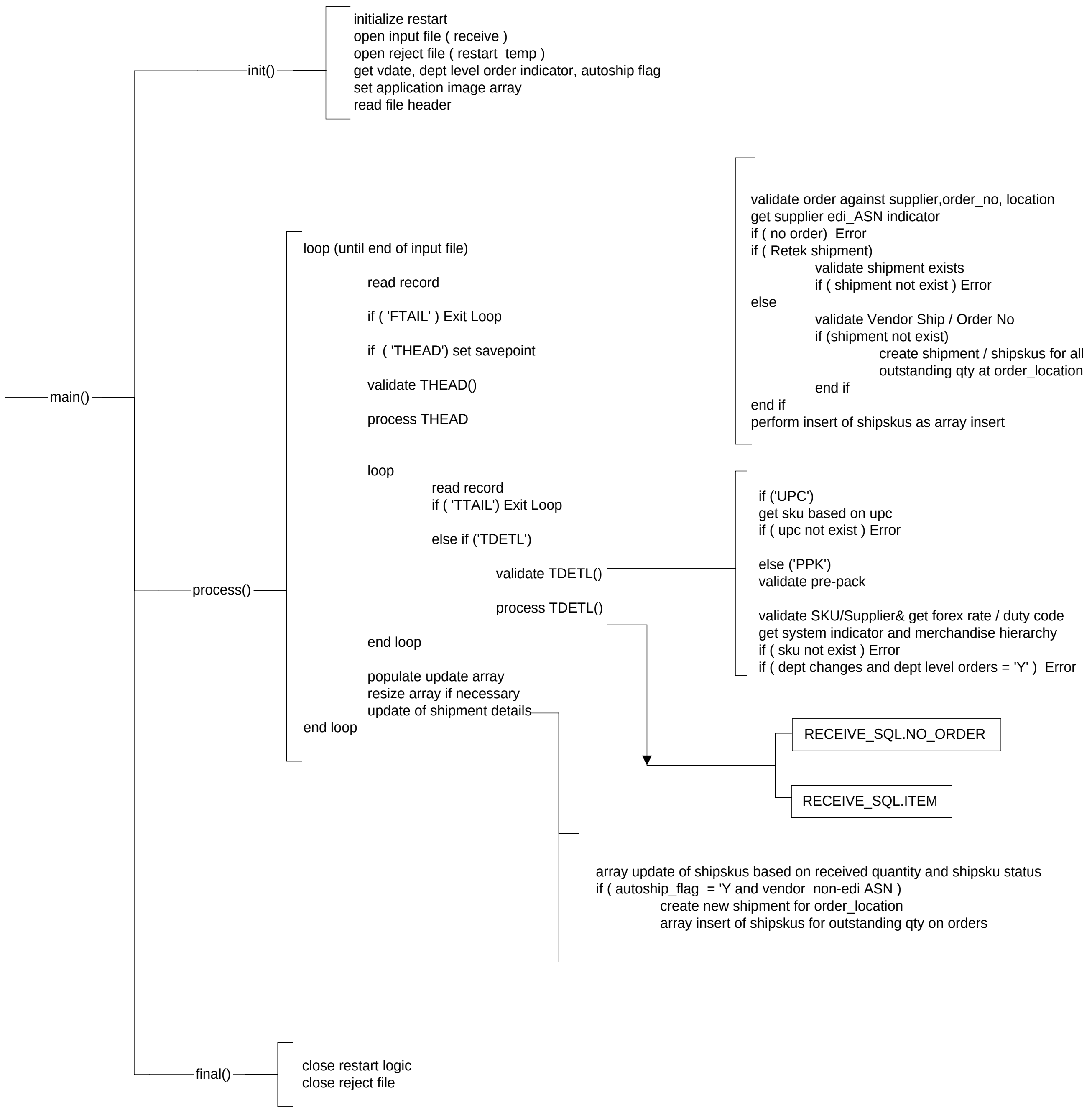
The logical unit of work for the receiving module is the discrete receipt transaction. Each receipt will be identified by the vendor ASN number (or shipment number if non-EDI), the Retek order number and a unique transaction set number generated by the external system. This receipt transaction will be defined as the logical unit of work. If any portion of the processing for the complete receipt transaction fails, the entire receipt must be re-processed.

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

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

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

Program flow



```
main()
    init()
        initialize restart
        open input file ( receive )
        open reject file ( restart  temp )
        get vdate, dept level order indicator, autoship flag
        set application image array
        read file header

    process()
        loop (until end of input file)

                read record

                if ( 'FTAIL' ) Exit Loop

                if  ( 'THEAD') set savepoint

                validate THEAD()
                        validate order against supplier,order_no, location
                        get supplier edi_ASN indicator
                        if ( no order)  Error
                        if ( Retek shipment)
                                validate shipment exists
                                if ( shipment not exist ) Error
                        else
                                validate Vendor Ship / Order No
                                if (shipment not exist)
                                        create shipment / shipskus for all
                                        outstanding qty at order_location
                                end if
                        end if
                        perform insert of shipskus as array insert

                process THEAD

                loop
                        read record
                        if ( 'TTAIL') Exit Loop

                        else if ('TDETL')

                                validate TDETL()
                                        if ('UPC')
                                        get sku based on upc
                                        if ( upc not exist ) Error

                                        else ('PPK')
                                        validate pre-pack

                                        validate SKU/Supplier& get forex rate / duty code
                                        get system indicator and merchandise hierarchy
                                        if ( sku not exist ) Error
                                        if ( dept changes and dept level orders = 'Y' )  Error

                                process TDETL()
                                        RECEIVE_SQL.NO_ORDER
                                        RECEIVE_SQL.ITEM

                end loop

                populate update array
                resize array if necessary
                update of shipment details
                        array update of shipskus based on received quantity and shipsku status
                        if ( autoship_flag  = 'Y and vendor  non-edi ASN )
                                create new shipment for order_location
                                array insert of shipskus for outstanding qty on orders
        end loop

    final()
        close restart logic
        close reject file
```

Shared modules

RECEIVE_SQL.ITEM: Package referenced to perform all receipt logic, including

- Create/update shipment and shipment/SKU records
- Create/update order/SKU and order/SKU/location records
- Update order header status to complete if all units received against the order
- Create item/location relation for receiving location (if it doesn’t exist)
- Update perpetual inventory
- Update average cost of item at receiving location (if stock on hand not = 0 or negative)
- Update unit cost through package call to RECEIVE_SQL.ITEM if the cost change is a base cost change (supplier is the primary supplier for the item)
- Write stock ledger financial transactions
- Update the snapshot stock on hand quantity for received item/location if a stock count is in progress
- Create transfers based on allocations tied to the order
- Update OTB table’s receipt amount
- Write supplier data record for vendor analytics
- Update contract_detail and contract_header if PO is for a contract
- Write ticket_request if item is received or allocated into store and if the ticket is generated at PO receipt time
- Write AP_tran if the supplier is using “Evaluated receipt settlement” as payment process method and a matched shipment is found for the receipt

The following are called from the RECEIVE_SQL package and are thus indirect calls.

STOCK_LEDGER_SQL.TRAN_DATA_INSERTS: Package referenced by RECEIVE_SQL package to perform the stock ledger transaction inserts for receipt of goods.

NEW_STAPLE_LOC, NEW_FASHION_LOC, NEW_PACK_LOC: These stored procedures are used to create item/location relationships for locations that are to receive goods.

AUTO_TRANSFER_WH: Function in the RECEIVE_SQL package, called when performing a warehouse receipt to create transfers automatically based on allocations tied to the order.

Function level description

init()

```
declare structure array for shipsku
initialize restart recovery
open input file ( receipts )
    - file should be specified as input parameter to program
open reject file ( as a temporary file for restart )
    - file should be specified as input parameter to program
get vdate from period table
set application image array - save the line counter
```

process()

```
loop
    read record from input file
    if ( 'FTAIL' )  Exit Loop
    if  ( 'THEAD')
        set savepoint and save current file pointer position
        get transaction header record details
        validate_THEAD()
        reset detail count
        process_THEAD()
    end if
    loop
        read record from input file
        if ( 'TTAIL')  Exit Loop
        if ('TDETL')
            validate_TDETL()
            process_TDETL()
            write detail transaction to shipsku array
        end if
        if ( detail count = max array count )
            resize array structures for shipsku array
            increase max array count
        end if
```

```
                increment detail count
        end loop
        if ( no errors )
                post_receipts()
        if ( non-Fatal Error encountered )
                reject_record  - call write error and pass file pointer as of last
                savepoint and current file pointer
                rollback transaction
        end if
        if ( transaction count > max commit count )
                restart file commit
                        - save the current input file pointer position
                        - save the line counter in restart image
        end if
end loop
restart commit final
```

validate_THEAD()

```
validate supplier number and get supplier edi_ASN indicator
validate order against supplier, order number, location
if ( order number does not exist )  Error
if ( Retek shipment)
        validate shipment exists
        if ( shipment does not exist ) Error
else
        validate Vendor Ship / Order No
        if (shipment does not exist)
                create shipment / shipskus for all outstanding qty at
                order_location
        end if
end if
perform insert of shipskus as array insert
```

validate_TDETL()

```
if ( Item Type = 'UPC' )
    select sku from upc_ean based on the upc and supplement
    if ( upc does not exist )
        write non-Fatal Error ( upc not found )
    end if
end if
```

process_TDETL()

```
- get sku system indicator and merchandise hierarchy
if ( system indicator does not exist )
    write non-Fatal Error ( sku not found )
end if
call RECEIVE_SQL.NO_ORDER package function
call RECEIVE_SQL.ITEM package function
( see design specification for RECEIVE_SQL )
update_shipsku()
updates the shipsku table with the quantity received from the input file; insert a
record if none exists
```

create_shipment()

get next shipment# and insert into shipment and shipsku tables

ON Fatal Error

rollback to last physical commit point

Exit Program

ON Non-Fatal Error

rollback to last savepoint

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

I/O specification

Input file

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

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)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	RCPT	Identifies file as ‘Receipt’
	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 Identifier	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	specified by external system	used to force unique transaction check
	Transaction Date	Date	specified by external system	date the transfer was created in external system
	Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of receiving location
	Location Value	Char(4)	location identifier	specifies the receiving location id number
	Adjustment Flag	Char(1)	‘Y’ – Adjustment‘N’ – Not Adjust.	Not used for Transfers – will always be ‘N’ (Used for receiver error adjustment for common receiving layout).
	Order Number	Char(8)	Retek order number	specifies the Retek order cross reference

File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	Shipment Number	Char(10)	Retek shipment number	specifies the Retek shipment cross reference
	Supplier Identifier	Char(10)	Retek supplier number	specifies the Retek shipment cross reference
	Vendor ASN	Char(15)	vendor shipment number reference	reference to vendor shipping document
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPCSKUVPN	item type will be represented as a UPC or SKU or VPN
	Item Value	Char(30)	item identifier	the id number of a SKU or UPC(or VPN)
	Supplement	Char(5)	supplemental identifier	used to further specify the id of an UPC item
	Carton	Char(20)	carton number	unique identifier of each carton associated with a shipment.
	Inventory Status	Char(2)	inventory status	The value of this field is not used by rcvupld.pc.

