Retek® Merchandising System™ 10.1.5

Operations Guide

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

Information in this documentation is subject to change without notice.

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

Corporate Headquarters:

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

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

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

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

Printed in the United States of America.

Customer Support

Customer Support hours:

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

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

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

When contacting Customer Support, please provide:

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

Contents

New and Changed Upload from Supplier [ediupcat]

Design Overview

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

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

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

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

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

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

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

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

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

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

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

Scheduling Constraints

Processing Cycle:	Daily, Phase 2
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	(File-based processing, don't use multithreading)

Restart Recovery

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

Program Flow

N/A

Shared Modules

SQL_LIB.BATCH_MSG—to write error messages

CURRENCY_SQL.CONVERT_BY_LOCATION—convert unit cost and unit retail

COUNTRY_VALIDATE_SQL.EXISTS_ON_TABLE—validates origin_country_id

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

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

Function Level Description

init()

- get vdate
- Restart/recovery initialization
- open input file and read file header
- open output file (run-time reject file)

process()

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

format_FDETL():

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

validate_FDETL()

- Validate that the input file has at least one of the item, VPN and ref_item fields populated. If none of the above fields exists, issue an error message and return NON_FATAL.
- Validate supplier by calling validate_supplier.
- When item parent passed in from the input file is not null, call validate_item_parent.
- Call validate_parent_VPN to validate that the parent VPN exists in the system and find the parent_item according to the parent_vpn.
- If both item parent and parent VPN are not null, compare the input item parent with the item parent retrieved from function validate_parent_VPN, if they are differnt, log an error and return NON_FATAL. Otherwise, if the input item parent is null, the item parent retrieved from function validate_parent_VPN should be used.
- Call functions validate_origin_country_id and validate_uom.
- When both ref_item_type and ref_item are presented, call function check_ref_item and passing the ref_item and ref_item_type to the function.

- If the item field has value,
  - call function validate_item;
  - if the item does not equal ref_item, call function validate_ref_item;
  - if the item VPN is not null, call validate_vpn.
- If the record's item field does not have a value, process as follows:
  - Call get_item
- If item parent is not null, item parent description or item parent number type is null, call function get_item_info(). Pass in item parent number and variables to hold the item_parent_desc and item_parent_number_type. Note dummy variables are needed to hold other parameters.
- If both VPN and ref_item are not null and their corresponding item exists in RMS, call function validate_VPN_vs_ref_item to make sure that the item is not above the transaction level. Since an item that is above the transaction level could not have a ref_item. If the function call doesn't return true, return whatever the function returns.
- If the case_ref_item field is not null, call function process_case.

validate_supplier():

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

validate_item()

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

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

validate_ref_item()

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

check_ref_item()

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

validate_parent_VPN()

Validate the parent_VPN against the item_supplier and edi_new_item tables.

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

validate_origin_country_id()

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

validate_uom()

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

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

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

validate_each_uom()

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

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

validate_vpn()

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

Validate_item_parent()

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

Find_ item_by_ref_item()

- The function will find the transaction level item that corresponding to the ref_item(item ref_item or case ref_item) passed in. It will take ref_item, item and item_exists as parameters.
- Since a ref_item could actually be a transaction level item or be a sub_transaction level item, crease a cursor c_item_by_ref_item, do a decode selection from item_master table to select item from item_master table if an item equals the passed in ref_item and item_level equals tran_level, or to select item_parent if the item equals the ref_item and the item_level = tran_level +1.
- If data is found set the item_exist to 1 and store the found item in the passed in variable. Otherwise, set the item_exist to 0. If no error occurred, return true. Otherwise, return fatal.

get_item()

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

Get_item_info()

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

convert_currency()

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

process_item()

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

insert_new_item()

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

update_item_info()

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

- vdate – processed date
- NVL(item_desc, edi_new_item.item_desc)
- NVL(short_desc, edi_new_item.short_desc)
- NVL(case_cost, edi_new_item.case_cost) – for new items only
- NVL(unit_cost, edi_new_item.unit_cost) – for new items only
- NVL(packing_method, edi_new_item.packing_method)
- NVL(gross_unit_weight, edi_new_item.gross_unit_weight)
- NVL(net_unit_weight, edi_new_item.net_unit_weight)
- NVL(unit_weight_uom, edi_new_item.unit_weight_uom)
- NVL(unit_length, edi_new_item.unit_length)
- NVL(unit_width, edi_new_item.unit_width)
- NVL(unit_height, edi_new_item.unit_height)
- NVL(unit_lwh_uom, edi_new_item.unit_lwh_uom)

- NVL(unit_liquid_volume, edi_new_item.unit_liquid_volume)
- NVL(unit_liquid_volume_uom, edi_new_item.unit_liquid_volume_uom)
- NVL(gross_case_weight, edi_new_item.gross_case_weight)
- NVL(net_case_weight, edi_new_item.net_unit_weight)
- NVL(case_weight_uom, edi_new_item.case_weight_uom)
- NVL(case_length, edi_new_item.case_length)
- NVL(case_width, edi_new_item.case_width)
- NVL(case_height, edi_new_item.case_height)
- NVL(case_lwh_uom, edi_new_item.case_lwh_uom)
- NVL(case_liquid_volume, edi_new_item.case_liquid_volume)
- NVL(case_liquid_volume_uom, edi_new_item.case_liquid_volume_uom)
- NVL(gross_pallet_weight, edi_new_item.gross_pallet_weight)
- NVL(net_pallet_weight, edi_new_item.net_pallet_weight)
- NVL(pallet_weight_uom, edi_new_item.pallet_weight_uom)
- NVL(pallet_length, edi_new_item.pallet_length)
- NVL(pallet_width, edi_new_item.pallet_width)
- NVL(pallet_height, edi_new_item.pallet_height)
- NVL(pallet_lwh_uom, edi_new_item.pallet_lwh_uom)
- NVL(lead_time, edi_new_item.lead_time)
- NVL(min_ord_qty, edi_new_item.min_ord_qty)
- NVL(max_ord_qty, edi_new_item.max_ord_qty)
- NVL(uom_conversion_factor, edi_new_item.uom_conversion_factor)
- NVL(standard_uom, edi_new_item.standard_uom)
- NVL(supp_diff_1, edi_new_item.supp_diff_1)
- NVL(supp_diff_2, edi_new_item.supp_diff_2)
- NVL(supp_diff_3, edi_new_item.supp_diff_3)
- NVL(supp_diff_4, edi_new_item.supp_diff_4)
- NVL(supp_pack_size, edi_new_item.supp_pack_size)
- NVL(inner_pack_size, edi_new_item.inner_pack_size)

Validate_VPN_vs_ref_item():

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

process_case()

- First, check if this is a new case pack. Call function find_item_by_ref_item to find the pack no that corresponding to the case_ref_item. Note this indicator will be used to populate the edi_new_item table's new_case_pack_ind field if the case_ref_item is valid. Pass in the case_ref_item to the function and also the variables to hold the pack no and the pack exists indicator. If the pack no is not found in RMS, check to make sure a type for the case_ref_item was specified in the input file. If not, log an error and return NON_FATAL. If pack no is found in the RMS, find the component item from the packitem table for the pack_no. Compare the pack component item found from the cursor with the item that from the input file, if they are different, log an error and return NON_FATAL.
- Next, compare the case pack exist indicator and the item exist indicator:
  - If both case pack and item are new to RMS, if case_cost is null and unit_cost is provided by the input file, calculate the case_cost by multiplying the unit_cost and the pack_size. Otherwise, if unit_cost is null and the case_cost is presented in the input file, divided the cast_cost by the pack_size to populate the unit_cost field.
- Finally, if both of the case_ref_item and case_ref_item_type are not null, call function check_ref_item and pass in the case_ref_item and case_ref_item_type. If the function doesn't return successfully, return whatever is returned from the function. Otherwise, return true.

final()

- restart/recovery close, close files

I/O Specification

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

- FHEAD file header
- FDETL item info
- FTAIL file trailer

Input Files

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

Record Name	Field Name	Field Type	Default Value	Description
	Ref item type	Char(6)		Reference item type. Valid reference types are stored in the code_detail table under Code Type of 'UPCT' and listed as follows: ITEM - Retek Item Number UPC-A - UPC-A UPC-AS - UPC-A with Supplement UPC-E - UPC-E UPC-ES - UPC-E with Supplement EAN8 - EAN8 EAN13 - EAN13 EAN13S - EAN13 with Supplement ISBN - ISBN NDC - NDC/NHRIC - National Drug Code PLU - PLU VPLU - Variable Weight PLU SSCC - SSCC Shipper Carton UCC14 - SCC-14 (blank if none).
	Item Parent	Char (25)		Retek Item Parent which uniquely identifies the item/group at the level above the item.
	Parent VPN	Char(30)		Vendor style id
	VPN	Char(30)		Vendor product number (blank if none) Must be in all capitals
	Supplier item differentiator 1	Char(80)		Item differentiator description. For example, color, size, descriptions. This field is displayed later when entering the item into Retek to use as a basis for choosing an appropriate differentiator within Retek.
	Supplier item differentiator 2	Char(80)		Item differentiator description. For example, color, size, descriptions. This field is displayed later when entering the item into Retek to use as a basis for choosing an appropriate differentiator within Retek.

Record Name	Field Name	Field Type	Default Value	Description
	Supplier item differentiator 3	Char(80)		Item differentiator description. For example, color, size, descriptions. This field is displayed later when entering the item into Retek to use as a basis for choosing an appropriate differentiator within Retek.
	Supplier item differentiator 4	Char(80)		Item differentiator description. For example, color, size, descriptions. This field is displayed later when entering the item into Retek to use as a basis for choosing an appropriate differentiator within Retek.
	Item description	Char(100)		Item description
	Short description	Char(20)		Item short description for point of sales.
	Effective date	Char(14)		Effective date, YYYYMMDDHH24MISS
	Min order qty	Number(12)		Minimum order quantity (4 implied decimal places)
	Max order qty	Number(12)		Maximum order quantity (4 implied decimal places)
	Lead time	Number(4)		Days from PO receipt to shipment
	Unit cost	Number(20)		Unit cost, 4 implied decimal places
	Gross unit weight	Number(12)		Gross unit weight (4 implied decimal places). The gross numeric value of weight per unit.
	Net unit weight	Number(12)		Net unit weight (4 implied decimal places). The net numeric value of weight per unit.
	Unit weight UOM	Char(4)		Item unit weight unit of measure
	Unit length	Number(12)		Item unit length (4 implied decimal places)
	Unit width	Number(12)		Item unit width (4 implied decimal places)
	Unit height	Number(12)		Item unit height (4 implied decimal places)
	Unit lwh UOM	Char(4)		Item unit dimension unit of measure.
	Unit liquid volume	Number(12)		Item unit liquid volume or capacity (4 implied decimal places)
	Unit liquid volume UOM	Char(4)		Unit of measure of the item liquid volume/capacity

Record Name	Field Name	Field Type	Default Value	Description
	Case ref item	Char(25)		Case reference number. For example: case UPC code.
	Case ref item type	Char(6)		Case reference number type. Valid case reference item types are stored in the code_detail table under Code Type of 'UPCT' and listed as follows: ITEM - Retek Item Number UPC-A - UPC-A UPC-AS - UPC-A with Supplement UPC-E - UPC-E UPC-ES - UPC-E with Supplement EAN8 - EAN8 EAN13 - EAN13 EAN13S - EAN13 with Supplement ISBN - ISBN NDC - NDC/NHRIC - National Drug Code PLU – PLU VPLU - Variable Weight PLU SSCC - SSCC Shipper Carton UCC14 - SCC-14 (blank if none).
	Case item desc	Char(100)		Case item description
	Case cost	number(20)		Case Cost (4 implied decimal places)
	Gross case weight	Number(12)		Gross weight of the case (4 implied decimal places)
	Net case weight	Number(12)		Net weight of the case (4 implied decimal places)
	Case weight UOM	Char(4)		Unit of measure of the case weight
	Case length	Number(12)		Case length (4 implied decimal places)
	Case width	Number(12)		Case width (4 implied decimal places)
	Case height	Number(12)		Case height (4 implied decimal places)
	Case lwh UOM	Char(4)		Case dimension unit of measure.
	Case liquid volume	Number(12)		Case liquid volume or capacity (4 implied decimal places)

Record Name	Field Name	Field Type	Default Value	Description
	Case liquid volume UOM	Char(4)		Unit of measure of the case liquid volume/capacity
	Gross pallet weight	Number(12)		Gross pallet weight (4 implied decimal places)
	Net pallet weight	Number(12)		Net pallet weight (4 implied decimal places)
	Pallet weight UOM	Char(4)		Unit of measure of the pallet weight
	Pallet length	Number(12)		Pallet length (4 implied decimal places)
	Pallet width	Number(12)		Pallet width (4 implied decimal places)
	Pallet height	Number(12)		Pallet height (4 implied decimal places)
	Pallet lwh UOM	Char(4)		Pallet dimension unit of measure.
	Ti	Number(12)		Shipping units (cases) in one tier of a pallet (4 implied decimal places)
	Hi	Number(12)		Number of tiers in a pallet (height). (4 implied decimal places)
	Pack Size	Number(12)		Supplied pack size. I.e., Number of eaches per case pack. This is the quantity that orders must be placed in multiples of for the supplier for the item.
	Inner pack size	Number(12)		Supplied inner pack size. I.e., Number of eaches per inner container.
	Origin Country ID	Char(3)		Supplied origin country ID.
	Standard UOM	Char(4)		Unit of measure in which stock of the item is tracked at a corporate level.
	UOM Conversion Factor	Number(20)		Conversion Factor, 10 implied decimal places. Conversion factor between an "Each" and the standard_uom when the standard_uom is not in the quantity class (e.g. if standard_uom = lb and 1 lb = 10 eaches, this factor will be 10). This factor will be used to convert sales and stock data when an item is retailed in eaches but does not have eaches as its standard unit of measure.
	Packing Method	Char(6)		Packing Method code (HANG,FLAT)

Record Name	Field Name	Field Type	Default Value	Description
	Location	Number(10)		RETEK location that the supplier distributes to or this may be a number used by the supplier to identify a non-RETEK location.
	Location Type	Char(1)		This field will contain the type of location ('S' for store and 'W' for warehouse).
	Bracket Value 1	Number (12,4)		This will contain the primary bracket value of the supplier.
	Bracket UOM 1	Char(4)		This field will contain the unit of measure of the primary bracket.
	Bracket Type 1	Char (6)		This field will contain the UOM class.
	Bracket Value 2	Number (12,4)		This will contain the secondary bracket value for the supplier.
	Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket.
	Case Bracket Value 1	Number (12,4)		This will contain the primary bracket value of the supplier for a case UPC.
	Case Bracket UOM 1	Char(4)		This field will contain the unit of measure of the primary bracket for a case UPC.
	Case Bracket Type 1	Char (6)		This field will contain the UOM class for a case UPC.
	Case Bracket Value 2	Number (12,4)		This will contain the secondary bracket value for the supplier for a case UPC.
	Case Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket for a case UPC.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Numeric ID(10)		Number of records/transactions processed in current file (only records between head & tail)

Test Conditions

Conditions	Expected Results	Programmer Sign-off
No records	no processing	
Missing required information	write TDETL line to reject file	
Process a valid input file:		
for a new item	Insert edi_new_item record – include the unit retail and cost	
for existing items	Insert edi_new_item record, if changes to other than cost. Only insert into edi_cost_change if cost change present	
for a new item with existing edi_new_item and edi_cost_change records	Update the edi_new_item and edi_cost_change tables	
Input file contains item, ref_item, and VPN:		
invalid ref_item	write TDETL to reject file	
invalid vpn	write TDETL to reject file	
Input file contains ref_item andVPN:		
invalid ref_item	write TDETL to reject file	
invalid vpn	write TDETL to reject file	
item with no ref_item (in Retek) but a valid VPN	Insert/update edi_new_item and edi_cost_change tables	

Technical Issues

1 Unit retail and cost will be inserted into edi_new_item table for new items only.

2 We are not using permanent substitutions.

3 It is assumed that currency will be in the supplier's currency. This currency must be converted to primary currency for the edi_new_item table. No translation is necessary for the edi_cost_change table since that stores the supplier's currency.

