Retek Merchandising System 9.0.3.0

Addendum to Operations Guide

rms-9030-og-addendum

Retek Merchandising System™

Customer Support

Customer Support hours:

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

Customer Support emergency hours:

24 hours a day, 7 days a week.

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

When contacting Customer Support:

- Always fill out an Issue Report Form before submitting issues to Retek (request forms from Customer Support if necessary).
- Provide a completely updated Customer Profile.
- Have a single resource per product responsible for coordination and screening of Retek issues.
- Respond to our requests for additional information in a timely manner.
- Use the Expert Web 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) 9.0.0.0 Operations Guide contains updates to the following information:

- Deals – cost calculations (dealcalc.pc) batch module design
- Product security rebuild (sprdrbld.pc) batch module design
- Approved warehouse transfers download (tsfdnld.pc) batch module design
- Stock position download (sohdnld.pc) batch module design
- RMS batch schedule spreadsheets for Deals (SIR number 35190) and System (SIR number 34728)

Refer to the following chapters for that information, which supercedes all comparable information in the RMS 9.0.0.0 Operations Guide. Each chapter contains a subsection indicating what specific modifications have been made.

Chapter 2 – Deals – cost calculations (dealcalc)

Modification

The driving cursor description was updated; a description was added of the new logic in calculate_cost_driver, which was added to make the data in deal_sku_cost more accurate.

Design overview

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

Scheduling constraints

This section contains a pre/post logic description:

Processing Cycle: Phase II (daily)

Scheduling Diagram: Must be run after ditinsrt.pc, which populates the deal_sku_temp table

Pre-Processing:

Post-Processing: Call prepost to delete all records from deal_sku_temp.

Threading Scheme: SUPPLIER

Restart recovery

This section contains information on the Logical unit of work and the driving cursor.

Logical unit of work

The logical unit of work is: SKU/supplier/origin country/start date.

Driving cursor

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

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

```
WHERE      dd.deal_id  = dh.deal_id

AND di.deal_id = dd.deal_id

AND di.deal_detail_id = dd.deal_detail_id

AND dh.status = ‘A’

AND dh.type in (‘A’,’P’)  /* only use promotional/annual, not
PO specific or vendor funded */

AND di.excl_ind  = ‘N’

AND sups.supplier = dst.supplier

AND isc.item = dst.sku

AND isc.supplier = dst.supplier

AND isc.origin_country_id = dst.origin_country_id

AND ((dh.close_date is NOT NULL

      AND dst.start_date BETWEEN DECODE(rebate_ind, 'Y',
NVL(dd.rebate_active_date, dh.active_date), dh.active_date)

                             AND dh.close_date)

     OR (dh.close_date is NULL

         AND dst.start_date >= DECODE(rebate_ind, 'Y',
NVL(dd.rebate_active_date, dh.active_date), dh.active_date)))

AND ((dh.supplier is NOT NULL AND dst.supplier = dh.supplier)
   /* supplier hierarchy match */

           OR(dh.partner_type = ‘S1’ AND isc.supp_hier_lvl_1 =
dh.partner_id)

           OR(dh.partner_type = ‘S2’ AND isc.supp_hier_lvl_2 =
dh.partner_id)

           OR(dh.partner_type = ‘S3’ AND isc.supp_hier_lvl_3 =
dh.partner_id) )

AND ( (di.merch_level = 1)

            OR (di.merch_level = 2 AND di.division =
dst.division

            OR (di.merch_level = 3 AND di.group_no  =
dst.group_no)

            OR (di.merch_level = 4 AND di.dept = dst.dept )

            OR (di.merch_level = 5 AND (di.dept = dst.dept AND
di.class = dst.class))

            OR (di.merch_level = 6 AND (di.dept = dst.dept AND
di.class = dst.class AND di.subclass = dst.subclass))

            OR (di.merch_level = 7 AND di.style = dst.style)
   --style/color hierarchy

            OR (di.merch_level = 8 AND (di.style = dst.style
AND di.color = dst.color)

            OR (di.merch_level = 9 AND (di.style = dst.style
AND ((di.size1 = dst.size1 OR di.size1 is NULL)
AND (di.size2 = dst.size2 OR di.size2 is NULL)))

            OR (di.merch_level = 10 AND di.sku = dst.sku))

AND  (di.org_level is NULL AND dst.chain is NULL
              AND dst.area is NULL AND dst.region is NULL
              AND dst.district is NULL AND dst.location is NULL
       OR (di.org_level = 1 AND di.chain = dst.chain)
       OR (di.org_level = 2 AND di.area = dst.area)
       OR (di.org_level = 3 AND di.region = dst.region)
       OR (di.org_level = 4 AND di.district = dst.district)
       OR (di.org_level = 5 AND di.location = dst.location))
AND (di.country_id = dst.country_id OR di.country_id is NULL)
/* exclude clause here –don’t fetch excluded skus */
AND (NOT EXISTS
   SELECT ‘x’
     FROM deal_itemloc di1
   WHERE di1.deal_id  = di.deal_id
    AND di1.deal_detail_id = di.deal_detail_id
                  AND di1.excl_ind  = ‘Y’
AND ( (di1.merch_level = 1)
                       OR (di1.merch_level = 2 AND
di1.division = dst.division
                       OR (di1.merch_level = 3 AND
di1.group_no  = dst.group_no)
                       OR (di1.merch_level = 4 AND di1.dept =
dst.dept )
                       OR (di1.merch_level = 5 AND (di1.dept =
dst.dept AND di1.class = dst.class))
OR (di1.merch_level = 6 AND (di1.dept = dst.dept AND di1.class
= dst.class
                                         AND di1.subclass =
dst.subclass))
            OR (di1.merch_level = 7 AND di1.style = dst.style)
   --style/color hierarchy
            OR (di1.merch_level = 8 AND (di1.style = dst.style
AND di1.color = dst.color)
            OR (di1.merch_level = 9 AND (di1.style = dst.style
                                                 AND
((di1.size1 = dst.size1 OR di1.size1 is NULL)
                                                 AND
(di1.size2 = dst.size2 OR di1.size2 is NULL)))
            OR (di1.merch_level = 10 AND di1.sku = dst.sku))
   AND  (di1.org_level is NULL AND dil.chain is NULL
              AND di1.area is NULL AND di1.region is NULL
              AND di1.district is NULL AND di1.location is NULL
     OR (di1.org_level = 1 AND di1.chain = dst.chain)
     OR (di1.org_level = 2 AND di1.area = dst.area)
     OR (di1.org_level = 3 AND di1.region = dst.region)
     OR (di1.org_level = 4 AND di1.district = dst.district)
     OR (di1.org_level = 5 AND di1.location = dst.location))
AND (di1.origin_country_id = dst.origin_country_id OR
di1.origin_country_id is NULL))
AND (dst.sku > NVL(:ps_restart_sku, -999) OR  /* restart on
item/supplier/country/start_date */
              (dst.sku = :ps_restart_sku AND
                 (dst.supplier > :ps_restart_supplier OR
         (dst.supplier = :ps_restart_supplier AND
                       (dst.origin_country_id >
:ps_restart_country  OR
            (dst.orign_country_id = :ps_restart_country AND
   dst.start_date > :ps_restart_date)))))
AND vrs.num_threads  = :pi_num_threads
AND vrs.thread_val  = :pi_thread_val
AND vrs.driver_value = dst.supplier
ORDER BY  dst.sku,
dst.supplier,
dst.origin_country_id,
dst.start_date,
dh.close_date,
loc,
cost_appl_num,
dh.type,
dh.create_date,
dd.application_order
```