File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	Receipt Quantity Sign Flag	Char(1)	‘P’ – positive‘N’ – Negative	Always positive for transfer (file layout common to receiving which can have pos. or neg. adjustments)
	Receipt Quantity	Char(12)		number of units received of the given item
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	Line number of the current file
	Transaction Detail Line Count	Number(6)	sum of detail lines	sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	Line number of the current file
	Total Transaction Line Count	Number(10)	sum of all transaction lines	all lines in file less the file header and trailer records

Reject file

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the receiving module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error file

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

Technical issues

N/A

Chapter 5 – ReSA 9.0 RTLOG Layout (SA RTLOG.doc)

Modification

Modified the description of Reference Number 1.

The following illustrates the file layout format of the Retek TLOG. The content of each Retek TLOG file is per store per day. The filename convention will be RTLOG_STORE_DATETIME.DAT (e.g. RTLOG_1234_01221989010000.DAT)

```
------------------------------------
FHEAD                   (Only 1 per file, required)
      THEAD             (Multiple expected, one per transaction, required for each transaction)
      TCUST       (Only 1 per THEAD record allowed, optional for some transaction types, see
table below)
            CATT  (Attribute record specific to the TCUST record – Multiple allowed, only valid
if TCUST exists)
            TITEM       (Multiple allowed per transaction, optional for some transaction types,
see table below)
                  IDISC (Discount record specific to the TITEM record – Multiple allowed per
item, optional see table below)
      TTAX        (Multiple allowed per transaction, optional see table below)
            TTEND       (Multiple allowed per transaction, optional for some transaction types,
see table below)
TTAIL             (1 per THEAD, required)
FTAIL                         (1 per file, required)
------------------------------------
```

The order of the records within the transaction layout above is important. It aids processing by ensuring that information is present when it is needed.

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	File Type Definition	Char(4)	RTLG	Identifies file as ‘Retek TLOG’.	Y	Left/Blank
	File Create Date	Char(14)	Create date	Date and time file was written by external system (YYYYMMDDHHMMSS).	Y	Left/None
	Business Date	Char(8)	Business Date to process	Business date of transactions. (YYYYMMDD).	Y	Left/None
	Location Number	Char(4)	Specified by external system	Store or warehouse identifier.	Y	Left/None
	Reference Number	Char(30)	Specified by external system	This may contain the Polling ID associated with the consolidated TLOG file or used for other purpose.	N	Left/Blank
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type.	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Register	Char(5)		Till used at store.	Y	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Transaction Date	Char(14)	Transaction date	Date transactions were processed at the POS (YYYYMMDDHHMMSS).	Y	Left/None
	Transaction Number	Number(10)		Transaction identifier.	Y	Right/0
	Cashier	Char(10)		Cashier identifier.	N	Left/Blank
	Salesperson	Char(10)		Salesperson identifier.	N	Left/Blank
	Transaction Type	Char(6)	Refer to 'TRAT' code_type for a list of valid types.	Transaction type.	Y	Left/Blank
	Sub-transaction type	Char(6)	Refer to 'TRAS' code_type for a list of valid types.	Sub-transaction type. For sale, it can be employee, drive-off etc.	N	Left/Blank
	Orig_tran_no	Number(10)		Populated only for post-void transactions. Transaction number for the original tran that will be cancelled.	N	Right/0
	Orig_reg_no	Char(5)		Populated only for post-void transactions. Register number from the original tran.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Reason Code	Char(6)	Refer to 'REAC' code_type for a list of valid codes. If the transaction type is ‘PAIDOU’ and the sub transaction type is ‘MV’ or ‘EV’ than the valid codes come from the non_merch_code_head table.	Reason entered by cashier for some transaction types. Required for Paid In and Paid out transaction types, but can also be used for voids, returns, etc.	N	Left/Blank
	Vendor Number	Char(10)		Supplier id for a merchandise vendor paid out transaction, partner id for an expense vendor paid out transaction.	N	Left/Blank
	Vendor Invoice Number	Char(30)		Invoice number for a vendor paid out transaction.	N	Left/Blank
	Payment Reference Number	Char(16)		The reference number of the tender used for a vendor payout. This could be the money order number, check number, etc.	N	Left/Blank
	Proof of Delivery Number	Char(30)		Proof of receipt number given by the vendor at the time of delivery. This field is populated for a vendor paid out transaction.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Reference Number 1	Char(30)		Number associated with a particular transaction, for example weather for a Store Conditions transaction. The sa_reference table defines what this field can contain for each transaction type. In case of a DCLOSE transaction, this value will be used as the number of files expected in saimptlogfin.pc. Hence, this field becomes mandatory when the transaction type is DCLOSE. If this field is left NULL, the system will set the store status to Partially Loaded, even though there is a DCLOSE transaction.	N	Left/Blank
	Reference Number 2	Char(30)		Second generic reference number.	N	Left/Blank
	Reference Number 3	Char(30)		Third generic reference number.	N	Left/Blank
	Reference Number 4	Char(30)		Fourth generic reference number.	N	Left/Blank
	Value Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the value.	Y if Value is present	Left/None

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Value	Number(20)		Value with 4 implied decimal places. Populated by the retailer for TOTAL trans, populated by Retek sales audit for SALE, RETURN trans.	Y if tran is a TOTAL.	Right/0
Transaction Customer	File Type Record Descriptor	Char(5)	TCUST	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Customer ID	Char(16)	Customer identifier	The ID number of a customer.	Y	Left/Blank
	Customer ID type	Char(6)	Refer to 'CIDT' code_type for a list of valid types	Customer ID type.	Y	Left/Blank
	Customer Name	Char(40)		Customer name.	N	Left/Blank
	Address 1	Char(40)		Customer address.	N	Left/Blank
	Address 2	Char(40)		Additional field for customer address.	N	Left/Blank
	City	Char(30)		City.	N	Left/Blank
	State	Char(3)	State identifier	State.	N	Left/Blank
	Zip Code	Char(10)	Zip identifier	Zip code.	N	Left/Blank
	Country	Char(3)		Country.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Home Phone	Char(20)		Telephone number at home.	N	Left/Blank
	Work Phone	Char(20)		Telephone number at work.	N	Left/Blank
	E-mail	Char(100)		E-mail address.	N	Left/Blank
	Birthdate	Char(8)		Date of birth. (YYYYMMDD)	N	Left/Blank
Customer Attribute	File Type Record Descriptor	Char(5)	CATT	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Attribute type	Char(6)	Refer to 'SACA' code_type for a list of valid types	Type of customer attribute	Y	Left/Blank
	Attribute value	Char(6)	Refer to members of 'SACA' code_type for a list of valid values	Value of customer attribute.	Y	Left/Blank
Transaction Item	File Type Record Descriptor	Char(5)	TITEM	Identifies file record type.	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Item Status	Char(6)	Refer to 'SASI' code_type for a list of valid codes.	Status of the item within the transaction, V for item void, S for sold item, R for returned item.	Y	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Item Type	Char(6)	Refer to 'SAIT' code_type for a list of valid codes.	Identifies what type of item is transmitted.	Y	Left/Blank
	SKU	Number(8)	Item identifier	ID number	Either SKU	Left/Blank
	UPC	Char(13)	Item identifier	ID number	OrUPC	Left/Blank
	Supplement	Number(5)	Supplemental identifier	Used to further specify the ID of a UPC.	N	Left/Blank
	Voucher	Char(16)		Gift certificate number	N	Right/0
	Item Number	Char(16)	Item identifier	Populated by retailer for Item types other than SKU, UPC or GCN. Allows retailers more flexibility to store additional item types within ReSA.	N	Left/Blank
	Department	Number(4)		Identifies the department this item belongs to.This is filled in by saimptlog.	N	Right/Blank
	Class	Number(4)	Item's class	Class of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Right/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Subclass	Number(4)	Item’s subclass	Subclass of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Right/Blank
	System Indicator	Char(1)	Refer to 'IMTP’ code_type for a list of valid codes.	The type of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Left/None
	Quantity Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the quantity	Y	Left/None
	Quantity	Number(12)		Number of items purchased with 4 decimal places.	Y	Right/0
	Unit Retail	Number(20)		Unit retail with 4 implied decimal places.	Y	Right/0
	Override Reason	Char(6)	Refer to 'ORRC' code_type for a list of valid codes.	This column will be populated when an item's price has been overridden at the POS to define why it was overridden.	Y if unit retail was manually entered	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Original Unit Retail	Number(20)		Value with 4 implied decimal places.This column will be populated when the item's price was overridden at the POS and the item's original unit retail is known.	Y if unit retail was manually entered	Right/0
	Taxable Indicator	Char(1)	Refer to 'YSNO' code_type for a list of valid codes.	Indicates whether or not item is taxable.	Y	Left/None
	Pump	Char(8)		Fuel pump identifier.	N	Left/Blank
	Reference Number 5	Char(30)		Number associated with a particular item within a transaction, for example special order number.The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank
	Reference Number 6	Char(30)		Second generic reference number at the item level.	N	Left/Blank
	Reference Number 7	Char(30)		Third generic reference number at the item level.	N	Left/Blank
	Reference Number 8	Char(30)		Fourth generic reference number at the item level.	N	Left/Blank
	Item_swiped_ind	Char(1)	Refer to 'YSNO' code_type for a list of valid codes.	Indicates if the item was automatically entered into the POS system or if it had to be manually keyed.	Y	Left/None