4 All input/validation errors will be non-fatal. All Oracle errors will be fatal.

ediupinv.pc

Design Overview

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

Scheduling Constraints

Pre/Post Logic Description

Table	Index	Select	Insert	Update	Delete
SYSTEM_OPTIONS	Yes	Yes	No	No	No
PERIOD	Yes	Yes	No	No	No
ORDHEAD	Yes	Yes	No	No	No
UPC_EAN	No	Yes	No	No	No
INVC_HEAD		Yes	Yes	No	No
INVC_DETAIL		No	Yes	No	No
INVC_XREF		No	Yes	No	No
INVC_MATCH_QUEUE		No	Yes	No	No
INVC_MATCH_WKSHT		No	Yes	No	No
INVC_NON_MERCH		No	Yes	No	No
INVC_MERCH_VAT		No	Yes	No	No
TERMS		Yes	No	No	No
VAT_CODES		Yes	No	No	No
RTV_HEAD		Yes	No	No	No

Processing Cycle: Phase I (daily)

Scheduling Diagram:

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A –file based.

Restart Recovery

Logical Unit of Work (recommended Commit check points)

The logical unit of work is the invoice.

Commits should occur every 2500 records (subject to experimentation).

Shared Modules

INVC_SQL.NEXT_INVC_ID—get a new invoice id

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

Function Level Description

Set up structures for each different type of file line.

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

init()

- Restart/recovery initialization
- Fetch vdate from period and vat_ind and invc_match_qty_ind from system_options. Vdate will be used as the create date.
- Open input file and reject file
- Read in FHEAD line (or seek to the right place in the file, if doing a restart)

process()

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

1 Deal with header-level information (THEAD line)—format the line (format_THEAD), do basic validation (validate_THEAD), get a new invoice number (get_new_invc_id) and insert into invc_head, invc_match_queue, and invc_non_merch tables.

2 Deal with VAT information (TVATS lines—format_TVATS, validate_TVATS). If VAT is turned on in the system, call insert_merch_vat. (if turned off, read through the lines but don't process).

3 Deal with cross reference-level information (TXREF lines – format_xref, validate_xref). If order numbers exist, call insert_order_xref to insert the order number into invc_xref table. If ASN numbers exist, loop through all shipment numbers containing the ASN number in the ext_shipment field on the shipment table and call insert_shipment_xref to insert the shipment numbers into the invc_xref table.

4 Deal with item_level information (TDETL line). Format, validate, insert into invc_detail table.

5 Deal with ASN/carton information (TSHIP line). Match ASN to shipment in RMS. If ASN is not used, will use the order on INVC_XREF. Format, validate, insert into invc_match_wksht. ASN and carton are now optional.

6 When get to TTAIL line, check number of lines in transaction from file against number of lines read in by program. If they don't match, reject the transaction.

7 If a transaction failed validation anywhere, roll back to a savepoint set at the beginning of the transaction and write the transaction to the reject file (write_to_reject)

8 When transaction has been processed, call restart commit logic if necessary.

get_new_invc_id()

- Call the invc_sql.next_invc_id package to get a new invoice number

process_THEAD()

- Format and validate header variables and do necessary table inserts.

process_TXREF()

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

process_TDETL()

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

process_TVATS()

- Format and validate TVATS variables and insert into invc_merch_vat.

process_TSHIP()

- Format and validate TSHIP variables and insert into invc_match_wksht.

process_FTAIL()

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

format_header_fields()

- parse THEAD line into variables

format_tvats_fields()

- parse TVATS line into variables

format_txref_fields()

- parse TXREF line into variables

format_detail_fields()

- parse TDETL line into variables

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

- item valid (check item_master) (If only given Ref. Item, get transaction – level item from item_master)
- partner/partner type valid if given (use validate_partner_id, and check against partner table)
- merchandise invoices cannot be associated with a partner, and must have supplier
- credit notes from a partner cannot have item records attached to it unless partner type is in (S1,S2,S3).
- order number valid if given (check ordhead)
- reference invoice id valid if given (check invc_head)
- RTV order number valid if given (check against rtv_head)
- if total qty given, individual item qtys sum to total (only need to check this if invc_match_qty_ind is 'Y" on system_options)
- total merchandise cost on THEAD line matches sum of costs from TDETL lines (sum of unit cost *qty)

  Note: Keep a running sum of individual qtys/costs; if don't match at end of transaction, reject transaction and rollback to savepoint
- Credit notes cannot have both a reference invoice id and a reference RTV
- If reference invoice and reference RTV are null, then no terms are required. If they are included, set due date to later of system date and sup_invoice_date (use set_due_date).
- If Item is not given, must have Ref. Item.
- dates are valid dates, numbers are all numeric
- If terms not given, default to supplier terms (from sups table)
- terms must exist on terms table
- VAT code must exist on vat_codes (validate on THEAD, and TVATS lines—use a separate function validate_VAT for this and pass in the VAT code)
- Addr_key must exist in the addr table for the partner os supplier given, if not given, will default to primary address of partner or supplier
- Paid_ind needs to be Y or N, if not given will default to that on the sups table corresponding to the supplier. If only a partner is given, paid_ind will default to N.

validate_THEAD()

- Validate THEAD line variables as needed

validate_TVATS()

- Validate VAT code from TVATS line.

validate_TXREF()

- validate TXREF line variables as needed.

validate_TDETL()

- validate TDETL line variables as needed.
- If Ref. Item is given but Item isn't, get the transaction-level Item from the item_master table

validate_order(order_no)

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

validate_shipment(ship_no)

- Match ASN to shipments in RMS.

validate_carton(ship_no, carton_no)

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

validate_vatcode(vat_code)

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

validate_partner_type

- Check that given partner_type is legal.

validate_addr_key

- Check that given addr_key is legal.

get_supplier_terms

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

get_addr_key

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

get_paid_ind

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

set_due_date()

- Sets due date to later of system date and sup_inv_date.

validate_terms()

- Check that terms exists on terms table.

insert_header()

- Insert a record into the invc_head table.

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

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

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

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

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

Values to insert into invc_head should be as follows:

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

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

insert_detail()

Insert a record into the invc_detail table.

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

Values should be as follows:

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

ORIG_VAT_RATE	from file
COST_DSCRPNCY_IND	'N'
QTY_DSCRPNCY_IND	'N'
VAT_DSCRPNCY_IND	'N'
PROCESSED_IND	'N'
COMMENTS	(null)

insert_wksht()

Insert a record into the invc_match_wksht table.

Values should be as follows:

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

insert_order_xref(order_no)

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

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

insert_ship_xref(shipment)

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

insert_match_queue()

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

insert_non_merch()

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

insert_merch_vat()

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

write_to_reject()

- Write record to reject file (use linked list)

get_line()

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

reset_variables()

- Set counters back to zero.

final()

- Restart/recovery close
- Close files, remove temporary file.

I/O Specification

Input file format:

FHEAD (start of file)

THEAD (Invoice level info)

TVATS (Summary level –cost at a VAT rate)

TXREF (Summary level – ASN and Order numbers)

TDETL (items within an invoice)

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

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

If running ediupinv.pc with no ASN in the upload file, then the TSHIP record should leave the field blank.

TTAIL (marks end of invoice)

FTAIL (marks end of file)

An input file can contain multiple invoices.

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

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

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

Record Name	Field Name	Field Type	Default Value	Description	Comments
	Cross-reference RTV number	Number(6)		RTV order that a credit note is for (blank for an invoice)	
	Partner Type	Char(6)		BK=Bank AG=Agent FF=Freight Forwarder IM=Importer BR=Broker FA=Factory AP=Applicant CO=Consolidator CN=Consignee S1=Supplier hierarchy level 1 (e.g. manufacturer) S2=Supplier hierarchy level 2 (e.g. distributor) S3=Supplier hierarchy level 3 (e.g. wholesaler)	
	Partner ID	Char(10)		Credit note or non-merchandise invoice partner.	
	Supplier	Number(10)		Invoice supplier	
	Supplier invoice date	Char(14)		Invc_head.invc_date YYYYMMDDHH24MISS Date invoice was issued by supplier	
	Terms	Char(15)		Invc_head.terms If not given in file, take from supplier terms	

Record Name	Field Name	Field Type	Default Value	Description	Comments
	Terms discount percentage	Number(12)		Discount percent if invoice paid on time 4 implied decimal places (optional—if not in file, take from terms table)	
	Discount apply indicator	Char(1)		'Y'—discount HAS been applied to total_merch_cost 'N'—it has not	
	Discount apply non merch ind	Char(1)		'Y'—discount HAS been applied 'N'—it has not	
	Payment method	Char(6)		Invc_head.payment_method LC=letter of credit OA= Open Account WT= wire transfer (other values possible)	
	Currency code	Char(3)		Currency code of invoice currency	
	Exchange rate	Number(20)		From the invoice currency to the primary currency. 10 implied decimal places. Optional.	
	Total merchandise cost	Number(20)		Total invoice cost, including all items on invoice (4 implied decimal places) (NOT including tax) VAT EXCLUSIVE	
	Total quantity	Number(12)		Total quantity of items on invoice (4 implied decimal places)	
	Freight terms	Char(30)		Freight terms associated w/invoice	

Record Name	Field Name	Field Type	Default Value	Description	Comments
	Sales tax	Number(20)		Sales tax on invoice (4 implied decimal places) Optional	
	Sales tax VAT code	char(6)		VAT code for sales tax (optional)	
	Freight charges	Number(20)		Freight charges for invoice (4 implied decimal places) Optional	
	Freight charge VAT code	char(6)		VAT code for freight charges (optional)	
	Miscellaneous charge	Number(20)		4 implied decimal places Optional	
	Misc. charge VAT code	char(6)		VAT code for miscellaneous charge—optional	
	Address key	Number(6)		Address key for the supplier or partner. (optional)	
	Paid_ind	Char(1)		Paid index for the invoice. ‘Y’ or ‘N’. (optional)	
TVATS	File record descriptor	Char(5)	TVATS	Marks costs at VAT rate line	
	Line id	Char(10)		Sequential file line number	
	Transaction number	Number(10)		Sequential transaction number	
	VAT code	Char(6)		VAT code that applies to cost	
	Cost at this VAT code	Number(20)		4 implied decimal places	

Record Name	Field Name	Field Type	Default Value	Description	Comments
TXREF	File record descriptor	Char(5)	TXREF	Marks cross-reference info line	
	Line id	Char(10)		Sequential file line number	
	Transaction number (10)	Number(10)		Sequential transaction number	
	ASN number	Char(30)		Suppliers shipment number	ASN must be used to map to RMS shipments that are written to the invc_xref table.
	Order Number	Number(8)		RMS Order Number	
TDETL	File record descriptor	Char(5)	TDETL	Marks item info line	
	Line id	Char(10)		Sequential file line number	
	Transaction number	Number(10)		Sequential transaction number	
	Item	Char(25)		Item (optional but at least one of Item/Ref. Item must be present)	
	Ref Item	Char(25)		Ref. Item (optional, see item)	
	Original Invoice quantity	Number(12)		How many of this item on invoice (4 implied decimal places)	
	Original Unit cost	Number(20)		Unit cost of item as reported by invoice 4 implied decimal places	
	Original VAT rate	Number(20)		VAT rate for this item (10 implied decimal places)	
TSHIP	File record descriptor	Char(5)	TSHIP	Marks ASN/carton info line	
	Line id	Char(10)		Sequential file line number	

Record Name	Field Name	Field Type	Default Value	Description	Comments
	Transaction number	Number(10)		Sequential transaction number	
	ASN number	Char(30)		Vendor ASN number used to identify shipment	ASN must be used to map to RMS shipments that are written to the invc_match_wksht table.
	Carton number	Char(20)		Carton ID if exists	
TTAIL	File record descriptor	Char(5)	TTAIL	Marks end of transaction	
	Line id	Char(10)		Sequential file line number	
	Transaction number	Number(10)		Sequential transaction number	
	Transaction lines	Number(6)		Total number of TXREF, TDETL, TVATS, and TSHIP lines in transaction	
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file	
	Line id	Char(10)		Sequential file line number	
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL	

Technical Issues

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

Extra Info

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

General structure of process function:

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

posdnld.pc

Design Overview

The posdnld program is used to download pos_mods records created in the RMS to the store POS systems. This program has one output file which contains all records for all stores in a given run. This program uses the Retek standard file format FHEAD, FDETL, FTAIL.

Program Flow

Stored Procedures / Shared Modules (Maintainability)

pos_config_sql.check_item - Updates POS item configuration information that is downloaded to the stores by poscdnld.pc.

Input Specifications

All input comes from the pos_mods table. All columns of this table can be NULL with the exception of tran_type and store. Most columns should default to blank (spaces) with the exception of:

- new_price, new_multi_units, new_multi_units_retail, proportional_tare_pct and fixed_tare_value. These should default to zero (0).
- start_date, start_time and end_time. These should default to period.vdate + 1.

Output Specifications

Output File

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Type Definition	Char(4)	POSD	Identifies file as 'POS Download'
	File Create Date	Char(8)	Create date (vdate).	Current date, formatted to 'YYYYMMDD'.
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number ID(10)	Sequential number. Created by program.	ID of current line being created for output file.
	Location Number	Number(10)	Store	Contains the store location that has been affected by the transaction
	Update Type	Char(1)	Update type. Created by program.	Code used for client specific POS system. 1 - Transaction Types 1 & 2. 2 - Transaction Types 10 thru 18, 31 & 32, 50 thru 57, 59 thru 64. 3 - Transaction Types 21 & 22 4 - Transaction Types 25 & 26 0 - All other Transaction Types. These should never exist.
	Start Date	Char(8)	Start_date or vdate + 1 if NULL.	The effective date for the action determined by the transaction type of the record. Formatted to 'YYYYMMDD'.

Record Name	Field Name	Field Type	Default Value	Description
	Time	Char(6)	Start_time, End_time or start_date.	This field will be used in conjunction with starting a promotion (Transaction Type = 31). Start time will indicate the time of day that the promotion is scheduled to start. This field will also be used in conjunction with ending a promotion (Transaction Type = 32). Any other Transaction Type will use the time from the start_date column. Formatted to 'HH24MISS'.

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Type	Number(2)	Tran_type	Indicates the type of transaction to determine what Retek action is being sent down to the stores from the Retek pos_mods table. Valid values include: 01 - Add new transaction level item 02 - Add new lower than transaction level item 10 - Change Short Description of existing item 11 - Change Price of an existing item 12 - Change Description of an existing item 13 - Change Department/Class/Subclass of an existing item 16 - Put Item on Clearance 17 - Change existing item's Clearance Price 18 - Remove Item from Clearance and Reset 20 - Change in VAT rate 21 - Delete existing transaction level item 22 - Delete existing lower than transaction level item 25 - Change item's status 26 - Change item's taxable indicator 31 - Promotional item - Start maintenance 32 - Promotional item - End maintenance 50 - Change item's launch date 51 - Change item's quantity key options 52 - Change item's manual price entry options 53 - Change item's deposit code 54 - Change item's food stamp indicator 55 - Change item's WIC indicator 56 - Change item's proportional tare percent 57 - Change item's fixed tare value 58 - Change item's rewards eligible indicator 59- Change item's electronic marketing clubs 60 - Change item's return policy 61 - Change item's stop sale indicator 62 – Change item’s returnable indicator

Record Name	Field Name	Field Type	Default Value	Description
	Item Number ID	Char(25)	Item	This field identifies the unique alphanumeric value for the transaction level item. The ID number of a item from the Retek item_master table.
	Item Number Type	Char(6)	Item_number_type	This field identifies the type of the item number ID.
	Format ID	Char(1)	Format_id	This field identifies the type of format used if the item_number_type is 'VPLU'.
	Prefix	Number(2)	Prefix	This field identifies the prefix used if the item_number_type is 'VPLU'. In case of single digit prefix, the field will be right-justified with blank padding.
	Reference Item	Char(25)	Ref_item	This field identifies the unique alphanumeric value for an item one level below the transaction level item.
	Reference Item Number Type	Char(6)	Ref_Item_number_type	This field identifies the type of the ref item number ID.
	Reference Item Format ID	Char(1)	Ref_Format_id	This field identifies the type of format used if the ref item_number_type is 'VPLU'.
	Reference Item Prefix	Number(2)	Ref_Prefix	This field identifies the prefix used if the ref item_number_type is 'VPLU'. In case of single digit prefix, the field will be right-justified with blank padding.
	Item Short Description	Char(20)	Item_short_desc	Contains the short description associated with the item.
	Item Long Description	Char(100)	Item_long_desc	Contains the long description associated with the item.
	Department ID	Number(4)	Dept	Contains the item's associated department.
	Class ID	Number(4)	Class	Contains the item's associated class.
	Subclass ID	Number(4)	Subclass	Contains the item's associated subclass.
	New Price	Number(20)	New_price	Contains the new effective price in the selling unit of measure for an item when the transaction type identifies a change in price. Otherwise, the current retail price is used to populate this field. This field is stored in the local currency.