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

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

2 Create date desc, annual before promotional

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

4 Create date desc, promotional before annual

Program flow

This following structure chart indicates the tables used:

Table	Select	Insert	Update	Delete
period	X			
system_options	X			
deal_sku_temp	X			X
deal_head	X			
deal_detail	X			
deal_itemloc	X			
deal_threshold	X			
deal_sku_cost		X		
item_supp_country	X			
sups	X			

Shared modules

This section lists all externally referenced functions and stored procedures, with a description of the usage.

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

Function level description

This section contains information on all database interactions that are required, and error handling considerations.

init:

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

process:

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

prepare_driving_cursor:

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

calculate_cost_driver:

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

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

1 For each new LUW + loc, reset the flag for 'F'ixed Amt value type discount. 'F'ixed Amt value type discount should only be applied once for each LUW + loc.

2 For each new LUW, reset the flag and merchandise level for 'EX'clusive deal class discount; for each LUW + loc, reset the merchandise/organization level for 'EX'clusive deal class discount (merchandise level needed to be reset back to before any loc-specific applied). 'EX'clusive deal class discount should only be applied once for each LUW + loc.

3 Reset the net/net net/dead net net costs according to the following rules:

   a If new LUW, set to supplier’s original unit cost

   b If the same LUW, check if location changed:

   - If new loc:
     - ► Check if just change from loc-independent to loc-specific. If yes, save net/net net/dead net net costs and the applied merch level (for ‘EX’clusive discount) of loc-independent discounts
     - ► Check if the flag for ‘EX’clusive deal class discount is set (previous ‘EX’clusive discount applied)
       - □ If NO previous ‘EX’clusive discount applied, check if this is an ‘EX’clusive discount:
         - If yes, set net/net net/dead net net costs to base cost (supplier’s unit cost)
         - If no, set net/net net/dead net net costs to costs of loc-independent discounts
       - □ If previous ‘EX’clusive discount applied check if this is an ‘EX’clusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level ‘EX’ discount):
         - If yes, set net/net net/dead net net costs to base cost (supplier’s unit cost)
         - If no, skip this discount.
   - If the same loc, check if the flag for ‘EX’clusive deal class discount is set (previous ‘EX’clusive discount applied)
     - ► If NO previous ‘EX’clusive discount applied, check if this is an ‘EX’clusive discount:
       - □ If yes, set net/net net/dead net net costs to base cost (supplier’s unit cost)
       - □ If no, set net/net net/dead net net costs to latest calculated costs
     - ► If previous ‘EX’clusive discount applied check if this is an ‘EX’clusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level ‘EX’ discount):
       - □ If yes, set net/net net/dead net net costs to base cost (supplier’s unit cost)
       - □ If no, skip this discount.

4 Call calculate_costs to calculate net/net net/dead net net costs. For the same LUW + loc, the driving cursor has sorted the discounts by cost_appl_ind: ‘N’ first, ‘NN’ later, ‘DNN’ last. For each cost application level, the same business rules are followed.

5 If the new LUW is not in the array, increment the writing index of the cost array (we always write a record into the cost array to keep track of last calculated costs, but change to a new record only if the LUW is changed)

6 Prepare an insert record into the deal_sku_cost table by writing costs into the current indexed record of the cost array. There are two dates to consider, start and ending (close_date from deal_head). When inserting the start_date as the active_date, set a flag in the array so we know that's which date it is, and insert the unit_cost from item_supp_country as the base_cost. The location and location type fields should be left NULL if no location was given on deal_sku_temp. Vdate should be used for the calc_date.

7 If the start_date is found in the array, calculate the change for each cost field and subtract that change from the net fields in the array. If there is no close date, subtract the change amounts from the net fields of each close date in the array. If we have a close_date and the date found originally in the array was a start_date, subtract the change amounts from the corresponding close_date entry in the array. Find the close_date by looking for the same LUW with the date indicator set to close_date.