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Return Reason Code	Char(6)	Refer to 'SARR' code_type for a list of valid codes.	The reason an item was returned.	N	Left/Blank
	Salesperson	Char(10)		The salesperson who sold the item.	N	Left/Blank
	Expiration_date	Char(8)		Gift certificate expiration date (YYYYMMDD).	N	
Item Discount	File Type Record Descriptor	Char(5)	IDISC	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	RMS Promotion Number	Char(6)	Refer to 'PRMT' code_type for a list of valid types	The RMS promotion type.	Y	Left/Blank
	Discount Reference Number	Number(4)		Discount reference number is associated with the discount type (e.g. if discount type is a promotion, this contains the promotion number).	N	Left/Blank
	Discount Type	Char(6)	Refer to 'SADT' code_type for a list of valid types.	The type of discount within a promotion. This allows a retailer to further break down coupon discounts within the "In-store" promotion, for example.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Coupon Number	Char(16)		Number of a store coupon used as a discount.	Y if coupon	Left/Blank
	Coupon Reference Number	Char(16)		Additional information about the coupon, usually contained in a second bar code on the coupon.	Y if coupon	Left/Blank
	Quantity Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the quantity.	Y	Left/None
	Quantity	Number(12)		The quantity purchased that discount is applied with 4 implied decimal places.	Y	Right/0
	Unit Discount Amount	Number(20)		Unit discount amount for this item with 4 implied decimal places.	Y	Right/0
	Reference Number 13	Char(30)		Number associated with a particular transaction type at the discount level. The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank
	Reference Number 14	Char(30)		Second generic reference number at the discount level.	N	Left/Blank
	Reference Number 15	Char(30)		Third generic reference number at the discount level.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Reference Number 16	Char(30)		Fourth generic reference number at the discount level.	N	Left/Blank
Transaction Tax	File Type Record Descriptor	Char(5)	TTAX	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Tax Code	Char(6)	Refer to 'TAXC' code_type for a list of valid codes	Tax code to represent whether it is a state tax type, provincial tax, etc.	Y	Left/Blank
	Tax Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of Tax Amount.	Y	Left/None
	Tax Amount	Number(20)		Amount of tax charged for this tax code type in a transaction with 4 implied decimal places.	Y	Right/0
	Ref_no17	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank
	Ref_no18	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Ref_no19	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank
	Ref_no20	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank
Transaction Tender	File Type Record Descriptor	Char(5)	TTEND	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Tender Type Group	Char(6)	Refer to 'TENT' code_type for as list of valid types	High-level grouping of tender types.	Y	Left/Blank
	Tender Type ID	Number(6)	Refer to the pos_tender_type_head table for as list of valid types	Low-level grouping of tender types.	Y	Left/Blank
	Tender Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the value.	Y	Left/None
	Tender Amount	Number(20)		Amount paid with this tender in the transaction with 4 implied decimal places.	Y	Right/0

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Cc_no	Number(16)		Credit card number	Y if credit card	Left/Blank
	Cc_auth_no	Char(16)		Authorization number for a cc	Y if credit card	Left/Blank
	cc authorization source	Char(6)	Refer to 'CCAS' code_type for as list of valid types		Y if credit card	Left/Blank
	cc cardholder verification	Char(6)	Refer to 'CCVF' code_type for as list of valid types		Y if credit card	Left/Blank
	cc expiration date	Char(8)		(YYYYMMDD)	Y if credit card	Left/Blank
	cc entry mode	Char(6)	Refer to 'CCEM' code_type for as list of valid types	Indicates whether the credit card was swiped, thus automatically entered, or manually keyed.	Y if credit card	Left/Blank
	cc terminal id	Char(5)		Terminal number transaction was sent from.	N	Left/Blank
	cc special condition	Char(6)	Refer to 'CCSC' code_type for as list of valid types		Y if credit card	Left/Blank
	Voucher_no	Char(16)		Gift certificate or credit voucher serial number.	Y if voucher	Right/0
	Coupon Number	Char(16)		Number of a manufacturer’s coupon used as a tender.	Y if coupon	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
	Coupon Reference Number	Char(16)		Additional information about the coupon, usually contained in a second bar code on the coupon.	Y if coupon	Left/Blank
	Reference No 9	Char(30)		Number associated with a particular transaction type at the tender level. The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank
	Reference No 10	Char(30)		Second generic reference no at the tender level.	N	Left/Blank
	Reference No 11	Char(30)		Third generic reference no at the tender level.	N	Left/Blank
	Reference No 12	Char(30)		Fourth generic reference no at the tender level.	N	Left/Blank
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Transaction Record Counter	Number(10)		No of records processed in current tran (only records between trans head & tail)		

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/Padding
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	File Record Counter	Number(10)		No of transactions processed in current file (only records between file head & tail)	Y	Right/0

The RTLOG file is imported into the Sales Audit tables after validation by the batch program saimptlog. This section describes the requirements and validations performed on the records.

1 Common requirements/validations:

This section details the common requirements and validations performed on all transactions. The following sections describe the specific requirements of each type of transaction. If a transaction is not mentioned, then it does not have specific requirements.

a Record Type Requirements:

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
OPEN	No	No	No	No
NOSALE	No	Optional	No	No
VOID	Optional	Optional	Optional	Optional
PVOID	No	No	No	No
SALE	Yes	Yes	Optional	Optional

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
RETURN	Yes	Yes	Optional	Optional
EEXCH	Yes	No	Optional	Optional
PAIDIN	No	Yes	No	No
PAIDOU	No	Yes	No	No
PULL	No	Yes	No	No
LOAN	No	Yes	No	No
COND	No	No	No	No
CLOSE	No	No	No	No
TOTAL	No	No	No	No
REFUND	This transaction is not sent through the RTLOG. It is entered at the HQ level. The TITEM and TCUST records are optional. The TTEND record is required. A TTAX record should not be included.			
METER	Yes	No	No	No
PUMPT	Yes	No	No	No
TANKDP	Yes	No	No	No

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
TERM	TERM records are created by saimptlog and then loaded into the database. They do not come from the RTLOG file. They require one TITEM, one TTEND, one TTAX, one TCUST record and one CATT record.			
DCLOSE	No	No	No	No

b Requirements per record type:

Record Type	Requirements
IDISC	IDISC records must immediately follow their associated TITEM record.
CATT	CATT records must immediately follow their associated TCUST record.

c Code Type Validations:

Record Name	Field Name	Code Type
Transaction Header	Transaction Type	TRAT
	Sub-transaction Type	TRAS
	Reason Code	REAC or values from non_merch_code_head if the transaction type is ‘PAIDOU’ and the sub transaction type is ‘MV’ or ‘EV’.
	Value Sign	SIGN

Record Name	Field Name	Code Type
	Vender No	If the transaction type is ‘PAIDOU’ and the sub transaction type is ‘MV’, this field is validated against the supplier table. If the transaction type is ‘PAIDOU’ and the sub transaction type is ‘EV’, this field is validated against the partner table.
Transaction Item	Item Type	SAIT
	Item Status	SASI
	System Indicator	IMTP
	Quantity Sign	SIGN
	Taxable Indicator	YSNO
	Price Override Reason Code	ORRC
	Item Swiped Indicator	YSNO
	Return Reason Code	SARR
Item Discount	RMS Promotion Type	PRMT
	Discount Type	SADT
	Quantity Sign	SIGN
Transaction Customer	Customer ID Type	CIDT
Customer Attribute	Attribute Type	SACA
	Attribute value	Code types from codes in SACA.
Transaction Tax	Tax code	TAXC
	Tax sign	SIGN

Record Name	Field Name	Code Type
Transaction Tender	Tender Type Group	TENT
	Tender Sign	SIGN
	Tender Type ID	Pos_tender_type_head table
	CC Authorization Source	CCAS
	CC Cardholder Verification	CCVF
	CC Entry Mode	CCEM
	CC Special Condition	CCSC

d Dates are validated: Business Date, Transaction Date, Expiration Date Also, saimptlog accepts only business dates that are within the PERIOD.VDATE minus the SA_SYSTEM_OPTIONS.DAYS_POST_SALE value.

e Store number is validated against the STORE table.

f Numeric fields are checked for non-numeric characters.

g For transaction of type SALE, RETURN and EEXCH, saimptlog checks whether a transaction is in balance:

```
  Transaction Items (Unit Retail * Unit Retail Sign * Quantity)
+ Item Discounts (Unit Discount Amount * Unit Discount Sign * Quantity)
+ Transaction Tax (Tax Amount * Tax Sign)
= Transaction Tenders (Tender Amount * Tender Sign)
```

saimptlog will populate the Value field (on THEAD) with the transaction’s sales value (item value – discount value + tax value) from the above calculation if it was not provided in the RTLOG.

h Treatment of vouchers.

i If an item sold is a gift certificate (Transaction Item, Voucher field has a value), issued information is written to the SA_VOUCHER table.

ii If the Transaction Type is a RETURN, and the Transaction Tender Type Group is voucher (VOUCH), issued information is written to the SA_VOUCHER table.

iii If the Transaction Type is a SALE, and the Transaction Tender Type Group is a voucher (VOUCH), redeemed information is written to the SA_VOUCHER table.

iv When a gift certificate is sold, customer information should always be included. A receiving customer name value should be populated in the ref_no5 field, a receiving customer state value should be populated in the ref_no6 field and a receiving customer country should be populated in the ref_no7 field. These reference fields can be changed by updating the sa_reference table but the code needs to be modified too. The expiration date is put on the expiration_date field on the TITEM record.

i Other validations/points of interest:

i A salesperson in the TITEM record takes precedence over the salesperson in the THEAD record.

ii If an item sold is a UPC (Transaction Item, UPC field has a value and SKU does not), it will be converted to the corresponding SKU using the Supplement.

iii If an item sold is a SKU (Transaction Item, SKU field has a value), it will be validated against RMS item tables.

iv The corresponding Department, Class, Subclass, System Indicator and Taxable Indicator will be selected from the RMS tables and populated for a SKU.

j The balancing level determines whether the register or the cashier fields are required.

i If the balancing level is ‘R’egister, then the register field on the THEAD must be populated.

ii If the balancing level is ‘C’ashier, then the cashier field on the THEAD must be populated.

iii If the balancing level is ‘S’tore, then neither field is required to be populated.

k The tax_ind and the item_swiped_ind fields can only accept ‘Y’ or ‘N’ values. If an invalid value is passed through the RTLOG, an error will be flagged and the value will be defaulted to ‘Y’.

2 Transaction of type ‘SALE’:

A transaction of type SALE is generated whenever an item is sold. A sale may be to an employee, the sub-transaction type would be EMP in this case. Or it may be a drive-off sale (sub-transaction type DRIVEO) when someone drives off with unpaid gas. A special type of sale is an “odd exchange” (sub-transaction type EXCH) where items are sold and returned in the same transaction. If the net value of the exchange is positive, then it is a sale. If the net value is negative, it is a return. If the net value is zero and the items exchanged are in the same SKU style, it would be a transaction of type EEXCH (Even Exchange).

a Requirements per record type (other than what is described in Layout section above):

Record Type	Requirements
THEAD	

Record Type	Requirements
TITEM	• Item Status is a required field; it determines whether the item is ‘S’old, ‘R’eturned or ‘V’oided. If the item status is S, the quantity sign is expected to be P. If the item status is ‘R’, the quantity sign is expected to be N. · • If the item status is V, the quantity sign is the reverse of the quantity sign of the voided item. That is, if an item with status S is voided, the quantity sign would be N. Furthermore, the sum of the quantities being voided cannot exceed the sum of the quantities ‘S’old or ‘R’eturned. Note: neither of the above two validations are performed by saimptlog but an audit rule could be created to check this.· • In a typical sale, the items would all have a status of ‘S’. In the case of an odd exchange, some items will have a status of ‘R’.· • In a typical return, the items would all have a status of ‘R’. In the case of an odd exchange, some items will have a status of ‘S’.· • If an item has status R, then the Return Reason Code field may be populated. If it is, it will be validated against code type ‘SARR’.· • If the price of an item is overridden, then the Override Reason and Original Unit Retail fields must be populated.