Record Name	Field Name	Field Type	Default Value	Description
	New Selling UOM	Char(4)	New_selling_UOM	Contains the new selling unit of measure for an item's single-unit retail.
	New Multi Units	Number(12)	New_multi_units	Contains the new number of units sold together for multi-unit pricing. This field is only filled when a multi-unit price change is being made.
	New Multi Units Retail	Number(20)	New_multi_units_retail	Contains the new price in the selling unit of measure for units sold together for multi-unit pricing. This field is only filled when a multi-unit price change is being made. This field is stored in the local currency.
	New Multi Selling UOM	Char(4)	New_multi_selling_UOM	Contains the new selling unit of measure for an item's multi-unit retail.
	Status	Char(1)	Status	Populates if tran_type for the item is 1(new item added) or 25 (change item status) or 26 (change taxable indicator). Contains the current status of the item at the store. Valid values are: A = Active I = Inactive D = Delete C = Discontinued
	Taxable Indicator	Char(1)	Taxable_ind	Populates if tran_type for the item is 1 (new item added) or 25 (change item status) or 26 (change taxable indicator). Indicates whether the item is taxable at the store. Valid values are 'Y' or 'N'.
	Promotion Number	Number(10)	Promotion	This field contains the number of the promotion for which the discount originated. This field, along with the Mix Match Number or Threshold Number is used to isolate a list of items that tie together with discount information.
	Mix Match Number	Number(10)	Mix_match_no	This field contains the number of the mix and match in a promotion for which the discount originated. This field, along with the promotion, is used to isolate a list of items which tie together with the mix and match discount information.

Record Name	Field Name	Field Type	Default Value	Description
	Mix Match Type	Char(1)	Mix_match_type	This field identifies which types of mix and match record this item belongs to. The item can either be a buy (exists on PROM_MIX_MATCH_BUY) or a get (exists on PROM_MIX_MATCH_GET) item. This field is only populated when the MIX_MATCH_NO is populated. Valid values are: B - Buy G - Get
	Threshold Number	Number(10)	Threshold_no	This field contains the number of the threshold in a promotion for which the discount originated. This field, along with the promotion, is used to isolate a list of items that tie together with discount information.
	Launch Date	Char(8)	Launch_date	Date that the item should first be sold at this location, formatted to 'YYYYMMDD'.
	Quantity Key Options	Char(6)	Qty_key_options	Determines whether the price can/should be entered manually on a POS for this item at the location. Valid values are in the code_type 'RPO'. Current values include 'R - required', 'P - Prohibited.
	Manual Price Entry	Char(6)	Manual_price_entry	Determines whether the price can/should be entered manually on a POS for this item at the location. Valid values are in the code_type 'RPO'. Current values include 'R - required', 'P - Prohibited', and 'O - Optional'.
	Deposit Code	Char(6)	Deposit_code	Indicates whether a deposit is associated with this item at the location. Valid values are in the code_type 'DEPO'. Additional values may be added or removed as needed. Deposits are not subtracted from the retail of an item uploaded to RMS, etc. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.
	Food Stamp Indicator	Char(1)	Food_stamp_ind	Indicates whether the item is approved for food stamps at the location.

Record Name	Field Name	Field Type	Default Value	Description
	WIC Indicator	Char(1)	Wic_ind	Indicates whether the item is approved for WIC at the location.
	Proportional Tare Percent	Number(12)	Proportional_tare_pct	Holds the value associated of the packaging in items sold by weight at the location. The proportional tare is the proportion of the total weight of a unit of an item that is packaging (i.e. if the tare item is bulk candy, this is the proportional of the total weight of one piece of candy that is the candy wrapper). The only processing RMS does involving the proportional tare percent is downloading it to the POS.
	Fixed Tare Value	Number(12)	Fixed_tare_value	Holds the value associated of the packaging in items sold by weight at the location. Fixed tare is the tare of the packaging used to (i.e. if the tare item is bulk candy, this is weight of the bag and twist tie). The only processing RMS does involving the fixed tare value is downloading it to the POS. Fixed tare is not subtracted from items sold by weight when sales are uploaded to RMS, etc. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.
	Fixed Tare UOM	Char(4)	Fixed_tare_uom	Holds the unit of measure value associated with the tare value. The only processing RMS does involving the proportional tare value and UOM is downloading it to the POS. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.
	Reward Eligible Indicator	Char(1)	Reward_eligible_ind	Holds whether the item is legally valid for various types of bonus point/award programs at the location.
	Elective Marketing Clubs	Char(6)	Elect_mtk_clubs	Holds the code that represents the marketing clubs to which the item belongs at the location. Valid values can belong to the code_type 'MTKC'. Additional values can be added or removed from the code type as needed

Record Name	Field Name	Field Type	Default Value	Description
	Return Policy	Char(6)	Return_pocily	Holds the return policy for the item at the location. Valid values for this field belong to the code_type 'RETP'.
	Stop Sale Indicator	Char(1)	Stop_sale_ind	Indicates that sale of the item should be stopped immediately at the location (i.e. in case of recall etc).
	Returnable Indicator	Char(1)	Returnable_ind	Indicates that the item is returnable at the location when equal to 'Y'es. Indicates that the item is not returnable at the location when equal to 'N'o.
	Refundable Indicator	Char(1)	Refundable_ind	Indicates that the item is refundable at the location when equal to 'Y'es. Indicates that the item is not refundable at the location when equal to 'N'o.
	Back Order Indicator	Char(1)	Back_order_ind	Indicates that the item is back orderable at the location when equal to 'Y'. Indicates that the item is not back orderable when equal to 'N'o.
	Vat Code	Char(6)		Indicates the VAT code used with this item.
	Vat Rate	Number(20, 10)		Indicates the VAT rate associated with this item and VAT code.
	Class Vat Indicator	Char(1)		Indicates whether or not the class VAT indicator is on or off for the class that this item exists in.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number ID(10)	Sequential number. Created by program.	ID of current line being created for output file.
	File Record Counter	Number ID(10)	Number of FDETL records. Created by program.	Number of records/transactions processed in current file (only records between head & tail)

Function Level Description

init - This function initializes restart/recovery for this program. It also retrieves system variables (period.vdate and vdate + 1), opens the output file and write the FHEAD record.

process - This function drives the processing of the program. It calls size_arrays() function to size the arrays used in this program and also, when done, it calls free_arrays() to release any memory it has been allocated. The driving cursor is opened and fetched here, which retrieves all the records from pos_mods where the pos_mods.store value is greater than zero.

If the Transaction Type is 31, then the time field returned by the cursor should be the start time, else if the Transaction Type is 32, then the time field should be the end time. If the Transaction Type is something else or if either the start time or end time is NULL, blanks should be used.

Once the records are fetched, if the Transaction Type of the record fetched is 1 or 21 then pos_config_check() is called. The write_rec() function is called to perform processing on all records fetched. Restart/Recovery and committing of records is also performed here.

final - This function will finish restart/recovery logic, write the FTAIL record and close the output.

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

free_arrays - This function frees the array allocated in size_arrays.

write_rec - This function will prepare records for insert into the output file and write them as FDETL records. The Transaction Type will determine the Update Type. If the Transaction Type is 1, 25 or 26 then the status and taxable_ind columns must be outputted, otherwise these should remain blank.

pos_config_check - This function will call the package pos_config_sql.check_item(). If the Transaction type is 1, then a status of 'A' will be passed in. If the Transaction Type is 21, then a status of 'D' will be passed in.

Scheduling Considerations

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program is run towards the end of the batch run when all pos_mods records have been created for the transaction day.

Pre-Processing: N/A

Post-Processing: prepost.pc - posdnld_post() – records in POS_MODS are truncated.

Threading Scheme: v_restart_store

Locking Strategy

None.

Restart/Recovery

Restart/recovery for this program is set up at the store/item or item level. Threading is done by store using the v_restart_store view to thread properly.

Performance Considerations

Both table and file restart/recovery must be used.

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

Security Considerations

Price changes for all stores are stored in a Unix file with the processes default permissions (umask). Care should be exercised so that this file cannot be tampered with.

Design Assumptions

Data columns required by a particular Transaction Type are filled in and correct.

Outstanding Design Issues

Issue Description	Priority	Issue Log Updated?
The columns pos_config_item.item and pos_merch_criteria.sku are of type number and have a length of 8. These columns are updated and referenced by the pos_config_sql.check_item() package function. These tables are then used by poscdnld.pc.	H	

Appendix

None.

Reclsdly.pc

Design Overview

Reclsdly will be modified to work with the new item dialog.

Function Level Description

Remove the following macros :

- NULL_PACK_TYPE.
- NULL_SYSTEM_IND.

Remove all EXEC SQL VARs.

Change the following variables :

- Os_restart_sku[NULL_SKU] to os_restart_item[NULL_ITEM].
- Os_sku[NULL_SKU] to os_item[NULL_ITEM].
- Os_system_ind[NULL_SYSTEM_IND] to os_item_num_type[NULL_ITEM_NUM_TYPE].

In the reclass_fetched_array, char (*ps_sku)[NULL_SKU] to char (*ps_item)[NULL_ITEM]. Also add variables to this fetch array for a pack_type, item_level, and tran_level.

Create variables os_pack_type, os_item_level, and os_tran_level.

The c_reclass cursor needs to something like the following :

```
SELECT ri.rowid,
    Ri.reclass_no,
                Ri.item,
    Item_level,
    Tran_level,
                to_dept,
                to_class,
                to_subclass,
                dept,
                class,
                subclass,
   pack_type,
   orderable_ind
   FROM v_restart_reclass,
          Reclass_item ri,
               Reclass_head,
```

```
                    Item_master
WHERE reclass_head.reclass_no = reclass_item.reclass_no
     AND reclass_head.reclass_date < vdate
     AND reclass_item.item = item_master.item
     AND item_master.item_level = 1
UNION
SELECT ri.rowid,
   Ri.reclass_no,
   im.item,
    Item_level,
    Tran_level,
   rh.to_dept,
   rh.to_class,
   rh.to_subclass,
   im.system_ind,
   im.dept,
   im.class,
   im.subclass,
   pack_type,
   orderable_ind
FROM v_restart_reclass,
   Reclass_item ri,
   Reclass_head rh,
   Item_master im
WHERE rh.reclass_no = ri.reclass_no
AND      rh.reclass_date <= VDATE
AND      ri.item = im.item_parent
AND      im.item_level = 2
UNION
SELECT ri.rowid,
   Ri.reclass_no,
   im.item,
    Item_level,
    Tran_level,
   rh.to_dept,
   rh.to_class,
   rh.to_subclass,
```

```
   im.system_ind,
   im.dept,
   im.class,
   im.subclass,
   pack_type,
   orderable_ind
FROM v_restart_reclass,
   Reclass_item ri,
   Reclass_head rh,
   Item_master im
WHERE rh.reclass_no = ri.reclass_no
AND      rh.reclass_date <= VDATE
AND      ri.item = im.item_grandparent
AND      im.item_level = 3
```

The cursor should retain the current threading and driver functionality.

Process()

The DELETE FROM reclass_sku needs to be changed to DELETE FROM reclass_item.

Check_domain_exists()

Process_sku()

Change name to Process_item(). The input parameter, char * is_sku, needs to be changed to char * is_item.

The c_get_pack_type cursor and all related functionality needs to be deleted.

The parameters passed to RECLASS_SQL.ITEM_PROCESS have changed.

```
      RECLASS_SQL.ITEM_PROCESS(ls_error_message,
                               :os_sku,
                               :os_item_level,
                               :os_tran_level,
                               TO_DATE(:ps_vdate,
'YYYYMMDD'),
                               :os_system_forecast_ind,
                               :lb_domain_exists,
                               :os_new_domain_id,
                               :os_old_dept,
                               :os_old_class,
                               :os_old_subclass,
                               :os_to_dept,
                               :os_to_class,
```

```
                                        :os_to_subclass)
```

Comment out the last part of this function, starting here :

```
if((*is_system_ind == 'S') || (*is_system_ind == 'f'))…
```

Process_security()

Local variable L_sku needs to be changed to L_item[NULL_ITEM];

C_sku cursor needs to be renamed to C_item, and to select ITEM from ITEM_MASTER where ITEM_GRANDPARENT = :os_item.

The DELETE FROM sec_group_prod_matrix needs to be changed to :

```
           DELETE FROM sec_group_prod_matrix
                                                WHERE level_1
= :os_item
                                                      OR
level_2 = :os_item
                      OR level_3 = :os_item
```

The DELETE FROM sec_user_prod_matrix needs to be changed to :

```
           DELETE FROM sec_user_prod_matrix
                      WHERE level_3 = :os_item
```

The parameters passed to LOC_PROD_SECURITY_SQL.INSERT_USER_SEC have changed.

```
LOC_PROD_SECURITY_SQL.INSERT_USER_SEC(ls_error_message,
                                      :os_to_dept,
                                      :os_to_class,
                                      :os_to_subclass,
                                      :os_sku,      /*
item level 1 */
                                      NULL,   /* item
level 2 */
                                      NULL)   /* item
level 3 */
```

Get_order_numbers()

The input parameter char * is_fetched_sku needs to be changed to char * is_fetched_item.

All code relating to this function must be commented out.

Call_insert_deal_sku_temp()

The input parameter char * is_sku needs to be changed to char * is_item.

All code relating to this function must be commented out.

Get_supplier_origin_id()

The input parameter char * is_sku needs to be changed to char * is_item.

All code relating to this function must be commented out.

Order_exists()

All code relating to this function must be commented out.

saexpach.pc

Design Overview

This module will post Store/day deposit totals to the SA_STORE_ACH table and bank deposit totals for a given day to a standard ACH format file. The ACH export deviates from the typical Sales Audit export in that store/days must be exported even though errors may have occurred for a given day or store (depending on the unit of work defined) and also the store/day does not need to be closed for the export to occur. The nature of the ACH process is such that as much money as possible must be sent as soon as possible to the consolidating bank. Any adjustments to the amount sent can be made via the sabnkach form.

Also, we are assuming that there is only one total to be exported for ACH per store/day.

Deposits for store/days that have not been 'F'ully loaded will not be transferred to the consolidating bank. After they are fully loaded, their deposits will be picked up by the next run of the program.

Furthermore, the program estimates a 0 for a store/day that is closed, for example due to a holiday. An example is shown below (Wednesday is a holiday):

	Mon	Tues	Wed	Thu	Fri
Estimated deposit for next day	5	0		10	
Adjustment to estimated deposit for this day		5		15	0
Exported at close	…	5		25	0
Actual deposit	…	10		15	10

In this example, we export only 5 (the adjustment) at close of Tuesday. The program is not run at close on Wednesday because it does not have a store_day_seq_no. Thus, on Thursday, the estimate for that day is 0 and the adjustment equals the actual. Also, on Thursday, we estimate that the total is going to be 10 and we export 25 at close of Thursday. Thus, the bank account should return to the minimum balance at this point.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	Yes	No
Sa_exported	No	Yes	No	No
Sa_store_ach	Yes	Yes	Yes	No
Sa_bank_ach	Yes	Yes	Yes	No

Table	Operations Performed			
	Select	Insert	Update	Delete
Sa_total	Yes	No	No	No
Sa_bank_store	Yes	No	No	No
Sa_store_day_write_lock	Yes	No	Yes	No
Sa_store_day_read_lock	Yes	No	No	No
Store	Yes	No	No	No
Partner	Yes	No	No	No

Background information – Quick Overview of the ACH process

ACH stands for Automated Clearing House and is a process by which funds can be transferred electronically from one account to another, possibly at a different financial institution. Instructions for each transaction are stored in a file, called an ACH file, which is then transferred across the ACH Network to be processed. This document provides only an overview of the process and will only describe points of interest for the saexpach program. It is beyond the scope of this document to provide the details of this process. Readers interested in knowing more about ACH should consult the 2000 ACH Rules published by the National ACH Association (NACHA).

There are 5 participants in an ACH transaction:

1 The originating company (called the Originator). The Originator is the entity requesting the transaction (i.e. this is where the transaction originates from).

