Retek

Retek Merchandising System 9.0.7

Addendum to Operations Guide

rms-907-og-addendum

Retek Merchandising System™

Copyright Notice

Trademarks

Policy on Retek End User Documentation

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

Printed in the United States of America.

Customer Support

Customer Support hours:

8 AM to 5 PM Central Standard Time (GMT-6), Monday through Friday, excluding Retek company holidays (in 2002: Jan. 1, May 27, July 4, July 5, Sept. 2, Nov. 28, Nov. 29, and Dec. 25).

Customer Support emergency hours:

24 hours a day, 7 days a week.

Contact Method	Contact Information
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: + 1 612-587-5000
Fax	(+1) 612-587-5100
E-mail	support@retek.com
Internet	www.retek.com/support Retek's secure client Web site to update and view issues
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support:

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

Contents

Chapter 1 – Introduction

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

- Advanced Shipment Notification Download [asndnld]
- Inventory Adjustment Purge [invaprg]
- Bill of Lading Upload [lifbolup]
- Purchase Order Purge [ordprg]
- Promotions Purge [prmprg]
- Transfer Shipments Upload [tsfoupld]
- Batch schedule

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

Chapter 2 – Advanced Shipment Notification Download [asndnld]

Modification

Revised due to changes from SIR 32210.

Design overview

The WMS system requires an advanced shipment notification (ASN) file to process receipt transactions. The file is created by this module, asndnld. This module will be run daily, with WMS ASNs being created based on information extracted from Retek shipment/order tables.

The file layout to be used is specified below.

Scheduling constraints

Processing Cycle:	Daily
Scheduling Diagram:	Run after ediupasn.pc and after ordrev.pc
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	Shipment (only allowed 1 thread)

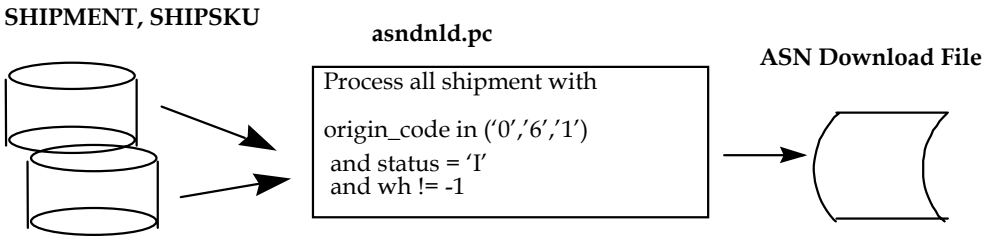
Restart recovery

The Logical Unit of Work will be a shipment.

```
   EXEC SQL DECLARE c_shipments CURSOR FOR
      SELECT sh.shipment,
             sh.order_no,
             TO_CHAR(sh.ship_date, 'YYYYMMDDHH24MISS'),
             sh.location,
             o.supplier,
             o.pre_mark_ind,
             o.cust_order,
             'A'            /* Action Type */
        FROM shipment sh,
             ordhead o
       WHERE sh.order_no = o.order_no
         AND sh.ship_origin  in ('0', '6', '1')
         AND sh.status_code  = 'I'
         AND sh.ext_shipment is not NULL
```

```
      AND sh.loc_type     = 'W'
      AND sh.shipment     > NVL(:os_restart_shipment, -
999)
 UNION ALL
   SELECT sp.shipment,
          sp.order_no,
          TO_CHAR(sp.ship_date, 'YYYYMMDDHH24MISS'),
          sp.location,
          sp.supplier,
          sp.pre_mark_ind,
          sp.cust_order,
          'D'            /* Action Type */
     FROM shipment_prg sp
 ORDER BY 1;
```

Program flow



Shared modules

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

RECEIVE_SQL.DIST_PARTIAL_SHIP(): calculates the distribution quantity

Function level description

init()

The init function will initialize the restart variables. It will also set the up the temporary and final files as well as setup the format string for the records that need to be written to the file. Memory will created for the structure that will hold the values fetched from the table. As well as, calling the function that writes the standard header record to the temporary output file.

process()

The process function will do an array fetch of all the shipments that have an origin code of ASN Shipment ('0'), ASN UCC - 128 ('6') or a Manual Shipment ('1') that are in input status, for a warehouse location. It will also fetch any shipments that need to be deleted from the WMS system due to orders being cancelled. The process function will loop through all the shipments that meet these criteria. For each shipment an Advanced Shipment Notice Header Record will be written to a temporary buffer variable.

If the shipment is a carton shipment, then (ie. carton indicator = 'Y'), set the asn_type = 'C', otherwise default the asn_type = 'P'. The action type will 'A'dd for new shipments and 'D'elete for shipments to be removed from the WMS system. The carrier code and container qty will be defaulted to NULL. The ASN number will be the PO number joined with the shipment number retrieved from the main shipment cursor.

Loop through all of the shipsku or shipsku_prg records associated with the current shipment.

For each item in the shipment the quantity expected (shipsku.qty_expected) and the quantity to cross-dock (min qty_to_allocate, shipsku.qty_expected) will be determined. If the cross-dock quantity is greater than 0, then the cross-dock flag will be set to 'Y', otherwise it will be 'N'

After the cross dock check has been made, three flat files will be written with the following guidelines:

If carton exists, then (creation of ASN container and container/item records)

```
Call process_carton()
```

If it is not a carton shipment, then (creation of ASN item file)

```
Call process_non_carton()
```

Update the shipment.status_code = 'E'xtracted for the shipment.

If no errors occurred while processing the shipment details, the temporary buffer holding the Advanced Shipment Notice Header Record, and all of the Advanced Shipment Notice Detail records will be written out to the temporary file. After processing all the shipments selected in each array fetch the restart_commit and restart_file_write function will be called. The final step in this function is to call the function that writes the Standard Trailer record to the file.

Process_carton()

This function will process all of the carton shipments.

If premark_ind = 'Y', then

Retrieve the location from the carton table for the specified carton value.

If it is a customer order, then set the destination as the warehouse from the shipment table. Otherwise, set the destination as the store or warehouse of the allocation found on the carton table.

Else

**note: non premark cross dock ASNs will be sent to Logistics through container and container/item records, however, there will be no true cross docking. This is due to the fact that only premark orders will have distribution locations for each carton sent from the supplier. Therefore, the non premark carton ASN will be received at a carton level, but must be put into "stock" at the warehouse, before it can be distributed. The receiving process will actually create the transfers for the merchandise.