Record Type	Requirements
IDISC	• The RMS Promotion Type field must always be populated with values of code type 'PRMT'. · • The Promotion field is validated, when a value is passed, against the promhead table. • If the promotion is 'In Store' (code 1004), then the Discount Type field must be populated with values of code type 'SADT'. • The Discount Reference Number is a promotion number which is of status 'A', 'E' or 'M'.· • If the Discount Type is 'SCOUP' for Store Coupon, then the Coupon Number field must be populated. The Coupon Reference Number field is optional.
TTEND	• If the tender type group is 'COUPON', then the Coupon Number field must be populated. The Coupon Reference Number field is optional.· • If the Transaction Tender Type Group is a credit card (CCARD), the number will be validated against the SA_CC_VAL table. The other cc fields are optional.

b Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Item Type	Tender Type Group	Reference Number Field	Meaning of Reference Field	Req?
SALE				1	Speed Sale Number	Y
SALE		GCN		5	Recipient Name	N
SALE		GCN		6	Recipient State	N
SALE		GCN		7	Recipient Country	N
SALE			CHECK	9	Check Number	N
SALE			CHECK	10	Driver's License Number	N
SALE			CHECK	11	Credit Card Number	N
SALE	DRIVEO			1	Incident Number	Y
SALE	EMP			3	Employee Number of the employee receiving the goods.	N

c Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
SALE	P if item is sold; N if item is returned; reverse of original item if item is voided.	P	P	P if item is sold; N if item is returned; reverse of original item if item is voided.

3 Transaction of type 'PVOID':

This transaction is generated at the register when another transaction is being post voided. The orig_tran_no and orig_reg_no fields must be populated with the appropriate information for the transaction being post voided. The PVOID transaction must be associated with the same store day as the original transaction. If the PVOID needs to be generated after the store day is closed, the transaction needs to be created using the forms.

4 Transaction of type ‘RETURN’:

This transaction is generated when a customer returns an item.

a This type of transaction has similar record type requirements as a ‘SALE’ transaction.

b Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Reference Number Field	Meaning of Reference Field	Req?
RETURN		1	Receipt Indicator (Y/N)	Y
RETURN		2	Refund Reference Number	N
RETURN	EMP	3	Employee Number of the employee returning the goods.	N

c Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
RETURN	P if item is sold; N if item is returned; reverse of original item if item is voided.	N	N	P if item is sold; N if item is returned; reverse of original item if item is voided.

5 Transaction of type ‘EEXCH’:

This transaction is generated when there is an even exchange.

a This type of transaction has similar record type requirements as a ‘SALE’ transaction.

b It is expected that the number of items returned equals the number of items sold. However, this validation is not performed by saimptlog. An audit rule could be created for this. Saimptlog only expects that there would be at least two item records.

c No tender changes hands in this transaction.

d Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Reference Number Field	Meaning of Reference Field	Req?
EEXCH		1	Receipt Indicator (Y/N)	Y
EEXCH	EMP	3	Employee Number of the employee exchanging the goods.	N

6 Transaction of type 'PAIDIN':

a This type of transaction has only one TTEND record.

b A reason code is required.

c Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reason Code	Reference Number Column	Meaning	Req?
NSF	1	NFS Check Credit Number	N
ACCT	1	Account Number	N

7 Transaction of type 'PAIDOU':

a This type of transaction has only one TTEND record.

b A reason code is required (code type REAC). If the sub-transaction type is 'EV' or 'MV', the reason code comes from the non_merch_codes_head table.

c If the sub-transaction type is 'EV' or 'MV', then at least one field among the vendor number, vendor invoice number, payment reference number and proof of delivery number fields should be populated.

d If the sub-transaction type is 'EV', then the vendor number comes from the partner table. If the sub-transaction type is 'MV', then the vendor number comes from the supplier table.

e Meaning of reference number fields:

Notes: The meaning of these reference number fields may be changed through the sa_reference table.

Sub Transaction Type	Reason Code	Reference Number Column	Meaning	Req?
EV		2	Personal ID Number	N
EV		3	Routing Number	N
EV		4	Account Number	N
	PAYRL	1	Money Order Number	N
	PAYRL	2	Employee Number	N
	INC	1	Incident Number	N

8 Transaction of type 'PULL':

This transaction is generated when cash is withdrawn from the register.

a This type of transaction has only one TTEND record.

b Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
PULL	N/A	N	N/A	N/A

9 Transaction of type 'LOAN':

This transaction is generated when cash is added to the register.

a This type of transaction has only one TTEND record.

b Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
LOAN	N/A	P	N/A	N/A

10 Transaction of type 'COND':

This transaction records the condition at the store when it opens. There can be at most one COND record containing weather information and at most one COND record containing temperature information. Both these pieces of information may be in the same COND record. There may be any number of COND records containing traffic and construction information.

a This type of transaction does not have TITEM or IDISC or TTAX or TTEND records.

b Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Weather – code type 'WEAT'	N
2	Temperature – a signed 3 digit number.	N
3	Traffic – code_type 'TRAF'	N
4	Construction – code_type 'CONS'	N

11 Transaction of type 'TOTAL':

This transaction records the totals that are reported by the POS. The value field must be populated. Some POS systems generate only one transaction number for all totals. In order to avoid duplicate errors to be reported, only one total transaction can have a transaction number and the subsequent ones can have blank transaction numbers. In other words, a TOTAL transaction is not required to have a transaction number.

a This type of transaction does not have TITEM or IDISC or TTAX or TTEND records.

12 Transaction of type 'METER':

This transaction is generated when a meter reading of a fuel pump is taken.

a This type of transaction has only TITEM records.

b Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Reading Type: (‘A’ Adjustment, ‘S’ shift change, ‘P’ price change, ‘C’ store close)	Y
5	Opening Meter Readings	Y
6	Closing Meter Reading	Y
7	If the reading type is ‘P’ for price change, the old unit retail should be placed here. Decimal places are required.	Y
8	Closing Meter Value	Y

13 Transaction of type ‘PUMPT’:

This transaction is generated when a pump test is performed. This type of transaction has only TITEM records.

14 Transactions of type ‘TANKDP’:

This transaction is generated when a tank dip measurement is taken.

a This type of transaction has only TITEM records.

b Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Tank identifier	Y
5	Dip Type (‘FUEL’, ‘WATER’, etc.)	Y
6	Dip Height Major (decimal places required)	Y
7	Dip Height Minor (decimal places required)	Y

15 Transaction of type ‘DCLOSE’:

This transaction is generated when day closed. Transaction number for this type of transaction has to be blank.

16 A note about vouchers: Vouchers are minimally handled by saimptlog. Voucher information is written to the savouch file which is passed to the program savouch.pc. For more information about this interface, see Interface File – SA Vouch and Batch Design – savouch.

A voucher will appear on the TITEM record only if it was sold. Thus when saimptlog encounters a ‘SALE’ transaction with a voucher, it writes the voucher to the savouch file as an ‘I’ssued voucher.

A voucher will be issued when it appears on the TTEND record of transactions of type ‘RETURN’ and ‘PAIDOU’. In other words, saimptlog will write it to the savouch file with status ‘I’.

A voucher will be redeemed when it appears on the TTEND record of transactions of type ‘SALE’ and ‘PAIDIN’. In other words, saimptlog will write it to the savouch file with status ‘R’.

Vouchers may not be returned. However, a transaction of type ‘PAIDOU’ may be generated when the customer exchanges a voucher for another form of tender.

Chapter 6 – Fashion merchandise hierarchy download (fmednldf.doc)

Modification

Modified the first paragraph of the design overview.

Design overview

fmednldf.pc is designed to extract the product hierarchy for all the SKUs into the output file. Sending all SKUs to the output file allows for flexibility and reusability so that many programs and applications can use this module to retrieve the hierarchy information for fashion items. This module will send the full merchandise hierarchy to each domain each night since changes could occur on a daily basis (clients may schedule this batch program, at their discretion, to run daily, weekly, etc.).

Before processing styles, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a size limitation for the multidimensional database, so they may declare multiple instances (or domains) of the database to optimize performance. The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/store combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_dept, domain_class, and domain_subclass tables maintain relationships between the chosen level (department, class or subclass) and the domain.

Scheduling constraints

Processing Cycle:	Daily
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart recovery

The logical unit of work is SKU. The commit (move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Fashion SKU Driving Cursors:

declare cursors - all ticks (') and quotes (") are replaced with spaces, fashion sku description is cut back to 50 characters

Department

```
EXEC SQL DECLARE C_fashion_dept CURSOR FOR
 SELECT lpad(to_char(rag_skus.sku),8,'0'),
          lpad(to_char(rag_skus.style),8,'0'),
          REPLACE(REPLACE(rag_style.style_desc,'''','
'),'"',' '),
          lpad(to_char(colour.colour),4,'0'),
          REPLACE(REPLACE(colour.colour_desc,'''','
'),'"',' '),
          rag_skus.size1,
          REPLACE(REPLACE(s1.size_desc,'''',' '),'"','
'),
          NVL(rag_skus.size2,' '),
          REPLACE(REPLACE(NVL(s2.size_desc,' '),'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0')||
          lpad(to_char(rag_style.class),4,'0')||
          lpad(to_char(rag_style.subclass),4,'0'),
          REPLACE(REPLACE(subclass.sub_name,'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0')||
          lpad(to_char(rag_style.class),4,'0'),
          REPLACE(REPLACE(class.class_name,'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0'),
          REPLACE(REPLACE(deps.dept_name,'''',' '),'"','
'),
          lpad(to_char(deps.group_no),4,'0'),
          REPLACE(REPLACE(groups.group_name,'''','
'),'"',' '),
          lpad(to_char(groups.division),4,'0'),
          REPLACE(REPLACE(division.div_name,'''','
'),'"',' '),
          lpad(to_char(its.supplier),10,'0'),
          REPLACE(REPLACE(sups.sup_name,'''',' '),'"','
'),
          rag_style.forecast_ind,
          lpad(to_char(NVL(domain_dept.domain_id,
0)),2,'0')
     FROM rag_skus,
          rag_style,
          subclass,
          class,
          deps,
          colour,
          groups,
          division,
          domain_dept,
          item_supplier its,
          sups,
          sizes s1,
          sizes s2,
          v_restart_dept
    WHERE v_restart_dept.driver_name  = 'DEPT'
      AND v_restart_dept.num_threads  = :pi_num_threads
      AND v_restart_dept.thread_val   = :pi_thread_val
      AND v_restart_dept.driver_value = rag_style.dept
      AND rag_style.style = rag_skus.style
      AND rag_skus.size1 = s1.size_id
      AND rag_skus.size2 = s2.size_id(+)
      AND rag_skus.colour = colour.colour
      AND rag_style.dept = deps.dept
      AND rag_style.dept = domain_dept.dept(+)
      AND rag_style.dept = class.dept
      AND rag_style.class = class.class
      AND rag_style.dept = subclass.dept
      AND rag_style.class = subclass.class
      AND rag_style.subclass = subclass.subclass
      AND deps.group_no = groups.group_no
      AND groups.division = division.division
      AND rag_skus.sku > NVL(:os_restart_sku, -999)
      AND its.item = rag_skus.sku
      AND its.primary_supp_ind = 'Y'
      AND its.supplier = sups.supplier
 ORDER BY rag_skus.sku;
```