8 After updating with the start_date, add one to the close date see if that reset_date is already in the array. If not, add it to the array setting the net costs to the base_cost.

9 If the reset date is found in the array, set the net costs to the base cost and exit.

10 Save current processed LUW and loc.

calculate_costs:

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

Outputs: calculated net/net net/dead net net costs

The definitions of different net costs are:

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

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

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

Business rules that need to be followed when applying discounts:

- Deal classes:
    - If an exclusive deal was previously found for this SKU/supplier/origin country/start date: new cost should be calculated only if THIS deal is also exclusive and is for a lower merchandise hierarchy. If this is the first exclusive deal, process it and set a flag, saving the hierarchy levels.
    - Cumulative discounts need to be applied to the original unit cost (2% off + 3% off = 5 %off original unit cost)
    - Cascade discounts need to be applied on the result thus far ("current cost")---take2% off of the unit cost, then take 3% off of that price, for example
- Deal value types (take N cost calculation for example):
    - for a % discount

      ```
      If 'CS'cade:
      discount cost = unit cost – (unit cost *%/100)
      If 'CU'mulative:
      discount cost = unit cost – (initial unit cost *%/100)
      ```
    - for an amt discount (first convert amount to be in supplier currency if necessary)

      ```
      discount cost = unit cost – amt   (amount discounts are per
      unit cost already)
      ```
    - fixed amt: if have fixed amount discount must start with THAT amount rather than the unit cost (convert to supplier currency if necessary)

      ```
      discount cost = fixed amt (converted to supplier's currency
      if necessary)
      ```
    - quantity discount ("buy some get some at discount") (these are not allowed on rebates)

      These are the most complicated. They affect the cost of the get item AND of the buy item, whose cost we also need to get. Both the get item and the buy item will be on deal_itemloc. You should only calculate the cost for whichever item you're presently on (if buy item, just calculate buy item cost; will get the free item separately later, or vice versa). The initial unit cost for the get item should be taken from deal_detail.qty_thresh_free_item_unit_cost (or, if that field is not populated, off of item_supp_country). Before any calculations are done, convert the unit costs into supplier currency if necessary. If a buy/get free discount is encountered, the following things need to happen:

        - Call get_unit_cost to get the original unit cost for the buy item (from item_supp_country), if it's different from the free item. Use the supplier and origin country of the free item (free and buy items are required to come from the same supplier and country).
        - Calculate the discount costs(for whichever is the current item, free or buy)

- ◻ If qty_thresh_buy_target of the buy item < qty_thresh_buy_qty, stop; you didn't get any discount
- ◻ Otherwise, figure out how many free items you actually get.
  - If the qty_thresh_recur_ind is 'N':
    ```
    free qty = deal_detail.qty_thresh_free_qty
    ```
  - If the qty_thresh_recur_ind is 'Y':
    - ♦ If buy item = free item:
      ```
      free qty = qty_thresh_free_qty *
      FLOOR(qty_thresh_buy_target /

      (qty_thresh_buy_qty + qty_thresh_free_qty))
      ```
    - ♦ If buy item different from free item:
      ```
      free qty = FLOOR(qty_thresh_buy_target /
      qty_thresh_buy_qty) *

      qty_thresh_free_qty
      ```
- ◻ If buy item = free item:
  - If qty_thresh_get_type is 'X', this is a "buy/free" discount:
    ```
    total discount = total get cost
    ```
  - If qty_thresh_get_type is 'P', this is a "buy/get % off" discount:
    ```
    total discount = (get item's unit_cost *
    qty_thresh_get_value / 100) * get qty
    ```
  - If qty_thresh_get_type is 'A', this is a "buy/get amt off" discount:
    ```
    total discount = qty_thresh_get_value * get qty
    ```
  - If qty_thresh_get_type is 'F', this is a "buy/get at fixed amt" discount:
    ```
    total discount = (get item's unit_cost -
    qty_thresh_get_value) * get qty
    ```
  - Discount rate = total discount / (buy item unit cost + buy target)
  - Discount = discount rate * get item unit cost
  - If the free item and the buy item are different:
  - If qty_thresh_get_type is 'X', this is a "buy/free" discount:
    ```
    total discount = total get cost
    ```
  - If qty_thresh_get_type is 'P', this is a "buy/get % off" discount:
    ```
    total discount = (get item's unit_cost *
    qty_thresh_get_value / 100) * get qty
    ```
  - If qty_thresh_get_type is 'A', this is a "buy/get amt off" discount:
    ```
    total discount = qty_thresh_get_value * get qty
    ```
  - If qty_thresh_get_type is 'F', this is a "buy/get at fixed amt" discount:
    ```
    total discount = (get item's unit_cost -
    qty_thresh_get_value) * get qty
    ```
  - Get discount rate = (get item cost * get qty) / total buy cost
  - Buy get discount rate = 1 – get discount rate
  - If current item is buy item

```
Discount = total discount * buy discount rate /
buy target
```

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

get_target_threshold_value:

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

get_unit_cost:

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

convert_currency:

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

post_insert_delete_records:

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

add_to_list:

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

init_list:

Initialize the linked list for target threshold values.

free_list:

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

size_arrays:

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

resize_array:

Allocate additional memory for the cost array.

free_arrays:

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

final:

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

I/O specification

N/A

Technical issues

There are two rebate_calc_type's: linear and scalar. Currently, the scalar type calculation is taken as the same as the linear type. These will be differentiated in a future release.

Testing scenarios

Test with:

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

Chapter 3 – Product security rebuild (sprdrbld)

Modification

The I/O specification section was modified to match the functionality changes made to the batch program.