Set the destination value as the warehouse (ie. shipment.location).

End If;

Call get_unit_retail() to get the unit retail of the item.

Call get_ticket_type() to get the item's ticket type.

If the carton number changes then write TCRTN record for the container by calling write_tcrtn(). The TCRTN should only be written once for every container retrieved.

Write the TCNIT record for each sku in the carton. By calling write_tcnit(). This function will write the ASN container/item record for the file. The destination id equals the value set in the above logic. The sku field will be the shipsku.sku record, the container id will be the shipsku.carton, and the unit qty will be the shipsku.qty value. There could be multiple records written for each container.

Process_non_carton()

This function will process all of the non carton shipments.

If cross dock, then

Loop through all of the allocation detail records for the order on the shipment.

If the shipsku quantity is less than the allocation quantity expected, call receive_sql.dist_partial_ship package. This package will calculate the quantity to distribute. If there is a split shipment (ie. supplier is sending 50 units now and 50 items later), then the quantity will be calculated based upon the allocation method retrieved from the allocation. A running total will be kept to keep track of how much has been allocated.

If the shipsku quantity is greater than the allocation quantity expected, allocate the shipment and calculate the balance of the shipment to be written after the loop ends.

If the shipsku quantity equals the allocation quantity expected, allocate the quantities of the shipment. No balance will need to be calculated.

If it is a customer order, then the destination is set to be the warehouse. Otherwise, the destination will be set based on the allocation location.

Call get_unit_retail()

Call get_ticket_type()

Call write_titem() function. Write TITEM record for the ASN item. A record will be written for each location found on the allocation for the ASN item. Destination id equals the allocation location (ie. store or wh), UNLESS if it is a customer order, then the destination id is the shipment warehouse, and the unit qty equals the quantity passed back from the package.

End looping through the allocation locations.

If there is a balance remaining after allocations, write the TITEM record for the balance by calling write_titem(), using the shipment location as the destination.

Else the order is not cross docked.

Set the destination location equal to the warehouse.

Call write_titem() function. Write TITEM record for the ASN item. Destination id equals the shipment.location (ie. this value should be the same as ordloc.wh). The unit qty will equal the shipsku.expected_qty.

End if;

Write_titem()

This function will write the TITEM records for the ASN file.

Write_tcrtn()

This function will write the TCRTN records for the ASN file.

Write_tcnit()

This function will write the TCNIT records for the ASN file.

Get_ticket_type()

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

Get_unit_retail()

Get Unit retail for the item/location from the item_zone_price tables by calling the pricing_attrib_sql.get_retail package.

write_std_header()

This function will format and write the Standard Header Record to the temporary output file.

write_std_trailer()

This function will format and write the Standard Trailer Record to the temporary output file.

final()

This will call the restart_file_write and restart_close library functions.

I/O specification

Standard Header Record

One per file batch file

Field Name	Field Type	Default Value	Description
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)	ASND	Identifies file as 'ASN header download'
File Create Date	Date	System date	date file was written by external system

ASN Transaction Header file

One record per each ASN

Field Name	Field Type	Default Value	Description
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(14)	specified by external system	Used to force unique transaction check
Location(DC)	Number(4)	Shipment.location	Code for the DC

Field Name	Field Type	Default Value	Description
Action Type	Char(1)	'A' or 'D'	Record code
Transaction Date/Time	YYYYMMDDHHMI	System date	Date/time created on host
ASN Nbr	char(20)	shipment.order_no + shipment.shipment	Unique identifier of the ASN. By combining order number and shipment number will create a unique number to save in Nautilus. PO (8) + shipment (9) => The PO will need to be zero padded to the left to fill the full 8 bytes. The shipment will need to be padded to the left with zeros to fill the full 9 bytes. This is important for receiving upload, when we strip off the shipment number from the ASN number. 3 null pads are added to the far right (after the shipment number).
ASN Type	Char(1)	'C'ontainer level ASN,'P'urchase Order level ASN	If shipsku has carton, then 'C' else 'P' for purchase order asn.

Field Name	Field Type	Default Value	Description
Carrier Code	char (4)optional	NULL	Contains the courier that will deliver the shipment
Container qty	number(6)	NULL	Number of containers in this ASN

ASN Container file

One record per each container level ASN

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	TCRTN	Identifies file record type. Carton record
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Detail sequence number	Number(6)	specified by external system	Used to force unique detail check. Assigned to detail records in transaction.
Location(DC)	Number(4)	Shipment.location	Code for the DC (not sent to Logistics)
Action Type	char(1)	'A' or 'D'	Record Code
Transaction Date/Time	YYYYMMDDHHMI	System date	Date/Time created on host
UCC Container ID	char(20)	Shipsku.carton	The UCC-128 carton number for shipments originating form the ASN process as carton shipments. This field will be zero for all shipments that are not at a carton level
ASN Nbr	VARCHAR(20)	Shipment.order_no + Shipment.shipment	Unique identifier of the ASN
PO Nbr	char(9)	Shipment.order_no prefixed with 'P'	Purchase Order identifier

Field Name	Field Type	Default Value	Description
Dest ID	number(4)	Ordloc.location OR alloc_detail.store/wh	Identifier of the shipping destinationContains the location that the shipment will be delivered to. If not cross-dock ordloc table; else if cross dock then alloc_detail table
Cube	number(6.2)999999.99	NULL	Container Cube (dimensionless)
Weight	number(4.3)9999.999	NULL	Container Weight(dimensionless)
Expedite Flag	char(1)Required ('Y' or 'N')	'N' (hardcode)	Flag indicating whether the order should be shipped via normal or expedited carrier service
Lot_nbr	Char(12)	NULL	Dye lot number.

ASN Container Item file

One record for each item on an ASN container

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	TCNIT	Identifies file record type. container/item record
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Detail sequence number	Number(6)	specified by external system	Used to force unique detail check. Assigned to detail records in transaction.