Class

```
EXEC SQL DECLARE C_fashion_class CURSOR FOR
 SELECT lpad(to_char(rag_skus.sku),8,'0'),
          lpad(to_char(rag_skus.style),8,'0'),
          REPLACE(REPLACE(rag_style.style_desc,'''','
'),'"',' '),
          lpad(to_char(colour.colour),4,'0'),
          REPLACE(REPLACE(colour.colour_desc,'''','
'),'"',' '),
          rag_skus.size1,
          REPLACE(REPLACE(s1.size_desc,'''',' '),'"','
'),
          NVL(rag_skus.size2,' '),
          REPLACE(REPLACE(NVL(s2.size_desc,' '),'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0')||
          lpad(to_char(rag_style.class),4,'0')||
          lpad(to_char(rag_style.subclass),4,'0'),
          REPLACE(REPLACE(subclass.sub_name,'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0')||
          lpad(to_char(rag_style.class),4,'0'),
          REPLACE(REPLACE(class.class_name,'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0'),
          REPLACE(REPLACE(deps.dept_name,'''',' '),'"','
'),
          lpad(to_char(deps.group_no),4,'0'),
          REPLACE(REPLACE(groups.group_name,'''','
'),'"',' '),
          lpad(to_char(groups.division),4,'0'),
          REPLACE(REPLACE(division.div_name,'''','
'),'"',' '),
          lpad(to_char(its.supplier),10,'0'),
          REPLACE(REPLACE(sups.sup_name,'''',' '),'"','
'),
          rag_style.forecast_ind,
          lpad(to_char(NVL(domain_class.domain_id,
0)),2,'0')
     FROM rag_skus,
          rag_style,
          subclass,
          class,
          deps,
          colour,
          groups,
          division,
          domain_class,
          item_supplier its,
          sups,
          sizes s1,
          sizes s2,
          v_restart_dept
   WHERE v_restart_dept.driver_name  = 'DEPT'
     AND v_restart_dept.num_threads  = :pi_num_threads
     AND v_restart_dept.thread_val   = :pi_thread_val
     AND v_restart_dept.driver_value = rag_style.dept
     AND rag_style.style = rag_skus.style
     AND rag_skus.size1 = s1.size_id
     AND rag_skus.size2 = s2.size_id(+)
     AND rag_skus.colour = colour.colour
     AND rag_style.dept = deps.dept
     AND rag_style.dept = domain_class.dept(+)
     AND rag_style.class = domain_class.class(+)
     AND rag_style.dept = class.dept
     AND rag_style.class = class.class
     AND rag_style.dept = subclass.dept
     AND rag_style.class = subclass.class
     AND rag_style.subclass = subclass.subclass
     AND deps.group_no = groups.group_no
     AND groups.division = division.division
     AND rag_skus.sku > NVL(:os_restart_sku, -999)
     AND its.item = rag_skus.sku
     AND its.primary_supp_ind = 'Y'
     AND its.supplier = sups.supplier
ORDER BY rag_skus.sku;
```

Subclass

```
EXEC SQL DECLARE C_fashion_subclass CURSOR FOR
 SELECT lpad(to_char(rag_skus.sku),8,'0'),
          lpad(to_char(rag_skus.style),8,'0'),
          REPLACE(REPLACE(rag_style.style_desc,'''','
'),'"',' '),
          lpad(to_char(colour.colour),4,'0'),
          REPLACE(REPLACE(colour.colour_desc,'''','
'),'"',' '),
          rag_skus.size1,
          REPLACE(REPLACE(s1.size_desc,'''',' '),'"','
'),
          NVL(rag_skus.size2,' '),
          REPLACE(REPLACE(NVL(s2.size_desc,' '),'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0')||
          lpad(to_char(rag_style.class),4,'0')||
          lpad(to_char(rag_style.subclass),4,'0'),
          REPLACE(REPLACE(subclass.sub_name,'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0')||
          lpad(to_char(rag_style.class),4,'0'),
          REPLACE(REPLACE(class.class_name,'''','
'),'"',' '),
          lpad(to_char(rag_style.dept),4,'0'),
          REPLACE(REPLACE(deps.dept_name,'''',' '),'"','
'),
          lpad(to_char(deps.group_no),4,'0'),
          REPLACE(REPLACE(groups.group_name,'''','
'),'"',' '),
          lpad(to_char(groups.division),4,'0'),
          REPLACE(REPLACE(division.div_name,'''','
'),'"',' '),
          lpad(to_char(its.supplier),10,'0'),
          REPLACE(REPLACE(sups.sup_name,'''',' '),'"','
'),
          rag_style.forecast_ind,
          lpad(to_char(NVL(domain_subclass.domain_id,
0)),2,'0')
     FROM rag_skus,
          rag_style,
          subclass,
          class,
          deps,
          colour,
          groups,
          division,
          domain_subclass,
          item_supplier its,
          sups,
          sizes s1,
          sizes s2,
          v_restart_dept
    WHERE v_restart_dept.driver_name  = 'DEPT'
      AND v_restart_dept.num_threads  = :pi_num_threads
      AND v_restart_dept.thread_val   = :pi_thread_val
      AND v_restart_dept.driver_value = rag_style.dept
      AND rag_style.style = rag_skus.style
      AND rag_skus.size1 = s1.size_id
      AND rag_skus.size2 = s2.size_id(+)
      AND rag_skus.colour = colour.colour
      AND rag_style.dept = deps.dept
      AND rag_style.dept = domain_subclass.dept(+)
      AND rag_style.class = domain_subclass.class(+)
      AND rag_style.subclass =
domain_subclass.subclass(+)
      AND rag_style.dept = class.dept
      AND rag_style.class = class.class
      AND rag_style.dept = subclass.dept
      AND rag_style.class = subclass.class
      AND rag_style.subclass = subclass.subclass
      AND deps.group_no = groups.group_no
      AND groups.division = division.division
      AND rag_skus.sku > NVL(:os_restart_sku, -999)
      AND its.item = rag_skus.sku
      AND its.primary_supp_ind = 'Y'
      AND its.supplier = sups.supplier
 ORDER BY rag_skus.sku;
```

Program flow

Pseudo-code (note functions shown in-line, should be defined outside main)

```
main(int argc, char* argv[])
{
    extract login ID, password, thread value from input arguments

    declare_fashion_cursor(pass_thread_value);
    open_fashion_cursor(pass_thread_value);
    fetch_fashion_cursor(pass_thread_value);

    while(1)
    {
        if record counter < global maximum counter
            write to product master (merch hier.) temp file
            increment counter of records written
        else
            call restart logic to cat temp Prod. Master file to
final file
            reset counter of records written
        end-if

        fetch_fashion_cursor(pass_thread_value);
    }

Function declarations

declare_fashion_cursor(pass_thread_value)
{
    switch pass_thread_value
        {
        case ‘D’
            declare f_domain_dept cursor
        case ‘C’
            declare f_domain_class cursor
```

```
                case 'S'
                        declare f_domain_subclass cursor
                default:
                        sprintf(err_date, "undefined level %c for declare
fashion cursor", pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, "declare %c_level fashion cursor",
pass_thread_value)
}

open_fashion_cursor(pass_thread_value)
{
        switch pass_thread_value
                {
                case 'D':
                        open f_domain_dept cursor
                case 'C':
                        open f_domain_class cursor
                case 'S':
                        open f_domain_subclass cursor
                default:
                        sprintf(err_date, "undefined level %c for open
fashion cursor", pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, "open %c_level fashion cursor",
pass_thread_value)
}

fetch_fashion_cursor(pass_thread_value)
{
        switch pass_thread_value
                {
                case 'D':
                        open f_domain_dept cursor
                case ‘C’:
                        open f_domain_class cursor
                case ‘S’:
                        open f_domain_subclass cursor
                default:
                        sprintf(err_date, “undefined level %c for open
fashion cursor”, pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, “fetcg %c_level fashion cursor”,
pass_thread_value)
}

close_fashion_cursor(pass_thread_value)
{
        switch pass_thread_value
                {
                case ‘D’:
                        close f_domain_dept cursor
                case ‘C’:
                        close f_domain_class cursor
                case ‘S’:
                        close f_domain_subclass cursor
                default:
                        sprintf(err_date, “undefined level %c for close
fashion cursor”, pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, “close %c_level fashion cursor”,
pass_thread_value)
}
```

Shared modules

N/A

Function level description

N/A

I/O specification

Output files fmhiernn.date (where nn equals the thread value – this is not equal to the domain_id)

SKU	Varchar2	20
SKU desc	Varchar2	30
Color code	Varchar2	4
Color desc	Varchar2	24
Size 1 code	Varchar2	6
Size 1 desc	Varchar2	30
Size 2 code	Varchar2	6
Size 2 desc	Varchar2	30
Style	Varchar2	20
Style desc	Varchar2	40
Subclass	Varchar2	20 – dept+class+subclass
Subclass desc	Varchar2	40
Class	Varchar2	20 – dept+class
Class desc	Varchar2	40
Dept	Varchar2	20 – dept
Dept desc	Varchar2	40
Group	Varchar2	20
Group name	Varchar2	40
Division	Varchar2	20
Division name	Varchar2	40
Supplier	Varchar2	20
Supplier name	Varchar2	40
Forecast Indicator	Varchar2	1
Domain ID	Varchar2	2

Technical issues

N/A

Chapter 7 – Staple merchandise hierarchy download (fmednlds.doc)

Modification

Modified the first paragraph of the design overview.

Design overview

fmednlds.pc is designed to extract the product hierarchy for all staple SKUs into the output file. Sending all SKUs to the output file allows for flexibility and reusability so that many programs and applications can use this module to retrieve the hierarchy information for staple items. This module will send the full merchandise hierarchy to each domain each night since changes could occur on a daily basis (clients may schedule this batch program, at their discretion, to run daily, weekly, etc.).

Before processing any SKUs, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a size limitation for the multidimensional database, so they may declare multiple instances (or domains) of the database to optimize performance. The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/store combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_dept, domain_class, and domain_subclass tables maintain relationships between the chosen level (department, class or subclass) and the domain.