2 The Originating Depository Financial Institution (ODFI).

3 The ACH Operator.

4 The Receiving Depository Financial Institution (RDFI).

5 The receiving company (called the Receiver).

It is important to note that the above description refers to direction of file transfers and not to direction of money flow.

Since the ReSA client has control over both the stores and the headquarters, the Originator can be either the former or the latter. To simplify the process, the headquarters will be the Originator, as this would require only one file to be produced, requesting money from each individual store. Figure 1 gives a pictorial overview.

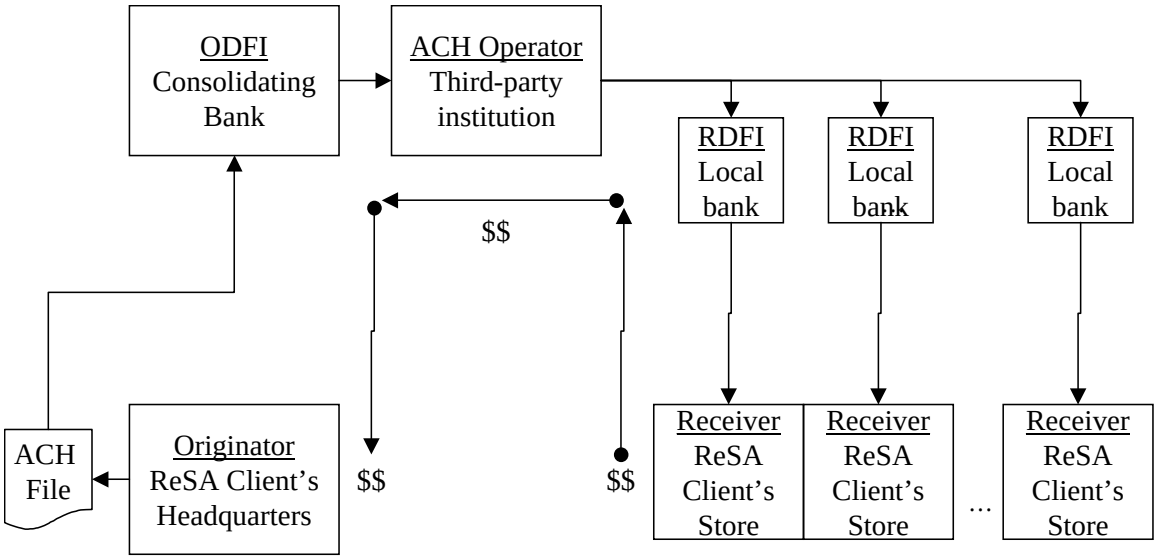


Figure 1: Overview of an ACH Network

The file that is produced at the Originator is sent to the ODFI which then routes it to the appropriate ACH operator(s). The latter will then contact the RDFI to request the money transfer.

In ACH jargon, the type of transaction that is being requested is a Cash Concentration and Disbursement (CCD). As of September 2000, however, transactions between institutions in different countries require a Corporate Cross-Border (CBR) Transaction. This program will meet this new requirement.

ACH is a US network of banks and therefore, this program should not be used for ACH look-alike networks outside the US, such as in Europe, as the file formats may be different. In other words, throughout this program, it is assumed that the country in which the consolidating bank is based is the United States.

Furthermore, all amounts in the ACH file are expected to be in US dollars (USD). Amounts for CBR transactions will have to be converted to USD.

Custom modifications can be made to this program such that output files that meet the requirements of other networks can be created. It is expected that the general structure of the program can be left unchanged and that only the functions that actually write the data out would have to change.

Data Security

The fact that this program automates the transfer of funds on behalf of the user makes it a likely target for electronic theft. It must be made clear that the responsibility of electronic protection lies with the users themselves. Retek does not provide any kind of encryption or authentication beyond what is provided by the operating system and the database management system. Retek does provide some tips and recommendation to users:

1 A specific user should probably be used to run the program. This user would be the only one (or one of a few) who has access to this program.

2 The umask for this user should be setup so as to prevent other users to read/write its files. This would ensure that when the output file is created, it will not be accessible to other users.

3 The appropriate permissions should be setup on the directory which holds the ACH files. The most restrictive decision would be to not allow any other user to view the contents of the directory.

4 The password to this user should be kept confidential.

5 A secure means of communication should be implemented for transferring the file from where it has been created to the ACH network. This may be done via encryption, or by copying the file to a disk and trusting the courier to deliver the files intact.

6 Retek assumes that the ACH network is secure.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA Totaling process: satotals and sarules.

Threading Scheme: N/A

Restart Recovery

This module is in two distinct parts, with two different logical units of work. Thus restart/recovery has to be implemented so that the first part does not get reprocessed in case the program is being restarted. Details on the implementation follow.

The first driving cursor in this module retrieves a store/day to generate ACH totals. Once the first cursor is complete, the second retrieves bank locations by account numbers.

The first Logical Unit of Work (LUW) is defined as a unique store/day combination. Records will be fetched, using the first driving cursor, in batches of commit_max_ctr, but processed one store/day at a time.

The first driving cursor will fetch all store/days that have been ‘F’ully Loaded, whose audit status is ‘A’udited, ‘H’Q Errors Pending or ‘S’tore Errors Pending and that are ready to be exported to ACH. Before processing starts, a write lock is obtained using get_lock (). This driving cursor only fetches store/days with a sa_export_log.status of SAES_R. After a store/day is processed, sa_export_log.status is set to SAES_P so that this store/day will not be selected again if the program is restarted. We commit using retek_force_commit after each store/day has been processed and sa_export_log updated, so as to release the lock.

In case a store/day could not be processed due to locking, then the store/day information is placed on a list (called locked store/day list) and the next store/day is processed. This list is kept in memory and is available only during processing. If the store for a store/day obtained from the first driving cursor, is on the locked store/day list, then this store/day cannot be processed. This is the case because there is a data dependency such that data from a particular store/day is dependent on data for the same store but at an earlier date. Thus, if a store/day cannot be processed, then subsequent store/days for the same store cannot be processed either. After the driving cursor returns no more data, the program attempts to process each store/day on the list two more times. If the store/day is still locked, then it is skipped entirely and a message is printed to the error log.

The second LUW is a bank account number. Again, records will be fetched in batches of commit_max_ctr. The second driving cursor cannot retrieve information by the LUW because it is possible for the store’s currency to be different from the local bank’s currency. In that case, a currency conversion is needed.

For each store/day, the query should retrieve the required ACH transfer. The latter is determined by adding the estimated deposit for the next day, the adjustment to the estimate for the current day and any manual adjustment to the estimate.

Since a store can be associated with different accounts at different banks, only accounts that are consolidated should be retrieved. Since it is possible for the local bank to be in a different country than the consolidating bank, the currency of the partner should also be fetched.

Since processing is dependent on the type of account at the RDFI, the account type should be fetched by this cursor.

Due to differences in transaction processing in cases when the bank is outside the US, the partner’s country should also be fetched. The results of the query should be sorted by partner country.

The results of the query should also be ordered by accounts.

Program Flow

Structure Chart

Please see the following document for the complete structure chart of the standard export for ReSA.

Functional Design – SA export.doc

Shared Modules

retek library functions:

- retek_init() – This function initializes restart/recovery.
- retek_close() – This function cleans up restart/recovery.
- retek_force_commit() – This function commits any change to the database.

Sales/Audit library functions (libresa):

- fetchVdate() – This function is used to get the vdate.
- fetchSysdate() – This function is used to get system date and time
- fetchStoreDayToBeExported() – This function contains the first driving cursor.
- get_lock() – This function is used to lock the store/day being processed.
- OraNumInit() – Initialize OraNum functions.
- OraNumAdd() – Add two large numbers passed in as strings.
- OraNumSub() –Subtract two large numbers passed in as strings.
- OraNumDiv() –Divide two large numbers passed in as strings.

Function Level Description

Init ()

- Initialize restart/recovery by calling restart_init().
- Get the vdate from the period table and the system time.
- Get the system level information: the sender id, the company id, the consolidating bank name, the consolidating routing number and the consolidating account number. These are on the sa_ach_info table.

Process ()

1 Get the next store/day to be processed (exported) by fetching from the first driving cursor.

2 Attempt to lock the store/day with a call to get_lock(). If this fails, write the store to a linked list (which contains all unprocessed store/days).

3 Skip to step 7 if the store of the store/day to be processed is for a store which is on the linked list.

4 Call the function postStoreACH() for the current store/day.

5 Set sa_export_log.status to SAES_P by calling setProcessed() for the current store/day, so that it will not be processed again in case of a restart.

6 Call retek_force_commit() to commit changes to the database and to release write lock.

7 Loop from beginning until the driving cursor returns no more data.

8 Call the function postBankSummaryTotals().

Final ()

- Clean up restart/recovery by calling retek_close().
- If the program has successfully processed the data, call retek_refresh_thread().

PostStoreACH ()

This function will generate and post an estimate and adjustment to the SA_STORE_ACH table for a given store/day. The function postStoreACH will accomplish the following processes in the following order:

Get the following pieces of data for the system code SYSE_ACH:

1 The total for the current business date,

2 Get the total for the following business date if it exists (by calling GetTomorrowTotal),

3 Call the function GetPastData() to get the totals for the past 4 weeks and for yesterday (that is, if the current store/day is for a Tuesday, then we want to get the totals for the past 4 Wednesdays and for yesterday). The latter pieces of data are obtained from the sa_store_ach table, by summing the estimate for a day with the adjustment for the same day.

4 Call the function GetPartnerInfo() to get partner type and partner id information.

- If there are more than one total for SYSE_ACH for a particular day, then this should be noted in the error log. We expect only one total per store/day. Only the first total returned by the function will be used, the rest will be ignored.
- Call the function CalculateData() to compute the estimate for the next business day and adjustment for the current store/day.
- Call the function PostStoreACHTable().

GetTomorrowTotal ()

This function attempts to get the total for the next business day to be used as the estimate. It returns a -1 if a fatal error occurred, a 0 if it was able to get the total. If a total was not found, the estimate is assigned to -1. If a store/day is never opened (i.e. a holiday), then a 0 is estimated for that store/day. Also, if a total is found, it should not be marked as exported.

GetPastData ()

This function retrieves totals for the same day of the week over the past 4 weeks and for the previous business day.

GetPartnerInfo ()

This function retrieves the bank partner (partner type and partner id) for the given store whose account is consolidated.

CalculateData ()

This function calculates the estimate for the next business day and adjustment for the current store/day.

- Find the estimate for the following business date using the following rules:
  - If the total for the following business date exists, then this is the estimate.
  - Otherwise, the estimate is the average for the data for the past 4 weeks. If we obtain data for fewer than 4 weeks, then we use the available data, but if we do not obtain any data, then we use the current day's total as the estimate.
  - If the estimate is a 0, then we use the current day's total as the estimate.
- Calculate the adjustment, which is the current date's total minus the estimate for the current date (which lies on the row for the previous day on the sa_store_ach table) and minus the manual adjustment for the current date (which lies on the row for the previous day on the sa_store_ach table).

ProcessLockedSD ()

This function processes any store/days that were not in the process() function due to locking. The list of such store/days is stored on the linked list.

1 Try to process the store/days that were not processed, that is, those that are on the linked list. Thus, for each store/day on the linked list, we try to obtain a lock. If one is not obtained, then we skip this store/day. If a lock is obtained, then we remove the store/day from the list.

2 Skip to step 5 if the store of the store/day to be processed is for a store which is on the linked list.

3 Call the function postStoreACH for the current store/day.

4 Set sa_export_log.status to SAES_P by calling setProcessed() for the current store/day, so that it will not be processed again in case of a restart.

5 Loop through steps 1 to 3, until each store/day in the list has been looked at.

6 Loop through steps 1 to 5 NUM_LOCK_RETRIES times. NUM_LOCK_RETRIES is by default 2. Thus, we try to attempt to process store/days that are locked two more times before giving up and skipping all locked store/days entirely.

7 For each store/day that was not processed, we write an error to the log.

PostStoreACHTable ()

This function inserts data into the sa_store_ach table. It updates if there is already an entry for the store, business date and partner.

- If there is no entry in the sa_store_ach table for the current store/day.
- Create an entry in the SA_STORE_ACH table with the current store_day_seq_no and the new estimate and adjustment deposits for the current store_day_seq_no.
- If there is an entry in the sa_store_ach table for the current store/day.
- Update the entry in sa_store_ach with the estimated deposit, and estimated deposit adjustment.

postBankSummaryTotals ()

This procedure will summarize the bank transaction totals to the ACH output file. Please see the section on I/O specifications for more information about the format of this file.

1 Open and fetch from the second driving cursor.

2 If any entries are to be made (i.e. there are results from the cursor), create ACH file and write file header by calling WriteACHFileHeader().

3 If the country of the bank just retrieved is different from the country of the previous bank, write a Batch Control Record by calling WriteACHBatchControl(), unless no Batch Header records have been written yet.

4 If the country of the bank just retrieved is different from the country of the previous bank, a new Batch Header record needs to be written. If the bank's country is the US, the WriteACHCCDBatchHeader() function should be called to write a Batch Header for CCD transactions. For all other countries, the WriteACHCBRBatchHeader() function should be called to write a Batch Header for CBR transactions.

5 If the store's currency is different from the bank's currency, do a conversion. Sum all the deposits for each bank account.

6 For each account at a bank in the US, create a CCD record in the file by calling WriteACHCCDEntry().

7 For each account at a bank outside the US, create a CBR record by calling WriteACHCBREntry().

- If the amount to be transferred is negative, the record should be skipped.
- If the account is a checking account, the transaction code to use is '27'.
- If the account is a savings account, the transaction code to use is '37'.

8 If the amount to be transferred is positive, call the function PostBankACHTable() to record the amount of the ACH entry, else do nothing.

9 Keep running totals for the current batch's total amount and the total ACH amounts.

10 Commit after pl_commit_max_ctr LUW have been processed. Redefine the SAVEPOINT after the commit because savepoints are lost after a commit.

11 Loop to step 3 until the cursor returns no data.

12 Write the ACH Batch Control record and the ACH File Control record

13 The ACH file format requires that the file size meet certain "block" requirements. See the section on the ACH file format for more details. Write the required number of "completion records" to meet the blocking requirements.

14 Mark all store/days *that were not locked* (i.e. those with a sa_export_log.status of SAES_P) as completed (SAES_E) in the sa_export_log.

postBankACHTable ()

This function inserts into the table sa_bank_ach. It updates if there already exist a record for the same partner and business date.

1 If an entry does not exist for the current bank and date in the sa_bank_ach table:

- Make an entry in the sa_bank_ach table for the current bank and account placing the sums of the store ACH amounts and adjustments in the ACH amount field (sa_bank_ach.ach_amt).

2 If an entry exists for the current bank and date in the sa_bank_ach table:

- Add the manual adjustment to the bank ACH deposit amount.
- Update the sa_bank_ach table with the bank ACH deposit amount (sa_bank_ach.ach_amt).

File Ouput functions

The functions WriteACHFileHeader(), WriteACHFileControl(), WriteACHCCDBatchHeader(), WriteACHCBRBatchHeader(), WriteACHBatchControl(), WriteACHCCDEntry(), WriteACHCBREntry(), WriteACHCBRAddendum() and WriteACHCompleteBlock() write the File Header Record, the File Control Record, the Batch Header Record for CCD transactions, the Batch Header Record for CBR transactions, the Batch Control Record, the CCD Entry Record, the CBR Entry Record, the CBR Addendum Record and the Completion Blocks, respectively. The WriteACHCBREntry() function should call the WriteACHCBRAddendum() function after writing to the file.

Linked list functions

The functions AddToList(), DeleteList(), GetNext() and RemoveFromList() provide means to manipulate and to retrieve data from the linked list which contains the store/days which were not processed due to locking issues.

MarkAllStoreDaysCompleted ()

This function sets the sa_export_log.status to SAES_E for store/days whose status is SAES_P. These are the store/days that have been exported. If a store/day was not exported, it will be picked up in the next run after it has met the conditions for export.

SetCurrencyDecimals ()

Given a currency code and an amount with 4 implicit decimals, this function will give out an amount with the appropriate number of decimals for the currency. For more details, see the BAI file format documentation. For example, there are two implicit decimals for the US Dollar, but none for the Japanese Yen. This function may need to be expanded because only a select few currencies are being processed. The last two decimal places are dropped for currencies that are not explicitly defined.

TruncateDec ()

This function truncates a number at the decimal point, i.e. “1234.56” becomes “1234”.