Field Name	Field Type	Default Value	Description
Location(DC)	Number(4)	Shipment.location	Code for the DC (not sent to Logistics)
Action Type	Char (1)	'A' or 'D'	Record Code
Transaction Date/Time	YYYYMMDDHHMI	System date	Date/Time created on host
UCC Container ID	char(20)	Shipsku.carton	The UCC-128 carton number for shipments originating form the ASN process as carton shipments. This field will be zero for all shipments that are not at a carton level
Item ID	Char(16)	Shipsku.sku	Contains the SKU associated with the shipment
Requested Unit Qty	Number(8)	Shipsku.qty_expected	The number of items expected to be received based on the associated order number of on the supplier's ASN for this SKU/shipment combination

Field Name	Field Type	Default Value	Description
Distribution Number	char(9) or char(11) if the allocation_ind is = 'Y'.	Alloc_header.alloc_no (prefixed with 'A')	Unique identifier for the distribution. If no distribution exists, then NONE will be written.
Ticket Type	Number(4)	Item_ticket.ticket_type_id	Type of ticket refers to ticket type table. The ticket type with printing at the time of receipt indicator.
Unit retail	Number(7, 2)	Item_zone_price.unit_retail	Price of merchandise

ASN Item file

One record for each PO, Destination, or Item combination on an PO level ASN

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	TITEM	Identifies file record type. Item record
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Detail sequence number	Number(6)	specified by external system	Used to force unique detail check. Assigned to detail records in transaction.
Location(DC)	Number(4)	Shipment.location	Code for the DC (not sent to Logistics)

Field Name	Field Type	Default Value	Description
Action Type	Char (1)	'A' or 'D'	Record Code
Transaction Date/Time	YYYYMMDDHHMI	System date	Date/Time created on host
ASN Nbr	char(20)	Shipment.order_no + Shipment.shipment	Contains the unique ASN number identifying a specific shipment of goods within the system.
PO Nbr	char(9)	Shipment.order_no (with 'P' prefix)	Purchase Order identifier
Item ID	Char(16)	Shipsku.sku	Unique identifier of the item
Dest ID	Number(4)	Ordloc.location OR alloc_detail.store/wh	Contains the location that the shipment will be delivered to. If not cross-dock ordloc table; else if cross dock then alloc_detail table
Requested Unit Qty	number(8)	Shipsku.qty_expected	The number of items expected to be received based on the associated order number of on the supplier's ASN for this SKU/shipment combination

Field Name	Field Type	Default Value	Description
NEW FIELDDistribution number	Char(9) or char(11) if the allocation_ind is = 'Y'.	Alloc_header.alloc_no (prefixed with 'A')	Unique identifier for the distribution
NEW FIELDTicket type	Number(4)	Item_ticket.ticket_type_id	Type of ticket refers to ticket type table. The ticket type with printing at the time of receipt indicator.
NEW FIELDUnit retail	Number(7, 2)	Item_zone_price.unit_retail	Price of merchandise

Transaction Trailer

One record per transaction header (that is, THEAD)

Field Name	Field Type	Default Value	Description
File Type Descriptor	Char(5)	TTAIL	Identifies file record type
File Line Sequence	Number(10)	incremented internally	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

One record per file header (that is, FHEAD)

Field Name	Field Type	Default Value	Description
File Type 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

Technical issues

N/A

Chapter 3 – Inventory Adjustment Purge [invaprg]

Modification

Restart recovery was added to the program.

Design overview

The Inventory Adjustment Purge (invaprg) module will delete all obsolete inventory adjustment records where a pre-determined number of months have elapsed. The pre-determined number of months is a system option.

Tables affected

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
INV_ADJ	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
RESTART_CONTROL	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart recovery

N/A

Program flow

N/A

Shared modules

N/A

Function level description

In the init() function, select the number of months inventory adjustments are held is selected from the UNIT_OPTIONS table.

Then, in the process() function, all records are delete from the INV_ADJ table where the number of months between the current date and the adjustment date is greater than the number of months that inventory adjustments are held.

I/O specification

N/A

Technical issues

N/A

Chapter 4 – Bill of Lading Upload [lifbolup]

Modification

The field length for Item Value was changed to Number (13).

Design overview

This program will format information originating from the Nautilus Bill of Lading files. The Nautilus files, BOL Upload Header, BOL item upload, and BOL Container Item Upload are SQL Loaded into staging tables: lif_bol_head, lif_bol_item, and lif_bol_container_item. This program will read from the staging tables and create a standard formatted file for Retek's tsfoupld program to process.

Scheduling constraints

Processing Cycle: N/A

Scheduling Diagram: This program should be run after uploading the bill of lading (BOL) information from Nautilus and SQL Loading the files into the staging tables.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart recovery

Driving cursor

```
Select lbh.dc_location,
           lbh.bol_number,
          NVL(SUBSTR(lbh.carrier,1,20),''),
          NVL(lbh.scheduled_arrival_date_time, ''),
          lbci.container_id,
          SUBSTR(lbci.item_id,1,13),
          lbci.distribution_number,
          lbci.unit_quantity*:pi_implied_dec,
          lbc.destination_id
     FROM lif_bol_head lbh,
          lif_bol_container lbc,
          lif_bol_container_item lbci
    WHERE lbh.bol_number = lbc.bol_number
      AND lbc.bol_number = lbci.bol_number
```

```
        AND lbc.container_id = lbci.container_id
        AND lbh.dc_location = lbc.dc_location
        AND lbc.dc_location = lbci.dc_location
    order by lbh.bol_number, lbci.distribution_number
desc;
```

Descending will allow us to fetch allocations before shipments

The program will select the data from the BOL temp tables and write the records to the RMS header and detail flat files.

In determining which fields in the RMS files must be populated, if the distribution number is prefixed by an 'A' we are uploading a cross dock allocation. The allocation number on the detail file must be populated. If the distribution number is not prefixed with an 'A', we are uploading a transfer initiated in RMS, and the distribution number is the shipment number. The 'A' will be stripped from the distribution number when uploading.

Program flow

N/A

Function level description

Init()

Open output file.

Write FHEAD

Process()

Loop through records found in the driving cursor.

Call get_quantity().

If the BOL number changes from what it was previously:

Write a header record

If the distribution number is prefixed with an 'A':

If the bol number changes, then

Call get_regular_to_loc() to retrieve the allocation location type.

elsif the distribution number is not prefixed with an 'A'

Call ship_direct().

If the shipment is a ship direct, customer order, then

Call get_ship_direct_to_loc().

Else

Call get_regular_to_loc()

End if;

End if;

Write a header record when the bol number changes for the new bol number. Also, write a trailer record for the previous bol number. Call write_Thead() and write_Ttail().

If the record is a shipment, then a header record is written for each new shipment number, regardless of the bol number. For allocations that are processed, only one header record is written.

Call write_tdetl() to write the detail information to the flat file.