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. SKUs 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 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 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 that will closely resemble the final destination security table will be sized to handle all the SKUs in the particular thread running. This array will be loaded with all the SKUs 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 have a rule affecting them and have therefore been “touched”. Each rule will affect certain SKUs in the array and their attributes may be changed multiple times. When they are changed, this indicator will be raised. After all the rules are processed for a given user/functional area, the data in the array that has the “touched” indicator raised will be written out to a SQL Loader file and its indicator reset. This cycle will be repeated until all users and functional areas are exhausted.

Table	Index	Select	Insert	Update	Delete
SEC_USER_GROUP	No	Yes	No	No	No
SEC_GROUP_PROD_MATRIX	No	Yes	No	No	No
V_RESTART_DEPT	No	Yes	No	No	No
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 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

This section contains information on the Logical unit of work and the driving cursor.

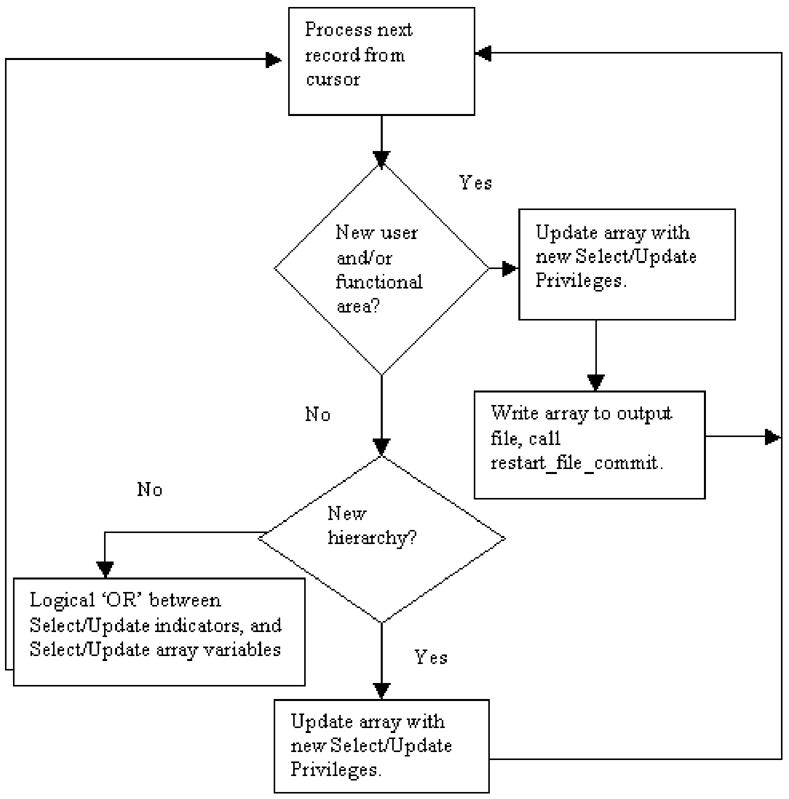
Logical unit of work

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.

Driving cursor

```
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;
```

Program flow


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

Shared modules

N/A

Function level description

Main()

Init()

- Check SYSTEM_VARIABLES.update_prd_sec_ind. If the indicator is not set then the program exit normally without further processing.
- Call retek_init() to get restart-recover variables.
- Get_total_skus()

  Get total skus in the current thread.
- Size_sku_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_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 level according to the last record rule.
- If it is new:
  - Call update_array() to blow security rule down to the SKU 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 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 table.

Note: In previous versions of RMS, outputfilename.extension_x was outputfilename_x.dat.