Scheduling constraints

Processing Cycle:	Daily
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart recovery

The logical unit of work is SKU. The commit (move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Staple SKU driving cursors:

Declare cursors - all ticks (') and quotes (") are replaced with spaces

Department

```
EXEC SQL DECLARE C_staple_dept CURSOR FOR
   SELECT lpad(to_char(win_skus.sku),8,'0'),
          REPLACE(REPLACE(win_skus.sku_desc,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0')||
          lpad(to_char(win_skus.class),4,'0')||
          lpad(to_char(win_skus.subclass),4,'0'),
          REPLACE(REPLACE(subclass.sub_name,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0')||
          lpad(to_char(win_skus.class),4,'0'),
          REPLACE(REPLACE(class.class_name,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0'),
          REPLACE(REPLACE(deps.dept_name,'''',' '),'"','
'),
          lpad(to_char(deps.group_no),4,'0'),
          REPLACE(REPLACE(groups.group_name,'''','
'),'"',' '),
          lpad(to_char(groups.division),4,'0'),
          REPLACE(REPLACE(division.div_name,'''','
'),'"',' '),
          lpad(to_char(item_supplier.supplier),10,'0'),
          REPLACE(REPLACE(sups.sup_name,'''',' '),'"','
'),
          win_skus.forecast_ind,
          lpad(to_char(domain_dept.domain_id),2,'0')
     FROM win_skus,
          subclass,
          class,
          deps,
          groups,
          division,
          item_supplier,
          sups,
          domain_dept,
          v_restart_dept
    WHERE v_restart_dept.driver_name  = 'DEPT'
```

```
     AND v_restart_dept.num_threads  = :pi_num_threads
     AND v_restart_dept.thread_val   = :pi_thread_val
     AND v_restart_dept.driver_value = win_skus.dept
     AND win_skus.dept = deps.dept
     AND win_skus.dept = domain_dept.dept(+)
     AND win_skus.dept = class.dept
     AND win_skus.class = class.class
     AND win_skus.dept = subclass.dept
     AND win_skus.class = subclass.class
     AND win_skus.subclass = subclass.subclass
     AND win_skus.sku = item_supplier.item
     AND item_supplier.supplier = sups.supplier
     AND item_supplier.primary_supp_ind = 'Y'
     AND deps.group_no = groups.group_no
     AND groups.division = division.division
     AND win_skus.sku > NVL(:os_restart_sku, -999)
ORDER BY win_skus.sku;
```

Class

```
EXEC SQL DECLARE C_staple_class CURSOR FOR
   SELECT lpad(to_char(win_skus.sku),8,'0'),
          REPLACE(REPLACE(win_skus.sku_desc,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0')||
          lpad(to_char(win_skus.class),4,'0')||
          lpad(to_char(win_skus.subclass),4,'0'),
          REPLACE(REPLACE(subclass.sub_name,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0')||
          lpad(to_char(win_skus.class),4,'0'),
          REPLACE(REPLACE(class.class_name,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0'),
          REPLACE(REPLACE(deps.dept_name,'''',' '),'"','
'),
          lpad(to_char(deps.group_no),4,'0'),
          REPLACE(REPLACE(groups.group_name,'''','
'),'"',' '),
          lpad(to_char(groups.division),4,'0'),
          REPLACE(REPLACE(division.div_name,'''','
'),'"',' '),
          lpad(to_char(item_supplier.supplier),10,'0'),
          REPLACE(REPLACE(sups.sup_name,'''',' '),'"','
'),
          win_skus.forecast_ind,
          lpad(to_char(domain_class.domain_id),2,'0')
     FROM win_skus,
          subclass,
          class,
          deps,
          groups,
          division,
          item_supplier,
          sups,
          domain_class,
          v_restart_dept
    WHERE v_restart_dept.driver_name  = 'DEPT'
      AND v_restart_dept.num_threads  = :pi_num_threads
      AND v_restart_dept.thread_val   = :pi_thread_val
      AND v_restart_dept.driver_value = win_skus.dept
      AND win_skus.dept = deps.dept
      AND win_skus.dept = domain_class.dept(+)
      AND win_skus.class = domain_class.class(+)
      AND win_skus.dept = class.dept
      AND win_skus.class = class.class
      AND win_skus.dept = subclass.dept
      AND win_skus.class = subclass.class
      AND win_skus.subclass = subclass.subclass
      AND win_skus.sku = item_supplier.item
      AND item_supplier.supplier = sups.supplier
      AND item_supplier.primary_supp_ind = 'Y'
      AND deps.group_no = groups.group_no
      AND groups.division = division.division
      AND win_skus.sku > NVL(:os_restart_sku, -999)
 ORDER BY win_skus.sku;
```

Subclass

```
EXEC SQL DECLARE C_staple_subclass CURSOR FOR
   SELECT lpad(to_char(win_skus.sku),8,'0'),
          REPLACE(REPLACE(win_skus.sku_desc,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0')||
          lpad(to_char(win_skus.class),4,'0')||
          lpad(to_char(win_skus.subclass),4,'0'),
          REPLACE(REPLACE(subclass.sub_name,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0')||
          lpad(to_char(win_skus.class),4,'0'),
          REPLACE(REPLACE(class.class_name,'''','
'),'"',' '),
          lpad(to_char(win_skus.dept),4,'0'),
          REPLACE(REPLACE(deps.dept_name,'''',' '),'"','
'),
          lpad(to_char(deps.group_no),4,'0'),
          REPLACE(REPLACE(groups.group_name,'''','
'),'"',' '),
          lpad(to_char(groups.division),4,'0'),
          REPLACE(REPLACE(division.div_name,'''','
'),'"',' '),
          lpad(to_char(item_supplier.supplier),10,'0'),
          REPLACE(REPLACE(sups.sup_name,'''',' '),'"','
'),
          win_skus.forecast_ind,
          lpad(to_char(domain_subclass.domain_id),2,'0')
     FROM win_skus,
          subclass,
          class,
          deps,
          groups,
          division,
          item_supplier,
          sups,
          domain_subclass,
          v_restart_dept
    WHERE v_restart_dept.driver_name  = 'DEPT'
```

```
      AND v_restart_dept.num_threads  = :pi_num_threads
      AND v_restart_dept.thread_val   = :pi_thread_val
      AND v_restart_dept.driver_value = win_skus.dept
      AND win_skus.dept = deps.dept
      AND win_skus.dept = domain_subclass.dept(+)
      AND win_skus.class = domain_subclass.class(+)
      AND win_skus.subclass =
domain_subclass.subclass(+)
      AND win_skus.dept = class.dept
      AND win_skus.class = class.class
      AND win_skus.dept = subclass.dept
      AND win_skus.class = subclass.class
      AND win_skus.subclass = subclass.subclass
      AND win_skus.sku = item_supplier.item
      AND item_supplier.supplier = sups.supplier
      AND item_supplier.primary_supp_ind = 'Y'
      AND deps.group_no = groups.group_no
      AND groups.division = division.division
      AND win_skus.sku > NVL(:os_restart_sku,-999)
 ORDER BY win_skus.sku;
```

Program flow

Pseudo-code (note functions shown in-line, should be defined outside main)

```
main(int argc, char* argv[])
{
    extract login ID, password, thread value from input arguments

    declare_staple_cursor(pass_thread_value);
    open_staple_cursor(pass_thread_value);
    fetch_staple_cursor(pass_thread_value);

    while(1)
    {
        if record counter < global maximum counter
            write to product master (merch hier.) temp file
            increment counter of records written
```

```
                else
                        call restart logic to cat temp Prod. Master file to
final file
                        reset counter of records written
                end-if

                fetch_staple_cursor(pass_thread_value);
        }
```

Function declarations

```
declare_staple_cursor(pass_thread_value) – job department/thread value
{
        switch pass_thread_value
                {
                case ‘D’:
                        declare s_domain_dept cursor
                case ‘C’:
                        declare s_domain_class cursor
                case ‘S’:
                        declare s_domain_subclass cursor
                default:
                        sprintf(err_date, “undefined level %c for declare
staple cursor”, pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, “declare %c_level staple cursor”,
pass_thread_value)
}

open_staple_cursor(pass_thread_value)
{
        switch pass_thread_value
                {
```

```
                case ‘D’:
                        open s_domain_dept cursor
                case ‘C’:
                        open s_domain_class cursor
                case ‘S’:
                        open s_domain_subclass cursor
                default:
                        sprintf(err_date, “undefined level %c for open
staple cursor”, pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, “open %c_level staple cursor”,
pass_thread_value)
}

fetch_staple_cursor(pass_thread_value)
{
        switch pass_thread_value
                {
                case ‘D’:
                        open s_domain_dept cursor
                case ‘C’:
                        open s_domain_class cursor
                case ‘S’:
                        open s_domain_subclass cursor
                default:
                        sprintf(err_date, “undefined level %c for open
staple cursor”, pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, “fetch %c_level staple cursor”,
pass_thread_value)
}

open_staple_cursor(pass_thread_value)
```

```
{
        switch pass_thread_value
                {
                case ‘D’:
                        open s_domain_dept cursor
                case ‘C’:
                        open s_domain_class cursor
                case ‘S’:
                        open s_domain_subclass cursor
                default:
                        sprintf(err_date, “undefined level %c for open
staple cursor”, pass_thread_value)
                }
        if SQL error found
                sprintf(err_date, “open %c_level staple cursor”,
pass_thread_value)
}
```

Shared modules

N/A

Function level description

N/A

I/O specification

Output file smhiernn.dat (where nn equals the thread value – this is not the domain_id)

SKU	Varchar2	20 – Populate with SKU
SKU desc	Varchar2	70 – Populate with SKU description
Style	Varchar2	20 – NULL
SKU desc	Varchar2	40
Subclass	Varchar2	20 – dept+class+subclass
Subclass desc	Varchar2	40
Class	Varchar2	20 – dept+class
Class desc	Varchar2	40
Dept	Varchar2	20 – dept
Dept desc	Varchar2	40
Group	Varchar2	20
Group name	Varchar2	40
Division	Varchar2	20
Division name	Varchar2	40
Supplier	Varchar2	20
Supplier name	Varchar2	40
Forecast Indicator	Varchar2	1
Domain ID	Varchar2	2

Technical issues

N/A

Chapter 8 – Product security rebuild (sprdrbld.doc)

Modification

Modified program to include security at dept.level.

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. Security will be set to either ‘S’KU or ‘D’epartment level on system_options.security_lvl_ind. SKUs/Dept 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 SKU/Dept 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 SKU in the department and in the class will be allowed to update. For the rest of the SKUs 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 SKU or Dept. 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 SKUs/depts in the particular thread running. This array will be loaded with all the SKUs/depts 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 SKUs/depts have a rule affecting them and have therefore been “touched”. Each rule will affect certain SKUs/depts 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/DEPS_MATRIX	No	Yes	No	No	No
V_RESTART_DEPT	No	Yes	No	No	No
DESC_LOOK	No	Yes	No	No	No
RAG_SKUS	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 (for SKUs level) or sdepsrbld.ctl (for Dept. level) to load the output file from sprdrbld.pc to database.