The output file will have the following format:

```
FHEAD
THEAD for first BOL
TDETL's for all allocations within first BOL
TTAIL
THEAD for first shipment within first BOL
TDETL's for each detail record within this shipment
TTAIL
<repeat THEAD,TDETL's,TTAIL for each shipment within
first BOL>
<repeat for subsequent BOL's>
```

Within a BOL, all allocations appear as details under one header. In contrast, shipments each have their own header record.

Call Write_FTAIL()

Win_skus_uom()

Fetch UOM from win_skus table.

Rag_style_uom()

Fetch UOM from rag_style table.

Packhead_uom()

Fetch UOM from packhead table.

Get_quantity()

It is assumed that both shipped_weight and unit_quantity fields will be populated from RDM.

Thus, UOM of the item corresponding to this BOL should be determined.

An item is catch_weight when its UOM is anything other than 'EA'. In order to determine the UOM of an item, the following columns need to be used:
RAG_STYLE.STANDARD_UOM for fashion items,
WIN_SKUS.STANDARD_UOM for staple items and
PACKHEAD.STANDARD_UOM for pack items.

(DESC_LOOK.SYSTEM_IND will display 'f' for fashion item, 'S' for staple item and 'P' for pack item.)

If the UOM of the corresponding item is 'EA' then unit_quantity field should be used for output file. Any item with a UOM other than 'EA' is assumed to be a catch weight item, therefore shipped_weight should be used to write to the transfer quantity field.

Write_FHEAD()

Write the FHEAD record to the file.

Write_THEAD()

Write the THEAD record to the file.

Bol number = DC + BOL

If it is a shipment, then the distribution number equals the shipment number. Otherwise, the shipment value is null.

Write_TDETL()

Write the TDETL record to the file.

If it is a shipment, then the allocation value is NULL. Otherwise, populate the allocation value with the distribution number.

Write_TTAIL()

Write the TTAIL record to the file.

Write_FTAIL()

Write the FTAIL record to the file.

Get_regular_to_loc()

Set location type variable equal to 'ST' or 'WH' whether the location is a store or warehouse.

Ship_direct()

Check if the shipment is a ship direct, customer order.

Get_ship_direct_to_loc()

If a record is a shipment, and if the shipment is scheduled for direct delivery to the customer, replace the destination location and loc_type, with the location and loc_type stored on the shipment table (presumably, the location of the selling store).

If location type = 'S', then

Set the location type = 'ST'

If location type = 'W' then

Set the location type = 'WH'

Write_files()

Write bill of lading information to the standard flat file.

Final()

Write FTAIL, TTAIL

Delete from the temp tables: lif_bol_head, lif_bol_container, lif_bol_container_item.

Close output file

I/O specification

Output file

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

Note that there is a space between fields in the RLS flat file format.

Output File (in format needed by RMS tsfoupld.pc program)

Record Name	Field Name	Field Type	Default Value	Description
File Header	File type record descriptor	Char(5)	FHEAD	Identifies the 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

Record Name	Field Name	Field Type	Default Value	Description
	From Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location sending items
	From Location Value	Number(4)	location identifier	Specifies the sending location id number
	To Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location receiving items
	To Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross-reference
	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	
	Scheduled Arrival Date	Date	shipment.ext_arr_date	arrival date
	Number of boxes	Number(4)		Number of boxes in this transfer

Record Name	Field Name	Field Type	Default Value	Description
	BOL Number	Number (13)	BOL Number	Bill of Lading number
Transaction Detail (Item)	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPCSKU	item type will be represented as a UPC or SKU
	Item Value	Number(13)	item identifier	the id number of a SKU or UPC
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item
	Allocation Number	Char(6) or char(10) if the allocation_ind is = 'Y'.	allocation identifier	Retek allocation number attached to the transfer

Record Name	Field Name	Field Type	Default Value	Description
	Inventory Status	Number(2)	inventory status of item	used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	Carton Number	Varchar(20)	Container ID	Container identifier
	Transfer Quantity	Number(12)		number of units to be transferred of the given item (*10000—4 implied decimal places)
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Detail Line Count	Number(6)	sum of detail lines	sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Current line number
	Number of transaction lines	Number(10)	specified by external system	total number of lines in file, excluding FHEAD and FTAIL

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 – Purchase Order Purge [ordprg]

Modification

Revised due to changes from SIR #33676 and SIR #32210.

Design overview

The purpose of this module is to remove old orders from the system.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'N' and if invoice matching is not installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). Orders are deleted only if shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'Y' and if invoice matching is not installed, then all details associated with the order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months) , as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status), and as long as all shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

This program will also create a PO header flat file to interface with the Nautilus system. When orders are deleted, a record with the action type = ‘D’eleted will be written to an output file. Nautilus will then process this file and delete the PO from the warehouse’s database to maintain consistency between the host and warehouse environment.

The program will also populate the SHIPMENT_PRG, SHIPSKU_PRG and CARTON_PRG tables, which are then used by the asndnld program to notify the Nautilus system of ASN shipments that need to be deleted.

Tables affected

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALC_COMP_LOC	No	No	No	No	Yes
ALC_HEAD	No	Yes	No	No	Yes
ALLOC_DETAIL	No	No	No	No	Yes
ALLOC_HEADER	Yes	No	No	No	Yes
CARTON	No	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
CARTON_PRG	No	No	Yes	No	No
CE_LIC_VISA	No	No	No	No	Yes
CE_CHARGES	No	No	No	No	Yes
CE_ORD_ITEM	No	Yes	No	No	Yes
CE_SHIPMENT	No	No	No	No	Yes
CE_HEAD	No	No	No	No	Yes
INVC_DETAIL	Yes	No	No	No	Yes
INVC_HEAD	Yes	Yes	No	No	Yes
INVC_MATCH_WKSHT	Yes	No	No	No	Yes
INVC_MATCH_QUEUE	Yes	No	No	No	Yes
INVC_MERCH_VAT	Yes	No	No	No	Yes
INVC_NON_MERCH	Yes	No	No	No	Yes
INVC_XREF	Yes	No	No	No	Yes
LC_ORDAPPLY	No	No	No	No	Yes
OBLIGATION	No	No	No	No	Yes
OBLIGATION_COMP	No	No	No	No	Yes
OBLIGATION_COMP_LOC	No	No	No	No	Yes
ORDCUST	No	No	No	No	Yes
ORDHEAD_DISCOUNT	No	No	No	No	Yes
ORDHEAD	Yes	Yes	No	No	Yes
ORDLOC	Yes	No	No	No	Yes
ORDSKU_INVC_COST	No	No	No	No	Yes
ORDSKU	Yes	No	No	No	Yes
LC_ORDAPPLY	No	No	No	No	Yes
DEAL_HEAD	No	No	No	No	Yes
DEAL_DETAIL	No	No	No	No	Yes
DEAL_ITEMLOC	No	No	No	No	Yes
DEAL_THRESHOLD	No	No	No	No	Yes
DEAL_QUEUE	No	No	No	No	Yes
DEAL_CALC_QUEUE	No	No	No	No	Yes
ORD_INV_MGMT	No	No	No	No	Yes
REPL_RESULTS	No	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
QC	Yes	No	No	No	Yes
RTV_DETAIL	No	No	No	No	Yes
SHIPMENT	Yes	Yes	No	No	Yes
SHIPMENT_PRG	No	No	Yes	No	No
SHIPSKU	Yes	No	No	No	Yes
SHIPSKU_PRG	No	No	Yes	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
TRANS_CLAIMS	No	No	No	No	Yes
TRANS_DELIVERY	No	No	No	No	Yes
TRANS_LIC_VISA	No	No	No	No	Yes
TRANS_PACKING	No	No	No	No	Yes
TRANS_SKU	No	No	No	No	Yes
TRANSPORTATION	No	Yes	No	No	Yes
TSFDETAIL	Yes	No	No	No	Yes
TSFHEAD	Yes	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart recovery