ACH File Structure

This section describes the structure of the output file of the saexpach.pc program. The output file conforms to the requirements imposed by the National Automated Clearing House Association (NACHA) and only the subset of records used by this program is outlined here. For more information on the other types of records and more information about the rules and regulations governing the ACH network, please refer to the "2000 ACH Rules" book published by NACHA.

The ACH file format is similar in many ways to Retek's flat file formats. The most distinctive differences are:

- The record type is a one-digit number rather than a five-digit character field.
- All records are 94 characters in length.
- Records are organized in blocks, where 1 block = 940 characters = 10 records.
- The File Control Record (similar to an FTAIL) contains a "Block Count" field which gives the total number of blocks in the file, including the File Header Record and the File Trailer Record. Records containing 9's must be used to complete the last block. For example, a file with 15 records will need 5 such records to give it a Block Count of 2. These "completion records" go at the end of the file.
- Transactions are organized in batches. Similar transactions make up one batch. In ReSA's case, the transactions are organized by the country of origin of the funds.

File Header Record

This record contains information about the characteristics of the file, such as sender and receiver, creation datetime, and so on.

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'1'	1	None
Priority Code	Reserved for future scheme for priority handling of files. '01' should be used.	'01'	2	None
Immediate Destination	Routing number of the consolidating bank. The field begins with a blank, followed by the 4-digit Federal Reserve Routing Symbol, the 4-digit ABA Institution Identifier, and the Check Digit.	SA_BANK_STORE . CONSOLIDATING _ROUTING_NO	10	None
Immediate Origin	A unique identification to determine the Originator. The ID and the format are supplied by the consolidating bank. Note that the user is responsible for the padding. That is, it is assumed that the data in the field will be exactly 10 characters wide.	SA_SYSTEM_OPT IONS. ACH_SENDER_ID	10	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
File Creation Date	Date when the file was created.	YYMMDD	6	None
File Creation Time	Time when the file was created.	HH24MM	4	None
File ID Modifier	This is used to differentiate files created on the same date and between the same Origin/Destination. Valid values are A through Z and 0 through 9. It is expected that only one file will be created per day, so a '0' should be used.	'0'	1	None
Record Size	Number of characters per record.	'094'	3	None
Blocking Factor	Number of physical records within a block.	'10'	2	None
Format Code	Reserved for future format variations. A '1' should be used.	'1'	1	None
Immediate Destination Name	The name of the consolidating bank.	SA_SYSTEM_OPTIONS. CONSOL_BANK_NAME	23	L/B
Immediate Origin Name	The name of the company.	COMPHEAD. CO_NAME	23	L/B
Reference Code	Any reference code. This is an optional field. ReSA will not populate this field as the create datetime should be enough to reference the data that was exported by comparing with SA_EXPORTED. EXP_DATETIME.	blanks	8	None

Note: This column described the justification and padding involved in the field being described. 'L' stands for left; 'R' stands for Right; 'B' stands for blank padding and '0' stands for 0 padding. None means that the field should be completely filled.

Batch Header Record for CCD transactions

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'5'	1	None
Service Class Code	This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used.	'225'	3	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
Company Name	The name of the company.	First 16 characters of COMPHEAD. CO_NAME	16	L/B
Company Discretionary Data	Any kind of data specific to the company. ReSA will not use this field	blanks	20	None
Company Identification	An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example, "1" IRS Employer ID Number "9" User Assigned Number. ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.	SA_SYSTEM_OPTIONS. COMPANY_ID	10	L/B
Standard Entry Class Code	This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CCD entries, this field should hold the value 'CCD'.	'CCD'	3	None
Company Entry Description	A short description from the Originator about the purpose of the entry.	'CONSOL.'	10	L/B
Company Descriptive Date	Optional field providing a date to the Receiver for descriptive purposes. ReSA will populate it with the next day's date in the YYMMDD format.	YYMMDD format of PERIOD.VDATE + 1	6	None
Effective Entry Date	The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date).	YYMMDD format of PERIOD.VDATE + 1	6	None
Settlement Date	This is inserted by receiving ACH Operator. ReSA will leave this blank.	blanks	3	None
Originator Status Code	This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution.	'1'	1	None
ODFI Identification	8-digit routing number of the ODFI.	First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO	8	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
Batch Number	The batch number.		7	R/0

Batch Header Record for CBR transactions

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'5'	1	None
Service Class Code	This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used.	'225'	3	None
Company Name	The name of the company.	First 16 characters of COMPHEAD. CO_NAME	16	L/B
Foreign Exchange Indicator	Code used to indicate the foreign exchange conversion methodology applied to a CBR entry. Retek uses the "Fixed-to-Variable" method to convert from the foreign currency into US dollars. Therefore, this field should be 'FV'.	'FV'	2	None
Foreign Exchange Reference Indicator	Code used to indicate the contents of the Foreign Exchange Reference field. The latter will contain the conversion rate used by Retek which means that the value should be '1'.	'1'	1	None
Foreign Exchange Reference	This should contain the foreign exchange rate used to compute the amounts in the CBR Entry Record. No decimal places are implied, that is, this field should contain the exact rate used.		15	L/B
ISO Destination Country Code	The country where the money is to be transferred to. Since ReSA assumes that the consolidating bank will be in the US, this should be 'US' – NOTE: verify that "US" is the correct ISO code for United States of America.	'US'	2	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
Company Identification	An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example, "1" IRS Employer ID Number "9" User Assigned Number. ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.	SA_SYSTEM_OPTIONS. COMPANY_ID	10	L/B
Standard Entry Class Code	This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CBR entries, this field should hold the value 'CBR'.	'CBR'	3	None
Company Entry Description	A short description from the Originator about the purpose of the entry.	'CONSOL.'	10	L/B
ISO Originating Currency Code	Currency code in which the funds are originating from. This must be the ISO code of the currency.	PARTNER. CURRENCY_CODE	3	None
ISO Destination Currency Code	Currency code in which the funds are to be received. This must be "USD".	'USD'	3	None
Effective Entry Date	The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date).	YYMMDD format of PERIOD.VDATE + 1	6	None
Settlement Date	This is inserted by receiving ACH Operator. ReSA will leave this blank.	blanks	3	None
Originator Status Code	This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution.	'1'	1	None
ODFI Identification	8-digit routing number of the ODFI.	First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO	8	None
Batch Number	The batch number. It is not expected that the file will have more than two batches.	'1' or '2'	7	R/0

CCD Entry Detail Record

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'6'	1	None
Transaction Code	Code used to identify the type of debit and credit. This is dependent on the type of account and on the direction of funds transfer. '27' – if the account is a checking account, '37' – if the account is a savings account.	'27' or '37'	2	None
RDFI Identification	8-digit routing number of the RDFI.	First 8 digits of SA_BANK_STORE . ROUTING_NO	8	None
Check Digit	This is the 9th digit from the routing number.	9th digit of SA_BANK_STORE . ROUTING_NO	1	None
DFI Account Number	The account number at the local bank.	SA_BANK_STORE . BANK_ACCT_NO	17	L/B
Amount	The amount involved in the transaction. This field is numeric only and the last two digits are automatically assumed to be decimals. ReSA amounts are stored as 20 digit numbers, with 4 for decimals. ReSA will truncate the last two digits of the amount and should the resulting amount be greater than 10 digits, this program will abort with an error. It is not expected that a client will send an ACH amount greater than US$100 million. The values for this are taken from the sa_store_ach table. The values from the columns today_adj_deposit_est, next_day_man_adj_deposit, and next_day_deposit_est are added up by business_date and then multiplied by 10000 and later divided by 100 to obtain a dollar amount.		10	R/0
Identification Number	Optional field containing a number used by Originator to insert its own number for tracing purposes. ReSA will not populate this field.	blanks	15	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
Receiving Company Name	Name of the local store.	STORE. STORE_NAME	22	L/B
Discretionary Data	Any kind of data specific to the transaction. ReSA will not use this field	blanks	2	None
Addenda Record Indicator	This field identifies whether this entry record contains addenda records. ReSA has no use for such records in CCD and will use the value of '0'	'0'	1	None
Trace Number	Used to uniquely identify each entry within a batch. The first 8 digits contain the routing number of the ODFI and the other 7 contains a sequence number. This sequence number should be ascending. Although the ACH specification does not require the numbers to be consecutive, ReSA will use consecutive numbers. Trace numbers should not be duplicated between batches.		15	None

CBR Entry Detail Record

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'6'	1	None
Transaction Code	Code used to identify the type of debit and credit. This is dependent on the type of account and on the direction of funds transfer. '27' – if the account is a checking account, '37' – if the account is a savings account.	'27' or '37'	2	None
RDFI Identification	8-digit routing number of the RDFI.	First 8 digits of SA_BANK_STORE . ROUTING_NO	8	None
Check Digit	This is the 9th digit from the routing number.	9th digit of SA_BANK_STORE . ROUTING_NO	1	None
DFI Account Number	The account number at the local bank.	SA_BANK_STORE . BANK_ACCT_NO	17	L/B
Amount	The amount involved in the transaction. This field is numeric only and the last two digits are automatically assumed to be decimals. This amount is in US dollars.		10	R/0
Identification Number	Optional field containing a number used by Originator to insert its own number for tracing purposes. ReSA will not populate this field.	blanks	15	None
Receiving Company Name	Name of the local store.	STORE. STORE_NAME	22	L/B
Discretionary Data	Any kind of data specific to the transaction. ReSA will not use this field	blanks	2	None
Addenda Record Indicator	This field identifies whether this entry record contains addenda records. Since CBR records must be followed by an addendum record, this value should be '1'.	'1'	1	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
Trace Number	Used to uniquely identify each entry within a batch. The first 8 digits contain the routing number of the ODFI and the other 7 contains a sequence number. This sequence number should be ascending. Although the ACH specification does not require the numbers to be consecutive, ReSA will use consecutive numbers. Trace numbers should not be duplicated between batches.		15	None

CBR Addendum Record

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'7'	1	None
Addenda Type Code	This code identifies the type of addendum record. CBR has only one type of Addenda Type Code: '01'.	'01'	2	None
Payment Related Information			80	L/B
Addenda Sequence Number	This is a sequence number denoting the position of each addendum record. The first record should always have a sequence number of 1 and subsequent records must be increasing and consecutive. ReSA will create only one addendum record for the CBR transaction.	'1'	4	R/0
Entry Detail Sequence Number	This is the sequence number part of the Trace Number of the entry record to which this addendum is referring.		7	R/0

Batch Control Record

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record	'8'	1	None
Service Class Code	This field identifies the general classification of dollar entries to be exchanged. Since money is being requested, this code should be 225 for "ACH Debits only".	'225'	3	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
Entry/Addenda Count	The number of entries and addenda in the batch. Basically, this is the number of records between the Batch Header Record and the Batch Control Record.		6	R/0
Entry Hash	This is the sum of the RDFI IDs in the detail records. It is the arithmetic sum of the 8-digit routing number. Overflow on the high order bits is ignored.		10	R/0
Total Debit Entry Dollar Amount in batch	These fields contain the accumulated debit and credit for the batch. This field is numeric only and the last two digits are automatically assumed to be decimals.		12	R/0
Total Credit Entry Dollar Amount in batch			12	R/0
Company Identification	An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example, “1” IRS Employer ID Number “9” User Assigned Number. ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.	SA_SYSTEM_OPTIONS. COMPANY_ID	10	L/B
Message Authentication Code (MAC)	The first 8 characters represent a code from the DES (Data Encryption Standard) algorithm. The remaining eleven characters are blanks. ReSA will not populate this field.	blanks	19	None
Reserved	Reserved	blanks	6	None
ODFI Identification	8-digit routing number of the ODFI.	First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO	8	None
Batch Number	The batch number.		7	R/0

File Control Record

This record contains summary information about the file to verify its integrity.

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'9'	1	None
Batch Count	The number of batches sent in the file.		6	R/0
Block Count	The number of physical blocks in the file, including both File Header and File Control Records. This is the ceiling of the number of records divided by the blocking factor, which is 10.	⌈(Number of records)/10⌉	6	R/0
Entry/Addenda Count	The number of entries and addenda in the file. Basically, this is the number of records between the Batch Header Record and the Batch Control Record.		8	R/0
Entry Hash	This is the sum of the Entry Hash fields on the Batch Control Records.		10	R/0
Total Debit Entry Dollar Amount in File	These fields contain the accumulated debit and credit for the file. This field is numeric only and the last two digits are automatically assumed to be decimals.		12	R/0
Total Credit Entry Dollar Amount in File			12	R/0
Reserved	This field should be filled with blanks. It is used to ensure that each record is of length 94.	blank	39	None

Technical Issues

Status	Issue	Resolution
Open	Tables and forms changes are required to ReSA to accommodate data that are currently not possible to store on the database. These are required before this program can be fully tested.	
Open	It is possible for an adjustment to be negative while the following day is a holiday, resulting in a negative ACH amount. ReSA expects these cases to be rare and will simply skip records with a negative ACH amount. It would be an enhancement to the product if the customer wants the system to estimate the next open day's deposit. Such entries will have to be bunched into a new batch with a different settlement date.	

Assumptions

1 This document assumes that the tables and forms changes are going to be applied accordingly.

2 It is assumed that the consolidating bank is US-based.

3 ReSA will assume that all country codes and all currency codes are ISO compliant.

saexpuar.pc

Design Overview

This module will post specified totals to the *Driscoll UAR* application. Using the typical export process, this module will loop through all available store day combinations. For each store day, all specified totals will be posted to their appropriate output files. All driving cursors will be handled by the libresa library functions.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	Yes	No
Sa_exported	No	Yes	No	No
Sa_exported_rev	Yes	No	No	No
Sa_total_head	Yes	No	No	No
Sa_total	Yes	No	No	No
Sa_bank_store	Yes	No	No	No
Sa_store_day_read_lock	Yes	Yes	No	Yes
Sa_store_value	Yes	No	No	No
Sa_sys_value	Yes	No	No	No
Sa_pos_value	Yes	No	No	No
Sa_hq_value	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA Totaling process.

Threading Scheme: N/A

Restart Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of pl_commit_max_ctr. Only two commits will be done. One to establish the store/day lock (this will be done by the package) and one at the end after a store/day has been completely processed.

Driving cursor 1

The libresa library function **fetchStoreDayToBeExportedLike** will drive the stores to be processed for any usage type starting with 'UAR'.

Driving Cursor 2

The libresa library function **getTotalLike** will drive the totals to be exported for any usage type starting with 'UAR'.

Program Flow

Please see the following document for the complete structure chart of the standard export for ReSA:

- Functional Design – SA export.doc

Shared Modules

libresa library functions:

- fetchStoreDayToBeExportedLike
- fetchSaSystemOptions
- fetchSysdate
- fetchStoredayErrorCount
- markStoreDayExported
- updateStoreDayExported
- markTotalExported
- getTotalLike
- get_lock
- OraNumInit
- OraNumAdd

Function Level Description

init ()

1 Call OraNumInit to initialize string numbers arithmetic operations.

2 Get the current system date from the library function fetchSysdate.

3 Get the unit-of-work by calling the library function fetchSaSystemOptions.

process ()

1 Loop through the libresa library function fetchStoreDayToBeExportedLike.

2 Attempt to obtain a read lock on the store/day with a call to get_lock. If this fails, go on to the next store/day and log the problem to the error log.

3 Call the function processStoreDay for the current store day.

4 Call the function markStoreDayExported.

5 Call the function retek_force_commit.

6 Loop from beginning until the return result of the function fetchStoreDayToBeExportedLike = 1.

final ()

1 Call the library function updateStoreDayExported to write any unwritten store days to the database.

2 Close output files.

3 Clean up any memory used.

4 Call the function retek_close.

```
addNewOutputFile (char is_usage_type,
                  char is_business_date,
                  char is_sysdate) returns integer
```

This function will generate a new output file for any new usage types retrieved from the getTotalLike function call. It will also add a new file item to a collection of any files currently being written to.

The file collection should contain the following items:

1 Usage type – the usage type returned by getTotalLike.

2 File name - <usage type>_<business date>_<system date>

3 File pointer – Pointer to the output file.

4 Wrote header – file header written indicator (0 – no, 1 – yes)

5 File sum – ongoing sum of each transaction in file.

getOutputFilePointer (char is_usage_type) returns integer

This function will retrieve the output file pointer for the usage type passed by checking to see if the usage type exists on the output file collection.