Pre-Processing: Prepost with parameters: sprdrbld pre

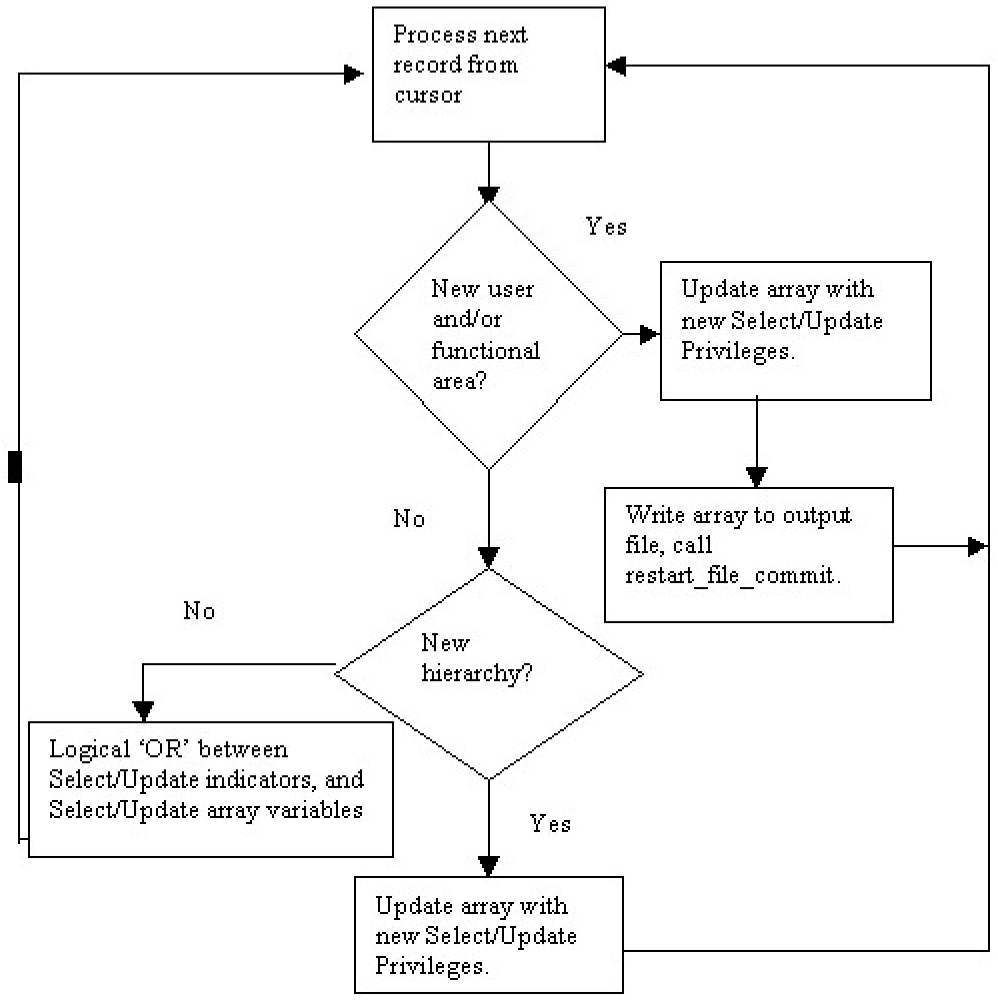
Post-Processing: Prepost with parameters: sprdrbld post

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 styles/skus 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 styles/skus can have much larger commit counters since fewer rows will be inserted into the database each time for one user-functional area.

Program flow


Process next record from cursor
New user and/or functional area?
Yes
Update array with new Select/Update Privileges.
Write array to output file, call restart_file_commit.
No
New hierarchy?
No
Logical 'OR' between Select/Update indicators, and Select/Update array variables
Yes
Update array with new Select/Update Privileges.

Shared modules

Driving Cursor for SKU

```
SELECT u.user_id,
            p.column_code,
            p.dept,
            p.class,
            p.subclass,
            p.style,
            p.sku,
            p.select_ind,
            p.update_ind
 FROM sec_user_group u,
           sec_group_prod_matrix p,
           v_restart_dept v
WHERE u.group_id = p.group_id
     AND v.driver_value = p.dept
     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 p.column_code > :ps_restart_column_code))
ORDER BY u.user_id, p.column_code, p.dept, p.class desc, p.subclass desc,
                p.sku desc, p.style desc;
```

Driving Cursor for Dept

```
EXEC SQL DECLARE c_sec_dep_prd_policy CURSOR FOR
   SELECT u.user_id,
             p.column_code,
             p.dept,
             p.select_ind,
             p.update_ind
     FROM sec_user_group u,
        sec_group_prod_matrix p,
        v_restart_dept v
    WHERE u.group_id = p.group_id
     AND v.driver_value = p.dept
     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 p.column_code > :ps_restart_column_code))
   ORDER BY u.user_id, p.column_code, p.dept;
```

Function level description

Main()

N/A

Init()

- Check SYSTEM_VARIABLES.update_prd_sec_ind. If the indicator is not set then the program exit normally without further processing.
- Check SYSTEM_OPTIONS.security_lvl_ind to determine if security is set at dept. or sku level.
- Call retek_init() to get restart-recover variables.
- Get_total_skus()/depts.()
- Get total skus in the current thread.
- Size_sku/dept_array()
- Size sku array based on the number of skus in the current thread. The sku array includes dept, class, subclass, style, style_ind, sku, select_ind, update_ind and touched columns.

- Load_sku/dept_array()
- Load all skus in the current thread to the sku 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. The rule array includes user_id, column_code, dept, class, class_ind, subclass, subclass_ind, style, style_ind, sku, sku_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 dept, class, subclass, style, and SKU 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/style/sku 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/dept level according to the last record rule.
- If it is new:
- Call update_array() to blow security rule down to the SKU/dept level according to the last record rule.
- Call write_array() to output the security rules of last record's user/functional area (down to SKU/dept level) to SQL load file.
- Call retek_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 SKU array according to the input security rule. There are five kinds of security rules. They are department, class, subclass, style and SKU level security rules.

- If the input rule is a department level security rule, then loop through the SKU array, for all the SKUs within the department, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.
- If the input rule is a class level security rule, then loop through the SKU array, for all the SKUs within the class, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.
- If the input rule is a subclass level security rule, then loop through the SKU array, for all the SKUs within the subclass, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.
- If the input rule is a style level security rule, then loop through the SKU array, for all the SKUs corresponding to the style, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.
- If the input rule is a SKU level security rule, then loop through the SKU array, set the select_ind and update_ind of the SKU equal to the input rule's select_ind and update_ind, respectively. Set 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.extension_x (x is current thread number) corresponds to one record row in the sec_user_prod_matrix or sec_user_deps_matrix table. The format of the output file is as follows:

Column_code;user_id;SKU/dept;select_ind;update_ind

Example:

PPRM;JOHN;10007986;N;N

PPRC;CLINTON;10001000;Y;N

PPRM;CLINTON;10007986;Y;Y

…

Technical issues

N/A

Chapter 9 – Transfer shipments upload (tsfoupld.doc)

Modification

Changed quanity_shipped number(6) to number(12).

Design overview

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

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

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	No	Yes	Yes	Yes	No
TSFDETAIL	No	Yes	Yes	Yes	No
SHIPMENT	No	Yes	Yes	Yes	No
SHIPSKU	No	Yes	Yes	Yes	No
POS_MODS	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_STORE	No	Yes	No	Yes	No
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No
RAG_SKUS	No	Yes	No	No	No
RAG_STYLE_ST	No	Yes	No	No	No
RAG_STYLE_WH	No	Yes	No	No	No
INV_STATUS_QTY	No	Yes	No	Yes	Yes
INV_STATUS_TYPES	No	Yes	No	No	No

Scheduling constraints

Processing Cycle: PHASE 2 (daily)

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

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE and WH

Threads driven by number of distinct files

Restart recovery

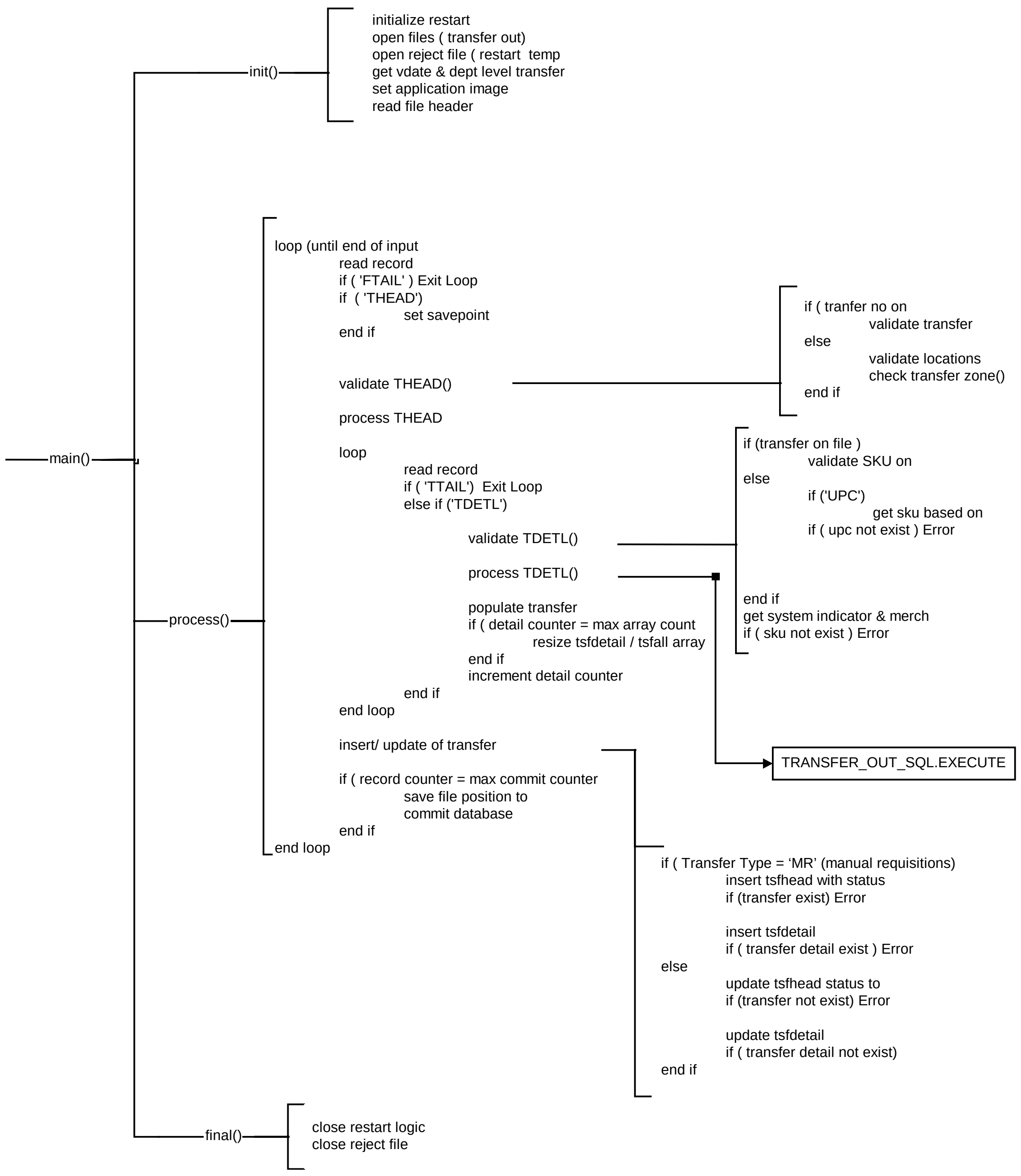
The logical unit of work for the transfer out module is the discrete transfer transaction. Each transfer will be identified by the transfer number (if it already exists in the Retek system) or an unique transaction set number generated by the external system. This transfer transaction will be defined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