Driving cursor

When Retek's Invoice Matching product is not in use:

```
SELECT oh.order_no,
       lc.lc_ind,
       a.status
  FROM ordhead oh,
       ordlc lc,
```

```
       alc_head a
 WHERE lc.order_no(+) = oh.order_no
   AND a.order_no(+) = oh.order_no
   AND ((0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
             oh.close_date),0) - :oi_hist_months))
        or (oh.status = 'W'
       AND oh.orig_ind = 0
       AND oh.contract_no is NULL
       AND (TO_DATE(:os_vdate,'YYYYMMDD') -
oh.written_date) >= :oi_repl_order_history_days));
```

When Retek's Invoice Matching product is in use:

```
SELECT distinct oh.order_no,
          lc.lc_ind,
          a.status
     FROM ordhead oh,
          shipment sh,
          shipsku ss,
          invc_head ih,
          ordlc lc,
          alc_head a
    WHERE oh.order_no = sh.order_no
      AND lc.order_no(+) = oh.order_no
      AND a.order_no(+) = oh.order_no
      AND sh.shipment = ss.shipment
      AND ss.match_invc_id = ih.invc_id(+)
      AND (0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
                oh.close_date),0) - :oi_hist_months))
      AND ((0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
                   sh.invc_match_date),0) -
:oi_hist_months))
           OR (sh.invc_match_date IS NULL))
      AND (ss.match_invc_id is null OR ih.status = 'P')
      AND sh.invc_match_status = 'C'
      AND 'C' = (SELECT
decode(max(ship.invc_match_status),
min(ship.invc_match_status),'C','X')
                  FROM shipment ship
```

```
                  where ship.order_no = oh.order_no
               group by ship.order_no)
    UNION
    SELECT oh.order_no,
           lc.lc_ind,
           a.status
      FROM ordhead oh,
           ordlc lc,
           alc_head a
     WHERE lc.order_no(+) = oh.order_no
       AND a.order_no(+) = oh.order_no
       AND oh.status = 'W'
       AND oh.orig_ind = 0
       AND oh.contract_no is NULL
       AND (TO_DATE(:os_vdate,'YYYYMMDD') -
oh.written_date) >= :oi_repl_order_history_days;
```

Program flow

N/A

Shared modules

INV_SQL.DELETE_INVC

Function level description

Delete from the appropriate ordering table and any table that may have referential integrity constraints for the fetched order number. Populate the _prg tables when ASN orders and shipments are to be deleted. Fetch order numbers appropriately based on whether or not invoice matching is installed.

Init()

Select the following fields values:

- invc_match_ind, import_ind, repl_order_history_days, edi_rev_days, rws_ind from the system_options table
- order_history_months from the unit_options table
- vdate from period table

Init_logistics()

Initialize the format of the output file.

Process()

Call del_rev to delete order revision

Open the particular driving cursor based on the indicator inv_match_ind

For each order:

Fetch the particular driving cursor based on inv_match_ind

If ordlc.lc_ind = 'Y' then skip and fetch next record;

If import_ind = 'Y' and alc_head.status ='PR' then

Call delete_landed_costs;

End if;

Delete from related tables associated with orders being purged by.

Calling delete_invc_data, delete_deals and delete_repl_orders

Delete from ordhead table.

Delete_invc_data()

Call INVC_SQL.DELETE_INVC function to delete the invoice data for the specific orders being purged.

Del_rev()

Delete records from the tables ordloc_rev, ordsku_rev, ordhead_rev and alloc_rev associated with the orders which have been closed for more days than specified in edi_rev_days(in table UNIT_OPTIONS). But deleting occurs only:

- when a letter of credit is not present(ordlc.lc_ind='N').
- Import indicator equals 'N'. Or
- import indicator equals 'Y', and the landed costs are completed (alc_head.status = 'PR'). In this case, purge these landed costs before deleteing the above tables.

Also, before deleting from these tables, purge all related transportation and custom entries.

Purge_transport()

Delete from the table transportation for specific orders being purged as well as child records from tables missing_doc, trans_packing, trans_delivery, trans_claims and trans_lic_visa(based on transportation_id).

Purge_customs_entry()

Delete from customs entry for specific orders being purged.

Delete_landed_costs()

Delete landed costs and obligations as well as their child records for specific orders being purged. Involved tables include: alc_head, alc_comp_loc, obligation, obligation_comp, obligation_comp_loc.

Write_logistic_details()

Write the deleted order to the output file with action type = 'D'eleted.

Delete_deals()

Delete all PO-specific deals assigned to the order being purged. PO-specific deals are identified by the existence of a value in deal_head.order_no.

Del_repl_orders ()

Delete records from the table ord_inv_mgmt and repl_results associated with the order being purged.

Final()

Close the output file.

I/O specification

Output file

The format will be as in the table given below. Each order is a separate transaction; multiple orders can be given in each file. The skeleton of the file is:

FHEAD (file identification – only one line per file) REQUIRED

FDETL detail lines – one line for every record on ordhead. NOT REQUIRED.