The format of the output file is as follows:

```
Column_code;user_id;SKU;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 4 – Approved warehouse transfers download (tsfdnld)

Modification

The I/O specification section was modified to match the functionality changes made to the batch program.

In the stock_order_header output file five new lines were added: Ship Address line 3, Ship Address line 4, Ship Address line 5, Billing Address line 4, Billing Address line 5. Fields were expected by RDM. RMS does not store data for these fields, so spaces will be sent down.

Function

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

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

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

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

Design overview

Table	Index	Select	Insert	Update	Delete
TSFALLOC	Yes	Yes	No	No	No
TSFHEAD	Yes	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	No	No	No
SHIPMENT	Yes	Yes	Yes	No	No
STORE_SHIP_DATE	Yes	Yes	No	No	No
WO_HEAD	Yes	Yes	No	No	No
WO_SKU_LOC	Yes	Yes	No	No	No
WO_WIP	Yes	Yes	No	No	No
ORDCUST	Yes	Yes	No	No	No
CUSTOMER	Yes	Yes	No	No	No
ITEM_TICKET	No	Yes	No	No	No
V_PACKSKU_QTY	No	Yes	No	No	No

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

Scheduling constraints

Processing Cycle: N/A

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

Pre-Processing: N/A

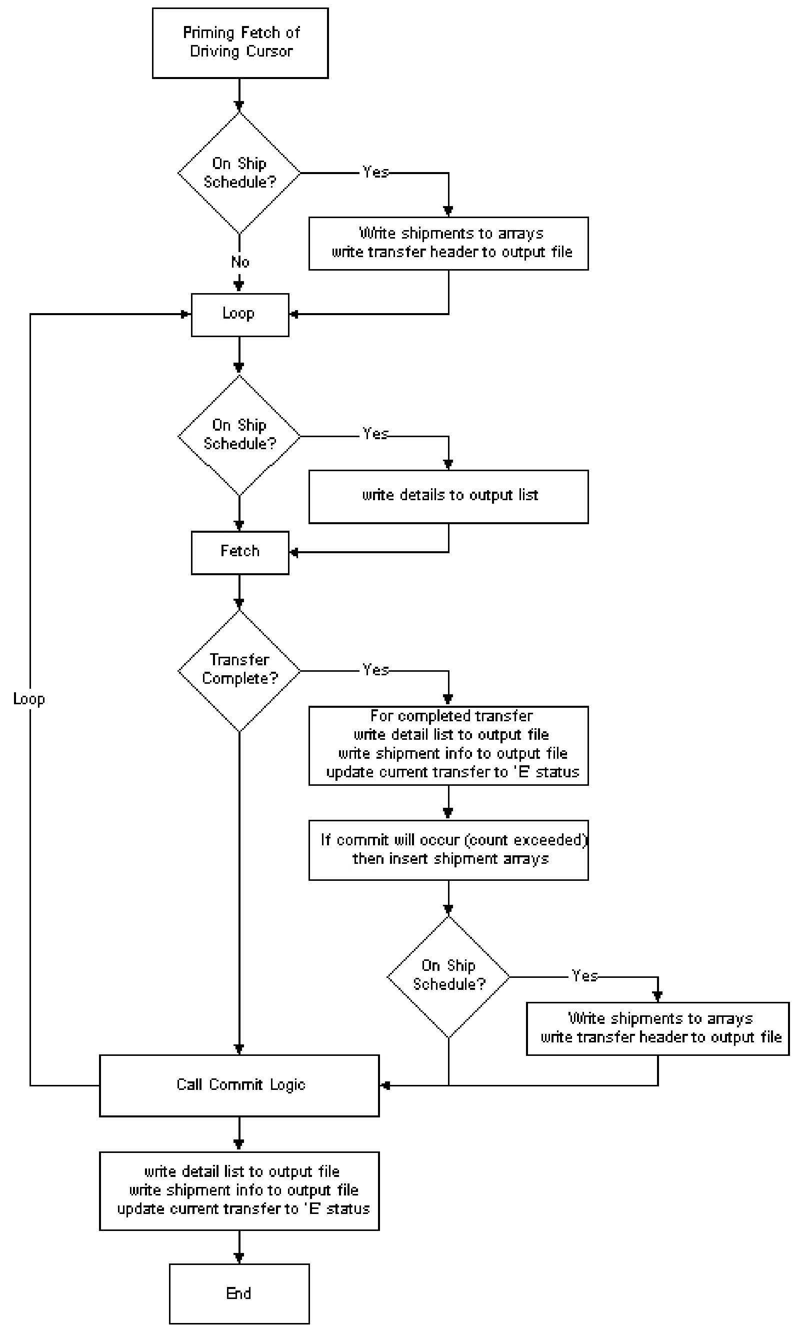
Post-Processing: N/A

Threading Scheme: N/A

Restart recovery

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

Program flow


Priming Fetch of Driving Cursor
On Ship Schedule?
Yes
Write shipments to arrays write transfer header to output file
No
Loop
On Ship Schedule?
Yes
write details to output list
Fetch
Transfer Complete?
Yes
For completed transfer write detail list to output file write shipment info to output file update current transfer to 'E' status
If commit will occur (count exceeded) then insert shipment arrays
On Ship Schedule?
Yes
Write shipments to arrays write transfer header to output file
Loop
Call Commit Logic
write detail list to output file write shipment info to output file update current transfer to 'E' status
End

Shared modules

NEXT_SHIPMENT_SQL used to get the next shipment number.

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

Function level description

Init()

- Initialize restart recovery.
- Open output file.
- Format header, detail, and shipment buffers (for writing output).
- Determine tomorrow's date
- Determine order type
- Call function get_order_type to determine order type
- Call function to write output file header information, write_std_header()

Process()

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

The flow of logic is as follows:

1 Fetch the first transfer record from the driving cursor.

2 Get_ship_flag (determines if current transfer is due to ship tomorrow)

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

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

Get_ship_flag()

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

validate_ship_schedule()

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

get_thead_info()

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

If customer order and ship direct

- set break by distro value = ‘Y’.
- populate billing and shipping addresses with customer address info.

- Set dest. Id = courier value from tsfhead
- Set Courier/route/service codes = NULL

If not customer order

- set break by distro value = ‘N’
- do not populate billing and shipping address
- set dest. Id = store or warehouse
- set courier/route/service codes = NULL

get_detail_info()

This function decodes the freight code.