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

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

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

Program flow



Shared modules

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

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

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

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

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

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

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

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

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

Function level description

init()

```
declare structure arrays for tsfdetail
initialize restart recovery

open input file ( transfer out )
    - file should be specified as input parameter to program
open reject file ( as a temporary file for restart )
    - file should be specified as input parameter to program
get vdate and department level transfer indicator from period table and system
options
set application image array - save the line counter
read file header record
if (record type <> 'FHEAD')  Fatal Error
```

process()

```
loop
    read record from input file
    if ( 'FTAIL' )
        Exit Loop
    end if
    if  ( 'THEAD')
        set savepoint and save current file pointer position
        validate_THEAD()

        reset detail count
        process_THEAD()
    end if

    loop
        check carton flag to determine if tdetl records will be for a carton
        or not
        read record from input file (different structure for carton or
        regular)
        if ( 'TTAIL')  Exit Loop
```

```
            if ('TDETL')
                validate_TDETL()
                process_TDETL()
            end if

            if ( detail count = max array count )
                resize array structures for tsfdetail
                increase max array count
            end if
            increment detail count
        end loop

        if ( no errors  )
            post_transfers() (don’t call this if doing a carton)
        end if
        if ( non Fatal Error Encountered )
            reject_record  - call write error and pass file pointer as of last
            savepoint and current file pointer
            Rollback transaction
        end if

        if ( transaction count > max commit count )
            restart file commit
                - save the current input file pointer position
                - save the line counter in restart image
        end if
    end loop

    restart commit final
```

validate_THEAD()

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

format_header_fields()

if ( shipment number provided in transaction )

    validate that the shipment number exists within Retek for a transfer.
    (check on shipment)
    validate that the transfer within Retek has a status of 'A', 'E', 'S', 'C'
    (approved, extracted, shipped, closed) and is applicable to the
        to/from locations specified (check on tsfhead) – also fetch
        transfer type

    if shipment number provided does not exist on shipment in 'I', 'R' status
    for a transfer then
        raise Non-Fatal Error
    if transfer does not exist in Retek with the appropriate status and
    locations then
        raise Non-fatal error
else if ( no shipment number is provided )
    if (external shipment number provided)
        - validate to and from locations
        if ( loc_type = 'S' )
            check for existence on store table
        else ( loc_type = 'W' )
            check for existence on wh table
        end if
        if any location not exist, write non-Fatal error

        - validate common transfer zone for store to store transfers
        if ( to_loc type = 'S' and from_loc = 'S')
            check transfer zone - select transfer zone of the from
            location and the to location.
```

```
                        if ( from_loc transfer zone <> to _loc transfer zone )
                                write non-Fatal Error ( transfer zones
                                incompatible )
                        end if
                end if
        else (no external shipment number)
                All detail records must have a allocation number.
        end if
end if
```

process_THEAD()

```
check for a bill of lading in 0 - open status for the destination location
retrieve the bill of lading number if one exists
if ( bill of lading does not exist )
        get next bill of lading number
        insert bill of lading header ( lad_head ) record
end if

if bol number passed in  ensure it is valid.
If it is not valid get next bol number.

if transfer type = ‘CO’
        retrieve delivery type from the ORDCUST table
end if
```

validate_TDETL()

```
format_detail_fields()

if inventory status field is not blank, validate it against inv_status_types table

if no shipment / ext shipment in file
    every detail line must have an allocation.

if (shipment number in file )
    validate item exists on the transfer
    else
    if ( Item Type = ‘UPC’ )
        select sku from upc_ean based on the upc and supplement
        if ( upc does not exist )
            write non-Fatal Error ( upc not found )
        end if
    else if ( Item Type = ‘SKU’ )
        SKU = item value from the input file
        case ID = ‘ ’
    end if
end if

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

- get item system indicator, department, class and subclass
if ( system indicator does not exist )
    write non-Fatal Error ( sku not found )
end if
```

process_TDETL()

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

```
if :oi_new_tsf_flag = 1 then
  if :os_store_rcv_type = 'A' then
    L_upd_resv_ind   := 'N';
    L_upd_intran_ind := 'N';
  else
    L_upd_resv_ind   := 'N';
    L_upd_intran_ind := 'Y';
  end if;
elsif :ora_tsf_type = 'CO' and :ora_deliver_type = 'S' or
    :os_store_rcv_type = 'A' then
  L_upd_resv_ind   := 'Y';
  L_upd_intran_ind := 'N';
else
  if :os_tsf_status = 'C' then
    L_upd_resv_ind   := 'N';
  else
    L_upd_resv_ind   := 'Y';
  end if;
  L_upd_intran_ind := 'Y';
end if;
```

call TRANSFER_OUT_SQL.EXECUTE package function

(see design specification for TRANSFER_OUT_SQL)

```
if transfer type = ‘CO’ and delivery type = ‘S’ or store receive type is ‘A’
        call TRANSFER_IN_SQL.EXECUTE package function
        (see design specification for TRANSFER_IN_SQL)
```

write_recs_to_struct()

post_transfers()

```
if ( shipment number was not passed in on the input file )
    insert TSFHEAD  (transfer_type = ‘MR’ or PO in an allocation is passed
    in, ext_ref_no = external shipment number)
    insert SHIPMENT  (ext_ref_no_out should be the transaction control
    number, ship date should be the transaction date)
    perform array insert of TSFDETAIL
    perform array insert of SHIPSKU
else (  for all other Retek initiated transfer transactions )
    try to update shipsku record if no data is found
    perform array update of TSFDETAIL, set ship_qty – if transfer type =
    ‘SA’, set tsf_qty = 0
    perform array insert of SHIPSKU
    - The this transfer is a customer order (tsf_type = ‘CO’) with a delivery
    type of direct ship to customer, then this transfer must also be closed
    when it is sent.
    if transfer type = ‘CO’ and delivery type = ‘S’ or store rcv type is ‘A’
        call TRANSFER_IN_SQL.CLOSE
        (see design specification for TRANSFER_IN_SQL)
    else if transfer type = ‘SA’ then
        update TSFHEAD status to ‘A’ - approved
    else
        update TSFHEAD status to ‘S’ - shipped
    end if
end if
```

format_header_fields()

```
assign input file fields to variables

if from location type = ‘ST’
    set ora_from_type = ‘S’
else if from location type = ‘WH’
    set ora_from_type = ‘W’
end if

if to location type = ‘ST’
    set ora_to_type = ‘S’
else if to location type = ‘WH’
    set ora_to_type = ‘W’
end if
```

format_detail_fields()

```
assign input file fields to variables

- transfer quantity has an implied 4 decimal places
transfer qty = transfer qty / 10000
```

process_carton()

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

ON Fatal Error

- rollback to last physical commit point
- Exit Program

ON Non-Fatal Error

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

I/O specification

Input file

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

IMPORTANT:

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

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFO	Identifies file as 'Transfer OUT'
	File Create Date	Date	create date	date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Transaction Date	Date	specified by external system	date the transfer was created in external system
	From Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location sending items

Record Name	Field Name	Field Type	Default Value	Description
	From Location Value	Number(4)	location identifier	Specifies the sending location id number
	To Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location receiving items
	To Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross-reference
	External shipment	Char(15)	External shipment number	specifies external shipment number; will be CARTON when transferring cartons
	Courier	Char (20)	Courier used to ship order	
	Arrival date	Date	Arrival date	
	Number of boxes	Number(4)		Number of boxes in this transfer
	BOL number	Number(13)	Bill of lading	
Transaction Detail (Item)	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPCSKU	item type will be represented as a UPC or SKU
	Item Value	Number(13)	item identifier	the id number of a SKU or UPC
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item
	Allocation Number	Char(6) or char(10) if the allocation_ind is = ‘Y’.	allocation identifier	Retek allocation number attached to the transfer
	Inventory Status	Number(2)	inventory status of item	used to indicate the type of non-salable merchandise transferred in an ‘NS’ transfer
	carton	Char(20)	carton identifier	UCC – 122 carton code

Record Name	Field Name	Field Type	Default Value	Description
	Transfer Quantity	Number(12)		number of units to be transferred of the given item (*10000—4 implied decimal places)
Transaction Detail (Carton)	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	CTN	item type will be represented as a CTN when transferring a carton
	Item Value	Char(20)	carton identifier	UCC – 122 carton code
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Detail Line Count	Number(6)	sum of detail lines	sum of the detail lines within a transaction

Record Name	Field Name	Field Type	Default Value	Description
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Current line number
	Number of transaction lines	Number(10)	specified by external system	total number of lines in file, excluding FHEAD and FTAIL

Output file

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

Record Name	Field Name	Field Type	Default Value	Description
	Ship Date	Number (14)	Ship date	Specifies date shipped (date when file was processed + 1)

Detail

Record Name	Field Name	Field Type	Default Value	Description
	Record Type	Char (1)	D	Specifies record type
	Store Order number	Number (10)	Order No	Specifies Shipment Number
	Division type	Char (2)	SA, PO, MR, CO, AD	Specifies Division Type
	Xref Div Item	Number (8)		RMS SKU
	UPC	Number (13)	UPC value	Specifies UPC Value
	UPC supplement	Number (5)	UPC supplement	Specifies UPC supplement value
	Unit of Measure	Char (2)	Unit of Measure	Specifies unit of measure
	SKU Deck Cost	Number (10)	Deck cost	Average unit cost
	Quantity Shipped	Number (12)	Quantity Shipped	Specifies quantity shipped value

Reject file

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the transfer out module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error file

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

Technical issues

N/A