- If the usage type exists on the file collection, the item number for the collection is returned.
- If the usage type does not exist, the function returns -1.

```
writeStoreDayDetail (FILE *if_file_pointer,
                     char is_total_id,
                     char is_store,
                     char is_business_date,
                     char is_amount,
                     char is_total_seq_no,
                     char is_UAR_tran_code) returns integer
```

This function will write the current UAR total to the output file specified. Each field is separated by commas and surrounded by double quotes.

UAR Detail record:

Field	Description	Sales Audit value
1	Detail flag	1
2	Serial number	Store number
3	Amount	Total value
4	Transaction date	Transaction date
5	Transaction code	Mapped value: see the function getAdditionalInfo for detailed explanation.
6	User defined value 1	Total sequence number
7	User defined value 2	Nothing
8	User defined value 3	Nothing
9	User defined value 4	Nothing
10	User defined value 5	Nothing
11	User defined value 6	Nothing
12	User defined value 7	Nothing
13	User defined value 8	Nothing
14	User defined value 9	Nothing
15	User defined value 10	Nothing
16	State	Nothing
17	Account	Total identifier
18	End of line	\n

- All 18 fields should be concatenated together.

```
writeStoreDayHeader (FILE *if_file_pointer,
                     char is_total_id,
                     char is_business_date) returns integer
```

This function will write the header record for the current store day to the output file. Each field is separated by commas and surrounded by double quotes.

UAR Header record:

Field	Description	Sales Audit value
1	Header flag	0
2	Account number	Total identifier
3	Source	D
4	Transaction date	Transaction date
5	Organization number	Nothing
6	Format	UAR34
7	End of line	\n

All 7 fields should be concatenated together.

```
writeStoreDayFooter (FILE *if_file_pointer,
                     char is_amount) returns integer
```

This function will write the footer record for the current store day to the output file. Each field is separated by commas and surrounded by double quotes.

UAR Footer record:

Field	Description	Sales Audit value
1	Footer flag	9
2	Beginning balance	+00000000000000
3	Ending balance	“+” || the ongoing sum for the file.
4	Available balance	Nothing
5	End of line	\n

All 5 fields should be concatenated together.

```
CloseOutputFiles () returns integer
```

This function will loop through the output file collection and call the 'fclose' C function for each.

```
getOutputFileName (char is_usage_type,
                   char is_business_date,
                   char is_sysdate,
                   char os_filename) returns integer
```

This function will generate the unique file name for the total usage type passed. The filename will have the following structure:

```
is_usage_type || "_" || is_business_date || "_" ||
is_sysdate
```

```
getAdditionalInfo (char is_total_seq_no,
                   char is_ref_no1,
                   char os_total_id,
                   char os_UAR_tran_code) returns integer
```

This function retrieves both the total identifier and UAR transaction code for the current total sequence number. The UAR transaction code is retrieved as follows:

- If ref_no1 is not null
  - If ref_no1 maps to the SA_CONSTANTS table (ref_no1 = SA_CONSTANTS.CONSTANT_ID).
    - UAR transaction code = SA_CONSTANTS.CONSTANT_VALUE
  - If ref_no1 does not map to the SA_CONSTANTS table (ref_no1 != SA_CONSTANTS.CONSTANT_ID).
    - UAR transaction code = ref_no1
- If ref_no1 is null
  - UAR transaction code = total identifier

processStoreDay (char is_store_day,
                 char is_sysdate,
                 char is_business_date,
                 char is_store) returns integer

This function will process an entire store days UAR totals.

1 Loop through all UAR totals by calling the function **getTotalLike** until the function returns anything but zero.

2 Call the function getAdditionalInfo.

3 Determine if the output file exists by calling the function getOutputFilePointer.

4 If the pointer does not exist, call the function addNewOutputFile to create the new file.

5 Check to see if the header detail record has already been written for the current file by checking the current item on the output file collection.

6 If the head has not been written, call the function writeStoreDayHeader. Set the header indicator to 1 in the output file collection for the current item.

7 Write the current total to the current output file by calling the function writeStoreDayDetail.

8 Add the current total value to the running total sum in the output file collection for the current item.

9 Call the function markTotalExported.

I/O Specification

The UAR output file specifications are listed in this document by the functions that write the output:

- writeStoreDayHeader
- writeStoreDayDetail
- writeStoreDayFooter

saimpadj.pc

Design Overview

This module will post external system adjustments to the Sales Audit total value table. The module will be passed an input file name. The input files will be of a standard format as detailed in the "Interface File – SA Import Adj.doc" document.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_store_day	Yes	No	Yes	No
Sa_export_log	Yes	Yes	No	No
Sa_total	Yes	No	No	No
System_options	Yes	No	No	No
Sa_total_usage	Yes	No	No	No
Sa_hq_value	No	Yes	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: This module should be executed after the ReSA transaction import process, and before the ReSA totaling process.

Threading Scheme: N/A

Restart Recovery

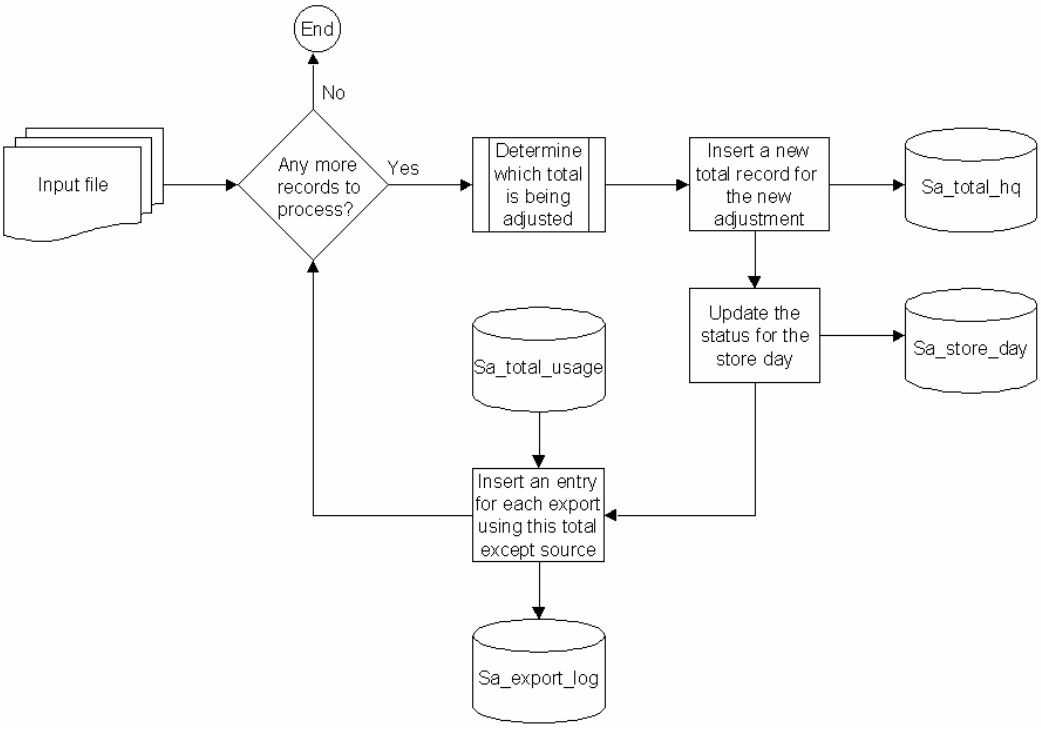
Logical Unit of Work

The logical unit of work for this module will be a parameterized number of records from the input file. Records will be processed until an internal counter has been reached and a commit will be executed. The Retek library (retek_2.h) function *retek_commit* will be called after each record processed. This function will keep track of the internal counter and commit the work once it has been reached.

Driving cursor

A standard *while* loop will read through all records in the input file. All file processing will use the new Retek library (retek_2.h) functions: retek_init, retek_get_record, retek_write_rej, retek_commit, retek_close.

Program Flow



The module Determine which total is being adjusted (as shown above) will depend on the source of the record being processed. If the record is from the UAR application the uar_source field will determine which total is being used. If the record is from the General ledger application the gl_account field will help determine which total is being adjusted.

The current status of the program passes through the “determine which total is being adjusted” section and just gives the program the total via the input file. The UAR and GL application information is still in question and will be modified when the best procedure is known.

Function Level Description

Init ()

- Read in the header from the input file.

Process ()

Loop through the input file adjustment records by calling the Retek library function **retek_get_record**.

1 Fetch the store_day_seq_no from the sa_total table.

2 Get a lock for the record by calling **get_Lock**.

   - Call the function processAdjustment. If processAdjustment returns a value <> 0 then call the Retek library function retek_write_rej in order to document unidentifiable adjustment records.

   Call the Retek library function retek_commit in order to commit on an internally defined number of records processed.

3 If lock could not be obtained, write an error to the error log.

Final ()

- Call retek_close
- Clean up any memory used.

processAdjustment (Char il_store_day_seq_no)

This function will complete one total adjustment by using the following processes:

1 Call the function determineTotal in order to identify the total identifier for the incoming adjustment.

2 Call the function postAdjustmentTotal in order to move the new adjustment amount into the current total table.

3 Call the function updateStoreDayStatus in order to ensure that the store day is re-calculated by the totaling process.

4 Call the function postExportLog in order to insert new records for all export requests except the source application of the adjustment.

5 Return 0 if successful adjustment posting.

6 Return 1 if unable to identify total.

7 Return –1 if a database error occurs.

determineTotal (o_totalIdentifier *char) as Long

This function will retrieve the total identifier for the current adjustment. This function will be the only location that will be checking specific source application data; any changes to the way applications handle exporting total adjustments will only have to be modified in this function. The following procedures will occur:

1 If the source of the current adjustment record is the General Ledger.

- Determine the total identifier by tracing the store, date and GL account number back to the GL mapping table (depending on which financial application is being used).

2 If the source of the current adjustment record is the UAR.

- Determine the total identifier by tracing the store, date and UAR source type back to the total definition table.

3 If the source of the current adjustment record is not UAR or General Ledger.

- Return the *Total* field in the o_totalIdentifier argument.

4 Return 0 if total identifier is found.

5 Return 1 if total identifier is not found.

6 Return –1 if a database error occurs.

postAdjustmentTotal () as Long

This function will insert the new total value record with the new data from the adjustment record. The following procedures will occur:

1 Fetch the max value_rev_no from the four value tables.

2 Find the max of the values returned

3 Insert the new total value into sa_hq_value table.

4 Return 0 if successful.

5 Return 1 if the total was not found in the total value table.

6 Return –1 if a database error occurred.

updateStoreDayStatus () as Long

This function will update the status field in the sa_store_day table. The following procedures will occur:

1 Update the sa_store_day table setting the audit status field to ‘Re-Totaling required’ (‘R’) and the audit changed date to the sysdate.

2 Return 0 if successful.

3 Return 1 if the store day record was not found in the sa_store_day table.

4 Return –1 if a database error occurred.

postExportLog () as Long

This function will make an entry in the export log (sa_export_log) for every application that requires export data for the current total being adjusted except the source application of the adjustment. The following procedures will occur:

1 Open a cursor to retrieve all export applications for the current total identifier:

```
Select   usage_type
From     sa_total_usage
Where    total = :ls_total
And              usage_type != :ls_data_source
```

2 Fetch the max seq_no from the sa_export_log table.

3 In a standard while loop

   a Insert a record into the export log (sa_export_log) for each usage type returned from the above query. Place the usage_type into the system_type field, and enter the status as 'R' for ready to export.

4 Return 0 if successful.

5 Return –1 if a database error occurred.

I/O Specification

The standard adjustment input format file can be found in the "Interface File – SA Import Adj.doc" document.

Technical Issues

Status	Issue	Resolution
Open	How do we trace back the total identifier with only the store number, date and UAR total specification (i.e. Lottery, etc.)?	The export to UAR will contain the total identifier, which can be passed back to ReSA when adjustments are sent for that total.

Sales audit import [saimptlog]

Purpose

The Batch Detailed Design is a thorough definition of a single batch program / module within one functional area. The documented information is derived from this functional area's Technical Design.

Objectives

This Batch Detailed Design must:

- Document specific functions for a single batch program,
- Enable project team review, validation and consensus regarding the individual batch program's scope,
- Document the batch program in preparation for and in response to prototyping, and
- Prepare for and provide a defined and documented framework in which to perform Development Phase activities.

Functional Area

Sales Audit import.

Module Affected

SAIMPTLOG (formerly saval.pc and saout.pc in 8.X)

- saimptlog.c
- saimptlog.h
- saimptlog_final.c
- saimptlog_init.c
- saimptlog_manval.c
- saimptlog_nexttsn.pc
- saimptlog_nextvhn.pc
- saimptlog_output.c
- saimptlog_uom.pc
- saimptlog_proto.h
- saimptlog_rtlog.c
- saimptlog_sqlldr.c*

The difference between SAIMPTLOG and SAIMPTLOGI is whether saimptlog_sqlldr.c or saimptlog_insert.pc is used. The former generates SQL*Loader files while the later performs actual inserts into the database.

- saimptlog_tdup.c
- saimptlog_tdup.h
- saimptlog_nextmtsn.pc
- saimptlog_nextesn.pc
- saimptlog_ccval.c
- saimptlog_ccval.h
- saimptlog_proto.h

SAIMPTLOGI

- saimptlog.c
- saimptlog.h
- saimptlog_final.c
- saimptlog_init.c
- saimptlog_insert.pc*
- saimptlog_manval.c
- saimptlog_nexttsn.pc
- saimptlog_nextvhn.pc
- saimptlog_output.c
- saimptlog_uom.pc
- saimptlog_proto.h
- saimptlog_rtlog.c
- saimptlog_tdup.c
- saimptlog_tdup.h
- saimptlog_nextmtsn.pc
- saimptlog_nextesn.pc
- saimptlog_ccval.c
- saimptlog_ccval.h
- saimptlog_proto.h

SAIMPTLOGFIN

- saimptlogfin.pc
- saimptlog_nexttbgsn.pc
- saimptlog.h

Design Overview

Importing POS data is a four or five-step process depending on whether saimptlogi or saimptlog is used. Saimptlog produces SQL*Loader files while saimptlogi does inserts directly into the database. Saimptlogi is meant for use in a trickle feed environment.

SAIMPTLOG	SAIMPTLOGI
SAGETREF must be run to generate the current reference files: • Items • Wastage • Sub-transaction level items • Primary variant relationships • Variable weight PLU • Store business day • Promotions • Code types • Error codes • Credit card validation • Store POS • Tender type • Merchant code types • Partner vendors • Supplier vendors • Employee ids These files are all used as input to SAIMPTLOG and SAIMPTLOGI. Since SAIMPTLOG and SAIMPTLOGI can be threaded, this boosts performance by limiting interaction with the database.	

SAIMPTLOG	SAIMPTLOGI
SAIMPTLOG is run against each POS file. SAIMPTLOG creates a write lock for store/day and than sets the data_status to loading until SAIMPTLOGFIN is executed. This generates distinct SQL*Loader files for that store/day for the sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally) sa_missing_tran tables. A Retek formatted voucher file is produced for processing by SAVOUCH. SAIMPTLOG may be threaded as long as the parallel executions do not include the same store/day.	SAIMPTLOGI is run against each POS file. SAIMPTLOGI creates a write lock for that store/day and then sets the data_status to loading until SAIMPTLOGFIN is executed. This inserts data into sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally) sa_missing_tran tables. A Retek formatted voucher file is produced for processing by SAVOUCH. SAIMPTLOGI may be threaded as long as the parallel executions do not include the same store/day.
SQL*Loader is executed to load the transaction tables from the files created by SAIMPTLOG. The store/day SQL*Loader files can be concatenated into a single file per table to optimize load times. Alternatively, multiple SQL*Loader files can be used as input to SQL*Loader. SQL*Loader may not be run in parallel with itself when loading a table. Header data (primary keys) must be loaded before ancillary data (foreign keys). This means that the sa_tran_head table must be loaded first; sa_tran_item before sa_tran_disc; and sa_customer before sa_cust_attrib. The remaining tables may be loaded in parallel.	
SAVOUCH is executed to load each of the Retek formatted voucher files. SAVOUCH may not be multiply threaded.	
SAIMPTLOGFIN is executed to populate the sa_balance_group table, cancel post voided transactions and vouchers, validate missing transactions, and to mark the import as either partially or fully complete loaded. SAIMPTLOGFIN may not be multiply threaded.	