FTAIL (total number lines) One line per file. REQUIRED.

Record	Variable name	Field type	Default value	Description
FHEAD	File record descriptor	Char (5)	FHEAD	identifies file record type
	Line number	Number (10)		identifies file line number
	File type	Char (4)	POHD	identifies file type
	Create date	Char (14)	vdate	YYYYMMDDHH24MISS format
FDETL	File record descriptor	Char (5)	FHEAD	identifies file record type
	Line number	Number (10)		identifies file line number
	Action type	Char (1)	D	identifies the action type "D"elete
	Location	Char (4)		ordloc.location number

Record	Variable name	Field type	Default value	Description
	File create date	Char (12)	vdate	YYYYMMDDHH24MI format
	Order number	Char (8)		ordhead.order_no
	Vender	Char (7)		
	Preassigned flag	Char (1)		
	Dev not before	Char (8)		
	Dev not after	Char (8)		
	Shipping terms	Char (3)		
	Buyer code	Char (12)		
FTAIL	File record descriptor	Char (5)	FTAIL	identifies file record type
	Line number	Number (10)		identifies file line number
	Transaction line number	Number (6)		identifies transaction's line number

Technical issues

N/A

Chapter 6 – Promotions Purge [prmprg]

Modification

Restart recovery was added to this program.

Design overview

All old promotions are purged when the number of months since the end_date of the promotion is greater than prom_history_months from UNIT_OPTIONS. Also, any promotion that is in deleted 'D' or canceled 'C' status will be purged from the system. Rejected promotions that remain in the system beyond the threshold number of promotion reject days will also be purged.

The following database tables are accessed by this program:

TABLE	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No
PROMHEAD	No	No	No	Yes
PROMSTORE	No	No	No	Yes
PROMSKU	No	No	No	Yes
PROM_THRESHOLD_HEAD	No	No	No	Yes
PROM_THRESHOLD_DEPT	No	No	No	Yes
PROM_THRESHOLD_SKU	No	No	No	Yes
PROM_MIX_MATCH_HEAD	No	No	No	Yes
PROM_MIX_MATCH_BUY	No	No	No	Yes
PROM_MIX_MATCH_GET	No	No	No	Yes
RESTART_CONTROL	Yes	No	No	No
SA_TRAN_DISC_REV	No	No	No	Yes
SA_TRAN_DISC	No	No	No	Yes
FIXED_DEAL	No	No	No	Yes
FIXED_DEAL_DATES	No	No	No	Yes
PROMSKU_IMPACT	No	No	No	Yes
PROMDEPT_IMPACT	No	No	No	Yes

Scheduling constraints

Processing Cycle:	PHASE 4 (monthly)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart recovery

Driving cursor

```
   SELECT p.promotion,
          NVL(orh.order_no, -1)
     FROM ordhead orh,
          promhead p
    WHERE orh.promotion (+) = p.promotion
      AND ( (:pi_prom_history_months <
(MONTHS_BETWEEN(to_date(:ps_vdate,'YYYYMMDD'),p.end_date
)) AND p.status = 'M')
            OR p.status in ('C','D')
            OR ((to_date(:ps_vdate, 'YYYYMMDD') -
p.start_date) >= :pi_prom_reject_days) AND p.status =
'R')
order by 1;
```

Program flow

N/A

Shared modules

N/A

Technical issues

N/A

Chapter 7 – Transfer Shipments Upload [tsfoupld]

Modification

Descriptions of a few fields in the output file were modified.

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

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
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

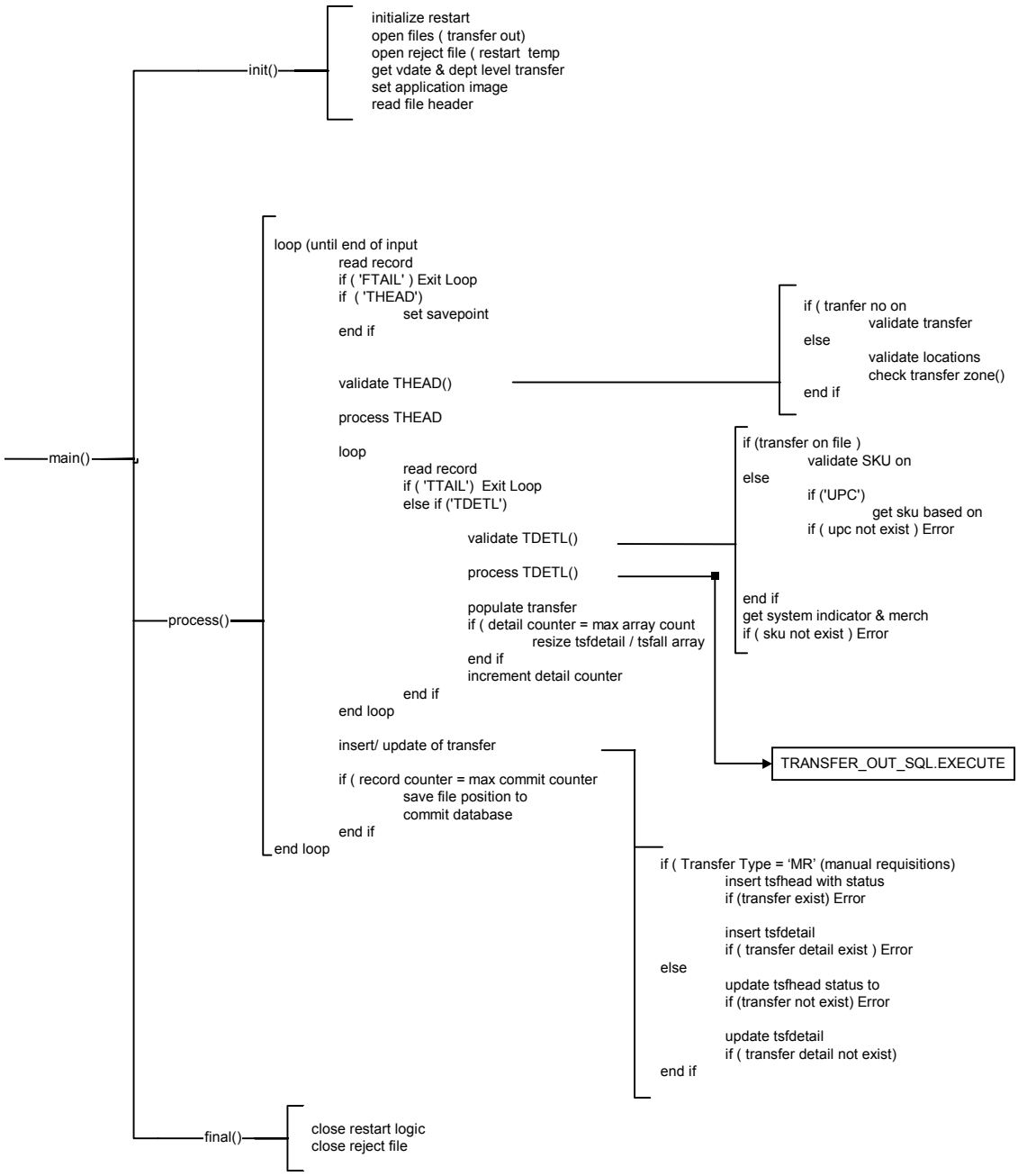
The logical unit of work for the transfer out module is the discrete transfer transaction. Each transfer will be identified by the transfer number (if it already exists in the Retek system) or a unique transaction set number generated by the external system. This transfer transaction will be defined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