```
if freight_code = ‘E’ then
expedite_flag  = ‘Y’;
else
   expedite_flag = ‘N’;
end if;
```

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

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

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

process_wo ()

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

Write_wo_to_list()

This function writes the work order information to the structures

Write_wo_to_file()

This function prints out the work order structure to flat file

Comp_tckt()

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

- If non Sellable ‘P’ack item is found

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

write_pack_to_list()

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

write_pack_to_file()

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

Write_std_header()

This function Increment counters and writes FHEAD record to file.

Write_std_trailer()

This function increments counters and writes FTAIL record to file

write_tail_to_file()

This function writes the TTAIL structure to the output file

write_detail_to_list()

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

add_dtl_to_list()

This function will add ps_temp_dtl string to linked list.

get_order_type()

This function gets order type from code_detail.

write_head_to_str()

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

Write_recs_to_struct()

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

write_list_to_file()

This function writes linked list detail records to file

update_records()

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

Final()

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

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

I/O specification

Output files should be specified on the command line

Transfer download file

Record Name	Field Name	Field Type	Field 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)	TSFD	Identified file as 'Inventory Adjustments'
	File Create Date	Date	Sysdate	Date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	Specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	Specified by external system	Used to force unique transaction check
	Action Type	Char(1)	‘A’ (hardcode)	‘A’dd, ‘D’elete, ‘M’odify
	Location (DC)	Number (4)	Tsfhead.from_loc	Code for the DC.

Record Name	Field Name	Field Type	Field Value	Description
	Transaction Date/Time	YYYYMMDDHHMI	Period.vdate	Date/Time created in RMS
	Distributio n Number	Char(9)	Shipment.shipment	Unique identifier of the distribution.
	Download Comment	Char (30)	NULL	Comment to be printed on the label (for future use)
	Pick-Not-Before-date	YYYYMMDD	Period.vdate	Date before which merchandise will not be distributed
	Pick-Not-After-Date	YYYYMMDD	Period.vdate + (specified time from codes table)	Date by which merchandise must be distributed. Extra days will be determined by a code type = 'DATE'
	Event Code	Char(6)	NULL or tsfalloc.tsfalloc_no	Identifier of event. Only used for stock allocations
	Event Description	Char(25)	NULL or tsfalloc.alloc_desc	Description of event. Only used for stock allocations
	Priority	Char(4)	Default to 1	Priority 1=highest
	Order Type	Char(9)	Default from system optionTables	Order type (Automatic, Manual or Wave)
	Break by Distro	Char(1)	Default from codes tables	Controls the mixing of orders (distros) in a container
	Carrier Code	Char(1)	NULL	Code of the carrier for the order
	Carrier Service Code	Char(6)	NULL	Carrier's service code for the delivery, First Class, etc.
	Route	Char(10)	NULL	Route specified for the delivery
	Ship Address Description	Char(30)	NULL or customer address	Used to store only customer order (ship direct) addresses.

Record Name	Field Name	Field Type	Field Value	Description
	Ship Address line 1	Char(30)	NULL or customer address	Shipping address line 1. Used to store only customer order (ship direct) addresses.
	Ship Address line 2	Char(30)	NULL or customer address	Shipping address line 2. Used to store only customer order (ship direct) addresses.
	Ship Address line 3	Char(30)	NULL or customer address	Shipping address line 3. Used to store only customer order (ship direct) addresses.
	Ship Address line 4	Char(30)	NULL or customer address	Shipping address line 4. Used to store only customer order (ship direct) addresses.
	Ship Address line 5	Char(30)	NULL or customer address	Shipping address line 5. Used to store only customer order (ship direct) addresses.
	City	Char(25)	NULL or customer address	Shipping city. Used to store only customer order (ship direct) addresses.
	State	Char(3)	NULL or customer address	Shipping state. Used to store only customer order (ship direct) addresses.
	Zip	Char(10)	NULL or customer address	Shipping zip. Used to store only customer order (ship direct) addresses.
	Billing Address Description	Char(30)	NULL or customer address	The description (such as company name, etc.). This is the first line of the address block. Used to store only customer order (ship direct) addresses.
	Billing Address line 1	Char(30)	NULL or customer address	Billing address line 1. Used to store only customer order (ship direct) addresses.

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

Record Name	Field Name	Field Type	Field Value	Description
	Requested Unit Qty	Num(12,4)	Tsfdetail.tsf_qty	Number of units to distribute to the destination
	Destination ID	Number (4)	Tsfhead.routing_code (if ship direct to Customer order)Tsfhead.to_loc (if store or wh)	Identifier of shipping destination. If customer order and ship direct, then field contains a carrier value. If it is direct to store or warehouse, then populate with the store or warehouse location.
	Price	Number (7,2)	Item_zone_price.unit_retail	Price of merchandise
	Print UPC Flag (‘Y’,’N’)	Char(1)	‘N’ (hardcode)	Whether to print UPC on tickets (Future use)
	Ticket Type	Number (4)	Item_ticket.ticket_type	Type of ticket refers to ticket type table. This field will be populated with the “ticket at receipt”.
	Priority	NUMBER (4)	1 (hardcode)	Priority 1 = highest
	Expedite Flag	VARCHAR(1)‘Y’ or ‘N’	Tsfhead.freight_code (translate value to ‘Y’ or ‘N’)	Flag indicating whether the order should be shipped via normal or expedited carrier service
Transaction Trailer	File type record descriptor	Char(5)	TTAIL	Identifies file record type
	File Line sequence	Number(10)	Specified by external system	Line number of the current file
	Transaction detail line count	Number(6)	Sum of detail lines	Sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file

Record Name	Field Name	Field Type	Field Value	Description
	File Line Count	Number(10)	total detail + transaction head lines	sum of all transaction lines, not including file header and trailer

Work order download file

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes:0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	OWOD	Identifies file as ‘Outbound Work Order Download’
	File Create Date	Date	Create date	date file was written by external system
Trans-action Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	‘A’	The action being performed on the work order. This will always be ‘A’ since transfer work orders can’t be modified once they’ve been extracted.
	Location (DC)	Char(4)	Wo_sku_loc.wh	When an item is crossdocked, this field holds the value of the flow-through warehouse. Otherwise it holds the value of the final destination.
	Transaction Date/Time	Char (12) format: YYYYMMDDHHMI	Vdate	sysdate without the seconds
	Distributio n Number	Char(9)	Shipment	This field will hold the shipment number of the transfer the work order is associated with.
	Item ID	Char (16)	Wo_sku_loc.sku	Valid item identifier for a staple SKU, fashion SKU, or Pack Item
	Dest ID	Number(4)	Wo_sku_loc.location	Unique identifier of the final shipping destination.

Record Name	Field Name	Field Type	Default Value	Description
	WIP Sequence No.	Number(7)	Wo_wip.seq_no	Work in Progress Sequence Number
	WO WIP Code	Char (6)	Wo_wip.code	WWIP code from codes table
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

Component ticketing file layout

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

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

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

Technical issues

N/A

Chapter 5 – Stock position download (sohdnld)

Modification

Modified the output specifications to match the functionality changes made to the match program. Added the system date to the format of the output file. Updated the function level description section to include that the system date is printed out to the output file.

Design overview

This affects the functional area of the RMS to RPP interface.

This ad-hoc batch program will be run to communicate current stock on hand, retail, and cost information to RPP. The information will be sent to RPP at the SKU/location level.

Scheduling constraints

This is an ad-hoc program. It can be run anytime (most likely in phase 4).

Restart recovery

This program has a unique logical unit of work (LUW) of item/location.

Shared modules

Curconv.pc/h – convert_to_primary. The library call is used to convert the retail and cost from the store's or wh's local currency to the system's primary currency.

Function level description

Init()

- Initialize restart recovery and file processing.
- Get the vdate from period and std_av_ind from system_options.
- Setup the output file.

Process()

Define driving cursor here:

- It should be a union all with four parts. Each part will be pretty much the same, except they will be driven by the different item/location tables (WIN_STORE, WIN_WH, RAG_SKUS_ST, RAG_SKUS_WH). The cursors should bring back every SKU, location with their associated stock_on_hand (+pack_comp_soh for warehouses), total unit retail, and total cost. The cost should be fetched with a DECODE. If SYSTEM_OPTIONS.STD_AV_IND = 'A', total cost should use av_cost*stock_on_hand, else total cost should use unit_cost*stock_on_hand. The cursor should be threaded by location, use V_RESTART_STORE_WH. (be sure to order by location then SKU – this will allow for the best performance by the convert_to_primary() library call)

Call size_array() and define_buffer() .

Use array processing to fetch the driving cursor

For each record brought back by the driving cursor, call write_to_file().

Call retek_forece_commit() once commit_max_counter records have been processed.

Call free_array() after the processing is finished.

Define_buffer()

Set up a string with the fprintf format that will be used when writing records to the output file.

Size_array()

Size the driving cursor fetch array to the commit max counter.

Free_array()

Give back the memory allocated by size_array().

Write_to_file()

Convert the retail and cost to the primary currency by calling:

```
int convert_to_primary(char    *ls_idnt,
                       char    *ls_idnt_type,
                       char    *ls_zone_group,
                       char    *ls_date,
                       double  *ld_amount);
```

ls_idnt should be sent as the location (store or wh)

ls_idnd_type should be sent as 'W' if the location is a wh, 'S' if the location is a store.

Ls_zone_group should be sent as "" (empty string or NULL)

Ls_date should be sent as the vdate in 'YYYYMMDD'

Ld_amount should be sent as the cost or retail.

Convert_to_primary() will be called twice per driving cursor record once for the cost, once for the retail.

Print out the SKU, location, loc_type, system_date, stock_on_hand, retail, and cost to the output file.

Final()

Clean up restart recovery and file processing.

I/O specifications

'Table-To-Table'

Input

This program sources WIN_STORE, WIN_WH, RAG_SKUS_ST, and RAG_SKUS_WH. See the driving cursor discussion in the functional level description.

Output

N/A

Files

Input

N/A

Output

The output file should be names sohdnld.dat.N. Where N is the thread number.

Note: The thread number has nothing to do with the domain in this program.

Field Name	Field Type	Description
SKU	Number(20) left justified	RMS item identifier. Left justified
Location	Number(20) left justified	Store or warehouse identifier. Left justified
Loc Type	Char(1)	Indicates whether the Location is a store or wh. S – if the location is a store W – if the location is a warehouse
System Date (vdate)	Date(8)	Date the output file was created.

Field Name	Field Type	Description
Stock-on-hand	Number(17)	Total stock-on-hand for the item at the given location. Right justified, decimal point is included in output file
Retail	Number(25)	Total extended retail for the item at the given location. Right justified, decimal point is included in output file
Cost	Number(25)	Total extended average cost (av_cost) or last cost (unit_cost) for the item at the given location – depending on the SYSTEM_OPTIONS. STD_AV_IND. Right justified, decimal point is included in output file

Design assumptions

This program doesn't need to split its output by domain.

This program will be run infrequently, thus performance considerations were not thoroughly investigated in the design process.

Technical issues

N/A

Chapter 6 – RMS batch schedule

Deals (SIR 35190)

Modification

The batch schedule references were changed to show sccext at the end of Phase 3 and dealcalc and orddscnt at the beginning of Phase 4.

Note: The same change has been made to the RMS Operations Guide.

Enterprise 9.0 Batch Schedule

Phase 0

aristart (ARI) **	aricntrl (ARI) **	aristop (ARI) **
r-r script		
dlyprg		
salins		
cntrmain		
vatdlxpl		

***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 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)
ditinsrt		

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				
(lcmt730)	lcupld			
(lcmt798)	lcup798			
fdayupld				