This design document encompasses SAIMPTLOG, SAIMPTLOGI and SAIMPTLOGFIN.

SAIMPTLOG				
Table	Operations Performed			
	Select	Insert	Update	Delete
period	yes	no	no	no
store	yes	no	no	no
sa_system_options	yes	no	no	no
sa_store_day	yes	No	Yes	no
sa_store_day_write_lock	yes	yes	Yes	no

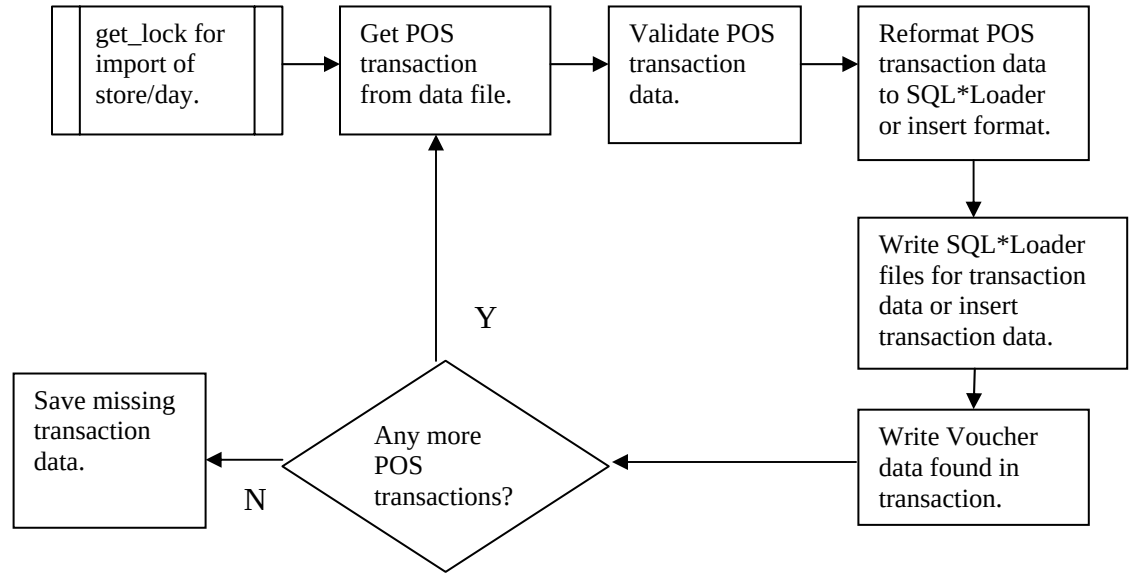
SAIMPTLOGI				
Table	Operations Performed			
	Select	Insert	Update	Delete
period	Yes	No	No	No
store	Yes	No	No	No
sa_system_options	Yes	No	No	No
sa_store_day	Yes	No	Yes	No
sa_store_day_write_lock	Yes	No	Yes	No
sa_tran_head	No	Yes	No	No
sa_customer	No	Yes	No	No
sa_cust_attrib	No	Yes	No	No
sa_tran_item	No	Yes	No	No
sa_tran_disc	No	Yes	No	No
sa_tran_tax	No	Yes	No	No
sa_tran_tender	No	Yes	No	No
sa_error	No	Yes	No	No
sa_missing_tran	No	Yes	No	No

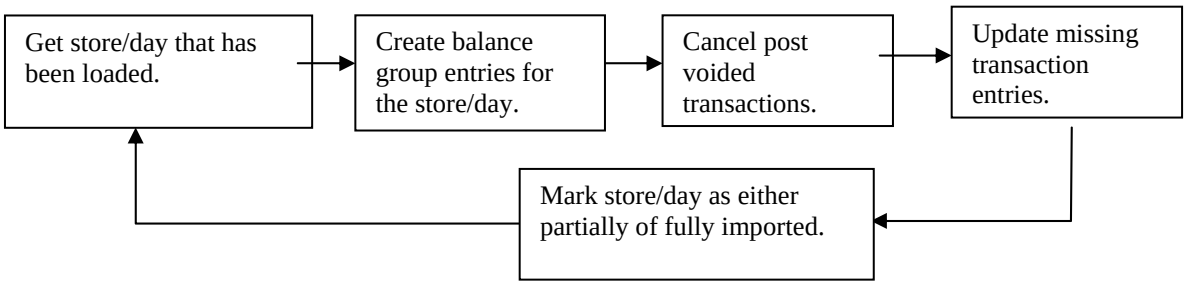
SAIMPTLOGFIN				
Table	Operations Performed			
	Select	Insert	Update	Delete
period	yes	no	No	no
store	yes	no	No	no
sa_system_options	yes	no	No	no
sa_store_day	yes	no	yes	no

SAIMPTLOGFIN				
Table	Operations Performed			
	Select	Insert	Update	Delete
sa_store_day_write_lock	yes	no	Yes	No
sa_import_log	yes	no	yes	no
sa_balance_group	Yes	Yes	No	No
sa_tran_head	Yes	No	Yes	Yes
sa_customer	Yes	No	No	Yes
sa_cust_attrib	Yes	No	No	Yes
sa_tran_item	Yes	No	No	Yes
sa_tran_disc	Yes	No	No	Yes
sa_tran_tax	Yes	No	No	Yes
sa_tran_tender	Yes	No	No	Yes
sa_error	Yes	No	No	Yes
sa_missing_tran	Yes	No	No	Yes
sa_tran_head_rev	No	Yes	No	No
sa_tran_item_rev	No	Yes	No	No
sa_tran_disc_rev	No	Yes	No	No
sa_tran_tax_rev	No	Yes	No	No
sa_tran_tender_rev	No	Yes	No	No
sa_error_rev	No	Yes	No	No

Program Flow

SAIMPTLOG and SAIMPTLOGI





SAIMTLOGFIN

Function Level Description

SAIMPTLOG and SAIMPTLOGI

As noted earlier, the difference between SAIMPTLOG and SAIMPTLOGI is whether saimptlog_sqlldr.c or saimptlog_insert.pc is used. Routines flagged with a ‡ denote that they exist in both of these modules and that behavior will depend on which module is used.

main() [saimptlog.c]

This should be the standard Retek main. Call LOGON to connect to the Sales Audit database. Call Init to initialize data structures and output file handles. Call Process to translate the RTLOG POS data into either the SQL*Loader files or to insert the data, and to produce a Retek formatted file for vouchers. Call Final to close file handles and to generally clean up.

Process() [saimptlog.c]

For each transaction in the POS RTLOG file, call getNextTran to read in the data and process it.

For each transaction, call WrOutputData‡ and writeSAVoucherData to write the voucher transaction data to a temporary file.

Init() [saimptlog_init.c]

Call retek_init to initialize threading.

Get the system options by calling fetchSaSystemOptions.

Get the current system data (SYSDATE) by calling fetchSysDate. This is used later to validate the dates in the POS RTLOGs.

Initialize the RTLOG file parser by calling InitInputData.

Load the item data generated by SAGETREF by calling item_loadfile.

Load the sub-transaction level item data generated by SAGETREF by calling ref_item_loadfile.

Load the variable weight PLU data generated by SAGETREF by calling vupc_loadfile.

Load the primary variant data generated by SAGETREF by calling primvariant_loadfile.

Load the store/day data generated by SAGETREF by calling store_day_loadfile.

Load the wastage data generated by SAGETREF by calling waste_loadfile.

Load the promotion data generated by SAGETREF by calling prom_loadfile.

Load the code type data generated by SAGETREF by calling code_loadfile.

Load the error data generated by SAGETREF by calling error_loadfile.

Load the store POS data generated by SAGETREF by calling storepos_loadfile.

Load the tender type group and ID data generated by SAGETREF by calling tendertype_loadfile.

Load the merchant code data generated by SAGETREF by calling merchcode_loadfile.

Load the partner vendor data generated by SAGETREF by calling partner_loadfile.

Load the supplier vendor data generated by SAGETREF by calling supplier_loadfile.

Load the employee data generated by SAGETREF by calling employee_loadfile.

Initialize transaction output processing by calling InitOutputData‡.

Initialize voucher output processing by calling openSAVoucher.

Initialize Oracle number arithmetic by calling OraNumInit.

If either of these last 2 fail, call InitOutputClean‡.

Final() [saimptlog_final.c]

If the system option check_dup_miss_tran is enabled, than call tdup_savedata to keep track of missing transaction numbers between invocations of SAIMPTLOG or SAIMPTLOGI and call tdup_misstran to create the SQL*Loader file for the sa_missing_tran table.

Call CreateTermRecords‡ to mark the end of the data and than call WrOutputData‡ to write them to the temporary files.

Terminate the RTLOG file parser by calling FinalInputData.

Call FinalOutputData‡ to finish any pending output processing.

Call closeSAVoucher to close and rename the voucher file.

Call retek_close to perform program status record keeping.

Call retek_refresh_thread to refresh the thread that was used during this execution so that it can be reused.

InitInputData() [saimptlog_rtlog.c]

Open the POS RTLOG file for reading.

Open a bad transaction file for writing.

Initialize the POS RTLOG transaction parser.

getNextTran() [saimptlog_rtlog.c]

This function reads in each transaction (by calling getRTLRec for each transaction) and validates each record contained within it (by calling procRTLFHead, procRTLFTail, procRTLTHead, procRTLTTail, procRTLTCust, procRTLCAtt, procRTLTItem, procRTLIDisc, procRTLTTax and procRTLTTend as appropriate). To simplify processing, the FHEAD and FTAIL records are treated as individual transactions. The function rtFind is used to determine the type of the record read.

Some record types will require some extra processing:

FHEAD – Need to retain the location (store) and business date for later validations. Also, the transaction structures must be reset by calling resetTran. Write out a FHEAD record to the voucher file by calling writeSAVoucherFHEAD.

FTAIL - Write out a FTAIL record to the voucher file by calling writeSAVoucherFTAIL.

TTAIL – Call chkTranFormat to check for format and data problems. Call chkTranTailCount to validate the number of records found in the transaction. Call tdup_addtran to check for duplicate transactions and to keep track of possible missing transactions, except when the transaction is a 'TOTAL' and its tran_no is blank. Call reformatTran to format the RTLOG transaction data into SQL*Loader flat file format. If any errors occur, call WrBadTran to write the failing transaction to the bad transaction file and call resetTran to reinitialize the RTLOG parser for the next transaction.

FinalInputData() [saimptlog_rtlog.c]

Close the POS RTLOG file.

Close the bad transaction file.

getRTLRec() [saimptlog_rtlog.c]

Read and return one record from the POS RTLOG file.

rtFind() [saimptlog_rtlog.c]

Return the type of the record that is passed in (i.e. THEAD, TCUST, TITEM, etc).

procRTLFHead() [saimptlog_rtlog.c]

Check that this is the first record in the POS RTLOG file. Validate the business date of the data. Call storeday_lookup to verify that there is a sa_import_log entry. If an entry is not found, generate an error and do not load any data. Call get_lock to lock the store/day for importing. If a lock is not obtained, keep trying a set number of times. Call updateDataStatus to set the store/day's data_status to loading (SADS_L). If missing transactions are being tracked, call storepos_lookup to get the transaction number starting and ending values and call tdup_loaddata to load into memory past transaction number ranges for the current store/day. Note that the maximum transaction number allowed is 2147483647.

procRTLFTail() [saimptlog_rtlog.c]

Process a FTAIL record, ensuring that it is the last record in the POS RTLOG file. The record count in the FTAIL record is checked against the number of records processed, if these do not match then records are missing and we should abort.

procRTLTHead() [saimptlog_rtlog.c]

Validate that the THEAD record is located within a valid position in the POS RTLOG file, after an FHEAD or TTAIL record.

Initialize the sale and tender transaction totals to 0.

procRTLTTail() [saimptlog_rtlog.c]

Validate that the TTAIL record is located within a valid position in the POS RTLOG file, after a TITEM, IDISC, TTAX, TTEND, TCUST or CATT record.

procRTLTCust() [saimptlog_rtlog.c]

Validate that the TCUST record is located within a transaction in the POS RTLOG file.

procRTLCAtt() [saimptlog_rtlog.c]

Validate that the CATT record is located within a transaction following either a TCUST or CATT record in the POS RTLOG file.

procRTLTItem() [saimptlog_rtlog.c]

Validate that the TITEM record is located within a transaction in the POS RTLOG file.

Convert selling unit of measure to standard, necessary.

Check if item number type is variable weight PLU. If so, decode it.

Add the quantity * the unit retail amount to the sale transaction total.

procRTLIDisc() [saimptlog_rtlog.c]

Validate that the IDISC record is located within a valid position in the POS RTLOG file, after either a TITEM or IDISC record.

Convert selling unit of measure to standard, if necessary.

Subtract the quantity * the unit discount amount from the sale transaction total.

procRTLTTax() [saimptlog_rtlog.c]

Validate that the TTAX record is located within a transaction in the POS RTLOG file.

Add the tax amount to the sale transaction total.

procRTLTTend() [saimptlog_rtlog.c]

Validate that the TTEND record is located within a transaction in the POS RTLOG file.

Add the tender amount to the tender transaction total.

resetTran() [saimptlog_rtlog.c]

Reinitialize the transaction structures.

chkTranTailCount() [saimptlog_rtlog.c]

Checks the counters in a transaction’s TTAIL record and produce an error if this figure does not match the actual number of records processed for this transaction.

chkTranFormat() [saimptlog_rtlog.c]

Checks the current transaction format and content. Produces an error if more than one TCUST record is found, an IDISC record does not correspond to a TITEM record, an unknown record type is encountered or the THEAD or TTAIL records are missing from the transaction.

For each record in the transaction call rrchk to look for invalid characters in the record.

Call trat_lookup to get the transaction type and then validate that type with the number of records within the transaction.

rrchk() [saimptlog_rtlog.c]

Make sure that there are no embedded null, tab, carriage return or new line characters in the record passed in.

WrBadTran() [saimptlog_rtlog.c]

Writes an erroneously formatted transaction out to the reject file for correction by an auditor. These transactions do not contain enough information to be loaded to the Sales Audit tables.

reformatTran() [saimptlog_rtlog.c]

Validate and reformat the data within the transaction into the SQL*Loader flat file format (SAIMPTLOG) or insert the data into the database (SAIMPTLOGI). This is accomplished by calling routines that know the validations and formats for each tables SQL*Loader control file or insert statements. Start by calling resetFmt‡ to initialize the formatting routines. The validation routines are mvSATHead, mvSATCust, mvSACAtt, mvSATItem, mvSAIDisc, mvSATTax and mvSATTend. The reformatting routines are fmtSATranHead‡, fmtSACustomer‡, fmtSACustAttrib‡, fmtSATranItem‡, fmtSATranDisc‡, fmtSATranTax‡ and fmtSATranTend‡. If there are any errors that prevent loading this transaction into the database, call abortFmt‡. If there are correctable errors, call saErrorSATHead‡ for THEAD, TCUST and CATT records, call saErrorSATItem‡, saErrorSATDisc‡, saErrorSATTax‡ or saErrorSATTend‡ for the other record types.

If the transaction type is TRAT_SALE, TRAT_RETURN or TRAT_EEXCH, than check that the transaction balances by comparing the sale and tender transaction totals. Generate an error if they do not match.

updateDataStatus() [saimptlog_datastat.pc]

If the data status for this store/day is ready to be loaded (SADS_R), loading (SADS_L) or partially loaded (SADS_P) than update it to loading (SADS_L) and commit the change.

mvSATHead() [saimptlog_manval.c]

Ensure that the transaction date and time has a valid value.

Ensure that, if they exist and sa_system_options.auto_validate_tran_employee_id is YSNO_Y, the cashier and salesperson ids are valid by calling employee_lookup.

Ensure that the transaction type has a valid value (code_type of TRAT) by calling code_lookup.

Ensure that, if the balancing level is register (SABL_R) or store.tran_no_generated is register (STRG_R), then the register field is populated, and that if the balancing level is cashier (SABL_C), then the cashier field is populated.

Ensure that the transaction number exists for all transaction types except TRAT_DCLOSE and TRAT_TOTAL. If transaction number exists, make sure that it is numeric.

Ensure that the sub transaction type has a valid value if present (code_type of TRAS) by calling code_lookup.

Ensure that the reason code has a valid value if present (code_type of REAC) by calling code_lookup.

If the transaction type is TRAT_PAIDIN, ensure that a reason code is present.