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

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

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

Program flow



Shared modules

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

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

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

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

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

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

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

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

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

Function level description

init()

declare structure arrays for tsfdetail

initialize restart recovery

open input file (transfer out)

- file should be specified as input parameter to program

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

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

set application image array - save the line counter

read file header record

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

process()

loop

read record from input file

if ('FTAIL')

Exit Loop

end if

if ('THEAD')

set savepoint and save current file pointer position

validate_THEAD()

reset detail count

process_THEAD()

end if

loop

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

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

if ('TTAIL') Exit Loop

if ('TDETL')

validate_TDETL()

process_TDETL()

end if

if (detail count = max array count)

resize array structures for tsfdetail

increase max array count

end if

increment detail count

end loop

if (no errors)

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

end if

if (non Fatal Error Encountered)

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

Rollback transaction

end if

if (transaction count > max commit count)

restart file commit

- save the current input file pointer position

- save the line counter in restart image

end if

end loop

restart commit final

validate_THEAD()

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

format_header_fields()

if (shipment number provided in transaction)

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

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

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

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

raise Non-Fatal Error

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

raise Non-fatal error

else if (no shipment number is provided)

if (external shipment number provided)

- validate to and from locations

if (loc_type = 'S')

check for existence on store table

else (loc_type = 'W')

check for existence on wh table

end if

if any location not exist, write non-Fatal error

- validate common transfer zone for store to store transfers

if (to_loc type = 'S' and from_loc = 'S')

check transfer zone - select transfer zone of the from location and the to location.

if (from_loc transfer zone <> to _loc transfer zone)

write non-Fatal Error (transfer zones incompatible)

end if

end if

else (no external shipment number)

All detail records must have an allocation number.

end if

end if

process_THEAD()

check for a bill of lading in 0 - open status for the destination location

retrieve the bill of lading number if one exists

if (bill of lading does not exist)

get next bill of lading number

insert bill of lading header (lad_head) record

end if

if bol number passed in ensure it is valid.

If it is not valid get next bol number.

if transfer type = 'CO'

retrieve delivery type from the ORDCUST table

end if

validate_TDETL()

format_detail_fields()

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

if no shipment / ext shipment in file

    every detail line must have an allocation.

if (shipment number in file)

    validate item exists on the transfer

else

    if (Item Type = 'UPC')

        select sku from upc_ean based on the upc and supplement

        if (upc does not exist)

            write non-Fatal Error (upc not found)

        end if

    else if (Item Type = 'SKU')

        SKU = item value from the input file

        case ID = ' '

    end if

end if

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

- get item system indicator, department, class and subclass

if (system indicator does not exist)

    write non-Fatal Error (sku not found)

end if

process_TDETL()

The upd_resv_ind and the upd_intran_ind should be 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

Record Name	Field Name	Field Type	Default Value	Description
	From Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location sending items
	From Location Value	Number(4)	location identifier	Specifies the sending location id number
	To Location Type	Char(2)	ST - storeWH - warehouse	specifies the type of location receiving items
	To Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross-reference
	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(15)		Number of boxes in this transfer
	BOL number	Number(13)	Bill of lading	

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

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

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

Output file

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

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 (6)	Quantity Shipped	Specifies quantity shipped value

Reject file

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

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

Error file

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

Technical issues

N/A

Chapter 8 – Batch schedule

Modification

The dealcalc post-dependency was removed.

Phase 0

aristop (ARI) ** aricntrl (ARI) ** aristart (ARI) **
r-r script
dlyprg

salins
cntrmain
vatdlxpl

Phase 1

ediupavl
ediupasn
pccdnld | pccrdnld
prmxpld | prmext
rcvext
stkvar
ediupinv
pcdnld

supdnld
locdnld
itemdnld

*****Sales Audit--see below*******

stkupld
pre | fifpldp (FIF) | fifcuru2 (FIF)
fiftrmu2 (FIF)
fifvndu2(FIF)

***Note: prepost pre batch cycle should be run before the batch cycle starts to turn off security, and prepost post batch cycle should be run after the entire batch cycle is finished to turn security back on

Phase 2

posupld
lifrtvup (LIF) | rtvupld
lifinvup (LIF) | invaupld

ediupack
promdnld

lifrcvup (LIF) | tsfparse | rcvupld | ctniupld
lifbolup (LIF) | tsfoupld | tsfiupld | invmatch (IM)
invcpost (IM) | fifinvcu(FIF)
invclshp (IM)
cednld | rdmeupld | rdmuitsf
(lcmt730) | lcupld
(lcmt798) | lcup798