pctrandn

pre

tsfresv

hstwksst (weekly)
hstwkswh (weekly)
hstwkfst (weekly)
hstwkfwh (weekly)
hstbld (rebuild wkly)
wasteadj

szrtbld

fcstrbld

salstage

supmth
dealcls

** 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.

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.

Sales Audit

sagetref	saimptlog	(sqlldr)	savouch****	saimptlogfin	saimpadj*	satotals	sarules	(Forms Auditing)
samastersfi	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 use
during the loading of t

Enterprise 9.0 Batch Schedule

Phase 3

											sccext	post
pctranex | post	pcext pccext	pccrext										
rplatupd	rpladjf rpladjs	cntrordb	post	rplext	cntrprss	vrplbld						
		reqext				pre	rplbld	supcnstr	rplprg	post		
tsfcomb												
										whstrasg		
stkxplst stkxplwh	post stkupd											
post												
saldly	stkdly	salapnd salweek	post	salmth	post	saleoh	pre	fifgldn1(F(F fifgldn2(FIF) fifgldn3(FIF)	fifpldp(FIF)			
post												

sapreexp	saexprms*** saexpim*** saexprdw*** saexpach*** saexpuar*** saexpgl***	sapurge

ed to correct any errors found
the data, totaling and rules checking.

*** after sprdrbld, must run SQLLoad using
sprdrbld.ctl to load data into database

Phase 4

dealcalc orddscnt	post	ordrev	edidlord powodld asndnld	tsfdcdld
		saaldnld	allocupd fifrecd1 (FIF)	
		edidlprd edidldeb asndnld tsfdnld reclsdly	post	
		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 stlgdnld sohdnld	onordext	onorddnld

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		

System (SIR 34728)

Modification

The location of Aristart and Aristop was switched. fifgldn1&2 were moved from phase 1 to phase 3. prepost fifgldn was moved from running before the program to after it for fifgldn1.

Enterprise 9.0 Batch Schedule

Phase 0		
aristop (ARI) **	aricntrl (ARI) **	aristart (ARI) **
r-r script		
dlyprg		
salins		
cntrmain		
vatdlxpl		

***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 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)
ditinsrt		

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				
(lcmt730)	lcupld			
(lcmt798)	lcup798			
fdayupld				

pctrandn

pre

tsfresv

hstwksst (weekly)
hstwkswh (weekly)
hstwkfst (weekly)
hstwkfwh (weekly)
hstbld (rebuild wkly)
wasteadj

szrtbld

fcstrbld

salstage

supmth
dealcls

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.

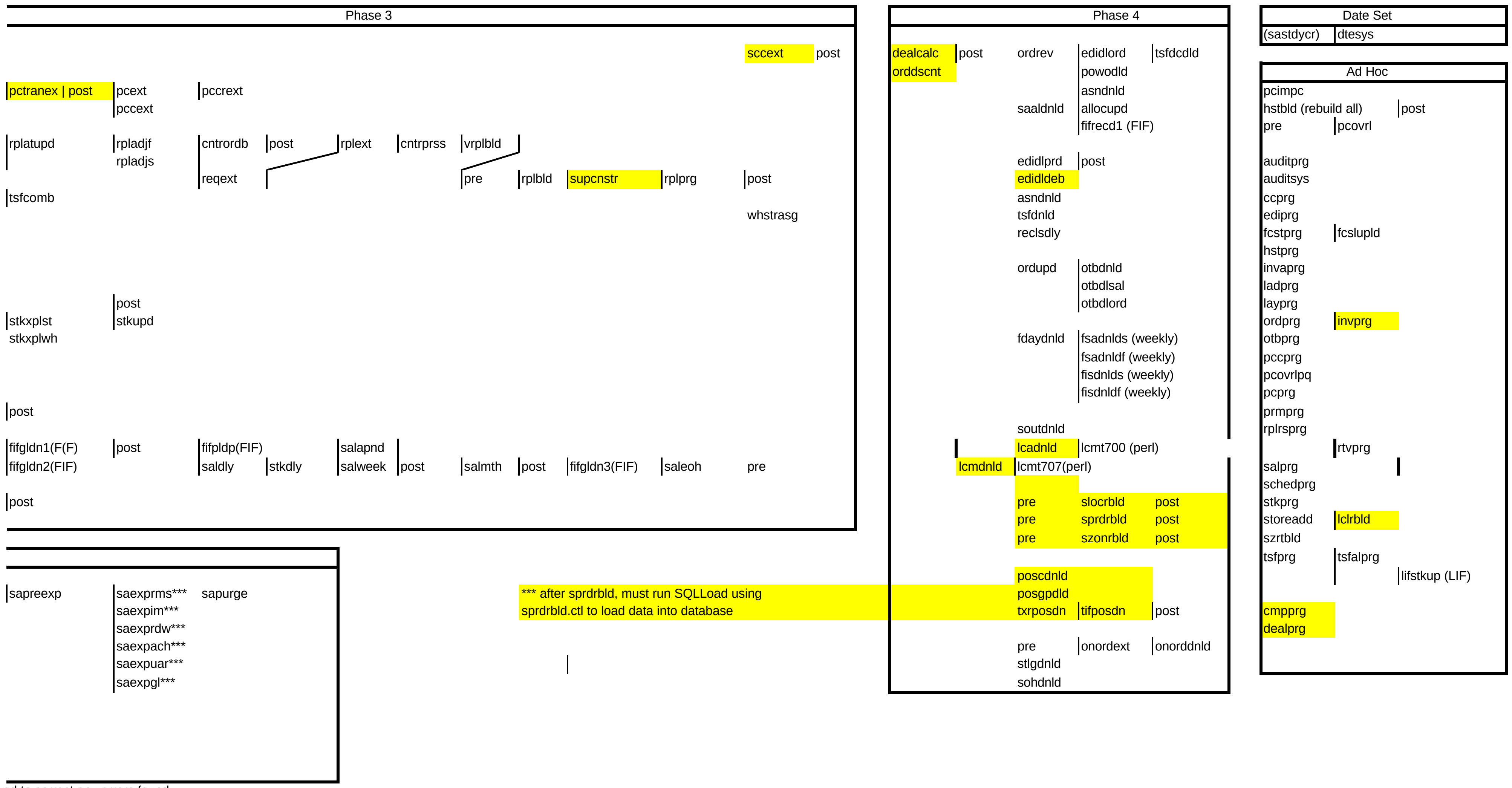
** Note that the ARI programs (aristart, aricntrl, aristop) must be run
with no other resources accessing the system. They can be run before
or after the rest of the batch schedule.

Sales Audit								
sagetref	saimptlog	(sqlldr)	savouch****	saimptlogfin	saimpadj*	satotals	sarules	(Forms Auditing)
samastersfi	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 use
during the loading of t

Enterprise 9.0 Batch Schedule



*** after sprdrbld, must run SQLLoad using
sprdrbld.ctl to load data into database

ed to correct any errors found
the data, totaling and rules checking.