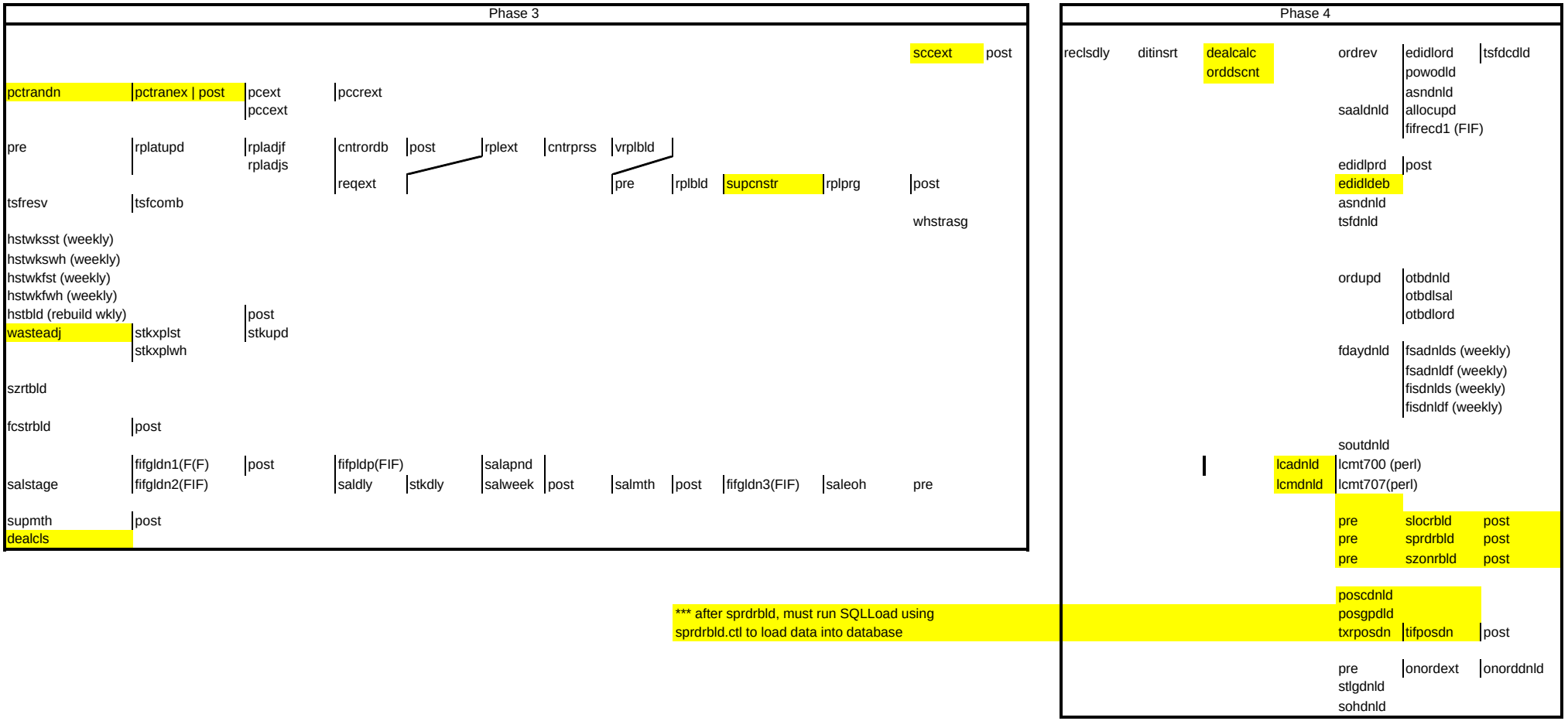
fdayupld

Ad Hoc Interfaces

posdnld | post
plncupld
plndupld
plnsupld
edidlcon
tcktdnld
ediupcat
ediupadd
fmednlds
fmednldf
forgdnld
otbupfwd
otbupld
edidladd
tranupld (RTM)
fifcoadn (FIF)
fwhdnld
pre htsupld
gcupld
txrtupld
ftmednld
stlgdnld *

* Ad-hoc running of stlgdnld is meant for historic downloads. See phase 4 for weekly stlgdnld runs.

** Note that the ARI programs (aristart, aricntrl, aristop) must be run with no other resources accessing the system. They can be run before or after the rest of the batch schedule.


Phase 3
sccext
post
pctrandn
pctranex | post
pcext
pccext
pccrext
pre
rplatupd
rpladjf
rpladjs
cntrordb
post
rplext
cntrprss
vrplbld
reqext
pre
rplbld
supcnstr
rplprg
post
tsfresv
tsfcomb
whstrasg
hstwksst (weekly)
hstwkswh (weekly)
hstwkfst (weekly)
hstwkfwh (weekly)
hstbld (rebuild wkly)
post
wasteadj
stkxplst
stkxplwh
stkupd
szrtbld
fcstrbld
post
fifgldn1(F(F)
fifgldn2(FIF)
post
fifpldp(FIF)
saldly
stkdly
salapnd
salweek
post
salmth
post
fifgldn3(FIF)
saleoh
pre
salstage
supmth
post
dealcls
*** after sprdrbld, must run SQLLoad using
sprdrbld.ctl to load data into database
Phase 4
reclsdly
ditinsrt
dealcalc
orddscnt
ordrev
edidlord
tsfdcdld
powodld
asndnld
saaldnld
allocupd
fifrecd1 (FIF)
edidlprd
post
edidldeb
asndnld
tsfdnld
ordupd
otbdnld
otbdlsal
otbdlord
fdaydnld
fsadnlds (weekly)
fsadnldf (weekly)
fisdnlds (weekly)
fisdnldf (weekly)
soutdnld
lcadnld
lcmt700 (perl)
lcmdnld
lcmt707(perl)
pre
slocrbld
post
pre
sprdrbld
post
pre
szonrbld
post
poscdnld
posgpdld
txrposdn
tifposdn
post
pre
onordext
onorddnld
stlgdnld
sohdnld

Date Set	
(sastdycr)	dtesys

Ad Hoc		
pcimpc		
hstbld (rebuild all)		post
pre	pcovrl	
auditprg		
auditsys		
ccprg		
ediprg		
fcstprg	fcslupld	
hstprg		
invaprg		
ladprg		
layprg		
ordprg	invprg	
otbprg		
pccprg		
pcovrlpq		
pcprg		
prmprg		
rplrsprg		
rtvprg		
salprg		
schedprg		
stkprg		
storeadd	lclrbld	
szrtbld		
tsfprg	tsfalprg	
		lifstkup (LIF)
cmpprg		
dealprg		

Sales Audit											
sagetref	saimptlog	(sqlldr)	savouch****	saimptlogfin	saimpadj*	satotals	sarules	(Forms Auditing)	sapreexp	saexprms***	sapurge
										saexpim***	
										saexprdw***	
										saexpach***	
										saexpuar***	
										saexpgl***	
samastersfm**	saexpsfm**										
				saescheat (monthly)****							

* Only if there are total adjustments from external systems
** Only if Oracle Site Fuel Management is used
*** Only if the external system is used
**** Only if vouchers are being tracked

Forms Auditing is used to correct any errors found during the loading of the data, totaling and rules checking.