If the transaction type is TRAT_PAIDOU:

If the sub transaction type is TRAS_MV or TRAS_EV, then validate the reason code by calling merchcode_lookup, else validate the reason code by calling code_lookup.

Ensure that the vendor number field is not empty.

If the sub transaction type is TRAS_MV then validate the vendor number against the suppliers by calling supplier_lookup.

Else if the sub transaction type is TRAS_EV then validate the vendor number against the partners by calling partner_lookup.

Else we do not validate.

If the sub transaction type is TRAS_MV or TRAS_EV then ensure that at least one of the vendor invoice number, payment reference number and proof of delivery number fields are present.

Else we do not validate.

If the transaction type is TRAT_TOTAL, ensure that ref_no1 and value are not empty.

Ensure that the value has a valid numeric value if present.

Return TRUE if all validations pass, else return FALSE.

mvSATCust() [saimptlog_manval.c]

Ensure that the customer ID has a value.

Ensure that the customer ID type has a valid value (code_type of CIDT) by calling code_lookup.

Ensure that the customers birthdate has a valid value if present.

Return TRUE if all validations pass, else return FALSE.

mvSACAtt() [saimptlog_manval.c]

Ensure that the customer attribute type has a valid value (code_type of SACA) by calling code_lookup.

Ensure that the customer attribute value has a valid value (code_type of attribute type) by calling code_lookup.

Return TRUE if all validations pass, else return FALSE.

mvSATItem() [saimptlog_manval.c]

Ensure that the item status has a valid value (code_type of SASI) by calling code_lookup. Also, if the tran_type is ‘SALE’, ‘RETURN‘ or ‘EEXCH’, then the only valid values are SASI_S, SASI_ R, and SASI_V. If the item status is SASI_S than the quantity sign must be SIGN_P. If the item status is SASI_R than the quantity sign must be SIGN_N.

Ensure that the item type has a valid value (code_type of SAIT) by calling code_lookup.

Ensure that the item, sub-transaction level item, or voucher number has a valid value depending on what the item type says should be present.

Ensure that the department, class, sub class and system indicator are valid if present.

Ensure that the quantity has a valid numeric value.

Ensure that the unit retail amount has a valid numeric value.

Ensure that the override reason code has a valid value (code_type of ORRC) by calling code_lookup if present.

Ensure that the original unit retail value has a valid numeric value if there is an override reason code.

Ensure that the tax indicator has a valid value (code_type of YSNO) by calling code_lookup. If the value is invalid, then an error is flagged and the value is defaulted to YSNO_Y.

Ensure that the item swiped indicator has a valid value (code_type of YSNO) by calling code_lookup. If the value is invalid, then an error is flagged and the value is defaulted to YSNO_Y.

Ensure that the return reason code has a valid value (code_type SARR) by calling code_lookup if present and the item status is SASI_R.

Ensure that, if it exists and sa_system_options.auto_validate_tran_employee_id is YSNO_Y, the salesperson id is valid by calling employee_lookup.

Ensure that if an expiration date exists, that it is valid.

Return TRUE if all validations pass, else return FALSE.

mvSAIDisc() [saimptlog_manval.c]

Ensure that the RMS promotion number has a valid value (code_type of PRMT) by calling code_lookup.

Ensure that the promotion has a valid value if present by calling prom_lookup. Valid values are PRST_A, PRST_E and PRST_M.

Ensure that the discount type has a valid value (code_type of SADT) by calling code_lookup.

Ensure that the quantity has a valid numeric value.

Ensure that the unit discount amount has a valid numeric value.

If the discount type is Coupon than ensure that the coupon number is present.

Return TRUE if all validations pass, else return FALSE.

mvSATTax() [saimptlog_manval.c]

Ensure that the tax code has a valid value (code_type of TAXC) by calling code_lookup.

Ensure that the tax amount has a valid numeric value.

Return TRUE if all validations pass, else return FALSE.

mvSATTend() [saimptlog_manval.c]

Ensure that the tender type group has a valid value (code_type of TENT) by calling code_lookup.

Ensure that the tender type ID has a valid value by calling tendertype_lookup.

Ensure that the tender amount has a valid numeric value.

If the tender type group is TENT_CCARD or TENT_DCARD than:

Ensure that the credit card number and expiration date are valid by calling ccval. The expiration date may be an empty field. If it is, no validation will be performed and there is no check as to whether the credit card has expired.

Ensure that the credit card authorization source if present has a valid value (code_type of CCAS) by calling code_lookup.

Ensure that the credit card cardholder verification if present has a valid value (code_type of CCVF) by calling code_lookup.

Ensure that the credit card entry mode if present has a valid value (code_type of CCEM) by calling code_lookup.

Ensure that the credit card special condition if present has a valid value (code_type of CCSC) by calling code_lookup.

If the tender type group is Coupon than ensure that the coupon number is present.

Return TRUE if all validations pass, else return FALSE.

nextTranSeqNo() [saimptlog_nexttsn.c]

Gets the next free header sequence number for use. This routine goes and gets a block of numbers when starting, and parcels them out as needed. Once they are all used up, another block is gotten.

tdup_savedata() [saimptlog_tdup.c]

Writes out what is currently known about transaction numbers for the current store/day.

tdup_misstran() [saimptlog_tdup.c]

Writes the entries for the sa_missing_tran table by calling fmtSAMissTran‡.

The sa_missing_tran.status column will be filled in with SAMS_M.

tdup_loaddata() [saimptlog_tdup.c]

Loads the data file of transaction number past ranges.

tdup_addtran() [saimptlog_tdup.c]

Adds a transaction number to the list of numbers encountered. If store.tran_no_generated is SRTG_S, than the transaction number must be unique to the store. If store.tran_no_generated is SRTG_R, than the transaction number must be unique to the store and register.

openSAVoucher() [saimptlog_output.c]

Generate a temporary filename for the voucher data and open it for writing.

closeSAVoucher() [saimptlog_output.c]

Close the voucher data file. If the RTLOG has been successfully processed, rename the temporary filename to a permanent name, else remove the temporary file.

writeSAVoucherFHEAD() [saimptlog_output.c]

Format and writes a FHEAD record to the voucher file.

writeSAVoucherFTAIL() [saimptlog_output.c]

Format and writes a FTAIL record to the voucher file.

writeSAVoucherData() [saimptlog_output.c]

If the current transaction type is a sale (SALE), or a return (RETURN) and the TITEM records contain a voucher number, then reformat the TITEM records into a sold voucher data by calling WrSoldSAVoucher. However, if the item was voided (i.e. for the same transaction, there is an item with status 'V' for the voucher), then do not call the function.

If the current transaction type is a sale (SALE), a paid in (PAIDIN), a return (RETURN) or paid out (PAIDOU), and the tender type group is a voucher (VOUCH) then:

- if the sign of the tender amount is positive, then reformat the TTEND records into an issued voucher data by calling WrIssuedSAVoucher
- else, if the sign of the tender amount is negative, then reformat the TTEND records into am issued voucher data by calling WrIssuedSAVoucher.

Note: It is not possible to return a voucher).

WrSoldSAVoucher() [saimptlog_output.c]

Format and write a sold voucher record to the voucher file.

In addition to the fields that are currently output in this function, information about the customer who purchased the gift certificate is required in the new iss_cust fields. This information can be copied directly from the RTLTCust record associated with the transaction being processed. The new recipient fields (name, state and country) will be stored in the RTLTItem record reference number fields for the Sale of a gift certificate. These values provide details on the intended receiver for a gift certificate at the time of sale. This might not be provided by every POS system, in which case they would be null. Expiration date will also be stored on the RTLTItem record and should be populated; it may also be null.

Source	Target
RTLTCust.name	SA_VOUCHER.iss_cust_name
RTLTCust.addr1	SA_VOUCHER.iss_cust_addr1
RTLTCust.addr2	SA_VOUCHER.iss_cust_addr2
RTLTCust.city	SA_VOUCHER.city
RTLTCust.state	SA_VOUCHER.state
RTLTCust.postal_code	SA_VOUCHER.postal_code
RTLTCust.country	SA_VOUCHER.country
RTLTItem.ref_no5	SA_VOUCHER.recipient_name
RTLTItem.ref_no6	SA_VOUCHER.recipient_state
RTLTItem.ref_no7	SA_VOUCHER.recipient_country
RTLTItem.expiration_date	SA_VOUCHER.exp_date

This function validates the datatype of numeric and date fields. The exp_date should be added to the fields that are validated. If it is populated, it must be in a valid date format.

WrRedeemedSAVoucher() [saimptlog_output.c]

Format and write a redeemed voucher record to the voucher file.

WrIssuedSAVoucher() [saimptlog_output.c]

Format and write an issued voucher record to the voucher file.

In the case of a credit voucher issued during a return transaction, the iss_cust fields will also come from the RTLTCust fields as described above. The recipient and exp_date fields are not relevant for this type of voucher; so do not need to be copied in this function.

InitOutputData() [saimptlog_sqlldr.c]

Generate temporary filenames for the SQL*Loader files for the sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally, depending on the value of the system option check_dup_miss_tran) sa_missing_tran tables.

Open all of the temporary files for writing.

InitOutputClean() [saimptlog_sqlldr.c]

Close and remove the SQL*Loader files for the sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally, depending on the value of the system option check_dup_miss_tran) sa_missing_tran tables.

CreateTermRecords() [saimptlog_sqlldr.c]

Create terminating records for each record type. These records are used by SAIMPTLOGFIN to determine if SQL*Loader has finished loading all of the transaction data for a store/day. NOT NULL column values are given in the table in the appendix. All other columns should be blank.

If check_dup_miss_tran is YSNO_Y than create a sa_missing_tran TERM record and call putrec to write it to the SQL*Loader file.

WrOutputData() [saimptlog_sqlldr.c]

Writes the current transaction to the SQL*Loader files.

FinalOutputData() [saimptlog_sqlldr.c]

Close the temporary SQL*Loader files for the sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally, depending on the value of the system option check_dup_miss_tran) sa_missing_tran tables.

Rename the temporary files to *record-type_store_business-date_sys-date*.out (i.e. sathead_1000_20000115_20000116053302.out).

resetFmt() [saimptlog_sqlldr.c]

Clears the arrays used for formatting the SQL*Loader files.

abortFmt() [saimptlog_sqlldr.c]

Dummy routine to support array inserts. See saimptlog_insert.pc.

saveFmt() [saimptlog_sqlldr.c]

Dummy routine to support array inserts. See saimptlog_insert.pc.

fmtSATranHead() [saimptlog_sqlldr.c]

Formats a sa_tran_head record. The status of the current transaction is updated, and the next sequential tran_seq_no is generated by nextTranSeqNo for the following transaction.

If the transaction type is not a ‘TOTAL’, than copy the sale transaction total to the transaction value column.

fmtSACustomer() [saimptlog_sqlldr.c]

Formats a sa_customer record.

fmtSACustAttrib() [saimptlog_sqlldr.c]

Formats a sa_cust_attrib record.

fmtSATranItem() [saimptlog_sqlldr.c]

Formats a sa_tran_item record. If the item contains a variable weight PLU, than call waste_lookup to get the wastage type and percent. If the type is an REF, it will be converted to an ITEM. The merchandise hierarchy information (department, class, sub-class, and system indicator) associated with the item will be retrieved for this item by calling item_lookup.

Produce an error if the item cannot be found, the REF item was not converted to an ITEM, the item type is not ITEM, REF or GCN, or non-numeric data is found in the quantity or amount field.

fmtSATranDisc() [saimptlog_sqlldr.c]

Formats a sa_tran_disc record.

fmtSATranTax() [saimptlog_sqlldr.c]

Formats a sa_tran_tax record.

fmtSATranTend() [saimptlog_sqlldr.c]

Formats a sa_tran_tender record.

fmtSAMissTran() [saimptlog_sqlldr.c]

Formats a sa_missing_tran record and writes it out to the SQL*Loader file by calling putrec.

setErrorSATHead() [saimptlog_sqlldr.c]

Sets the error indicator to YSNO_Y for the current THEAD record.

setErrorSATItem() [saimptlog_sqlldr.c]

Sets the error indicator to YSNO_Y for the current TITEM record.

setErrorSATDisc() [saimptlog_sqlldr.c]

Sets the error indicator to YSNO_Y for the current IDISC record.

setErrorSATTax() [saimptlog_sqlldr.c]

Sets the error indicator to YSNO_Y for the current TTAX record.

setErrorSATTend() [saimptlog_sqlldr.c]

Sets the error indicator to YSNO_Y for the current TTEND record.

InitOutputData() [saimptlog_insert.pc]

Allocate space for insert arrays for the sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally, depending on the value of the system option check_dup_miss_tran) sa_missing_tran tables.

InitOutputClean() [saimptlog_insert.pc]

Frees the space allocated by InitOutputData.

CreateTermRecords() [saimptlog_insert.pc]

Create terminating records for each record type. These records are used by SAIMPTLOGFIN to determine if SQL*Loader has finished loading all of the transaction data for a store/day. NOT NULL column values are given in the table in the appendix. All other columns should be blank.

If check_dup_miss_tran is YSNO_Y than create a sa_missing_tran TERM record. If the array is full, first call flushMissTranInsertArray.

WrOutputData() [saimptlog_insert.pc]

Dummy routine to support SQL*Loader. See saimptlog_sqlldr.c.

FinalOutputData() [saimptlog_insert.pc]

Flushes the final entries in the insert arrays by calling flushTranInsertArray and flushTranInsertArray. Commit the work. Free the memory used for the arrays by calling InitOutputClean‡.

resetFmt() [saimptlog_insert.pc]

Dummy routine to support SQL*Loader. See saimptlog_sqlldr.c.

abortFmt() [saimptlog_insert.pc]

Reset array indexes to the last saved transaction.

saveFmt() [saimptlog_insert.pc]

Save the array indexes for inserting the current transaction.

fmtSATranHead() [saimptlog_insert.pc]

Formats a sa_tran_head record for array insert. If the array is full, first call flushTranInsertArray. The status of the current transaction is updated, and the next sequential tran_seq_no is generated by nextTranSeqNo for the following transaction.

If the transaction type is not a 'TOTAL', than copy the sale transaction total to the transaction value column.

fmtSACustomer() [saimptlog_insert.pc]

Formats a sa_customer record for array insert. If the array is full, first call flushTranInsertArray.

fmtSACustAttrib() [saimptlog_insert.pc]

Formats a sa_cust_attrib record for array insert. If the array is full, first call flushTranInsertArray.

fmtSATranItem() [saimptlog_insert.pc]

Formats a sa_tran_item record for array insert. If the array is full, first call flushTranInsertArray. If the item contains a variable weight PLU, than call waste_lookup to get the wastage type and percent. If the type is a REF, it will be converted to an ITEM. The merchandise hierarchy information (department, class, sub-class, and system indicator) associated with the item will be retrieved for this item by calling item_lookup.

Produce an error if the item cannot be found, the REF item was not converted to an ITEM, the item type is not ITEM, REF or GCN, or non-numeric data is found in the quantity or amount field.

fmtSATranDisc() [saimptlog_insert.pc]

Formats a sa_tran_disc record for array insert. If the array is full, first call flushTranInsertArray.

fmtSATranTax() [saimptlog_insert.pc]

Formats a sa_tran_tax record for array insert. If the array is full, first call flushTranInsertArray.

fmtSATranTend() [saimptlog_insert.pc]

Formats a sa_tran_tender record for array insert. If the array is full, first call flushTranInsertArray.

fmtSAMissTran() [saimptlog_insert.pc]

Formats a sa_missing_tran record for array insert. If the array is full, first call flushMissTranInsertArray.

setErrorSATHead() [saimptlog_insert.pc]

Sets the error indicator to YSNO_Y for the current THEAD record.

setErrorSATItem() [saimptlog_insert.pc]

Sets the error indicator to YSNO_Y for the current TITEM record.

setErrorSATDisc() [saimptlog_insert.pc]

Sets the error indicator to YSNO_Y for the current IDISC record.

setErrorSATTax() [saimptlog_insert.pc]

Sets the error indicator to YSNO_Y for the current TTAX record.

setErrorSATTend() [saimptlog_insert.pc]

Sets the error indicator to YSNO_Y for the current TTEND record.

flushTranInsertArray() [saimptlog_insert.pc]

Inserts the contents of the transaction arrays and resets the indexes for more loading.

flushMissTranInsertArray() [saimptlog_insert.pc]

Inserts the contents of the missing transaction array and resets the index for more loading.

SAIMPTLOGFIN

main() [saimptlogfin.pc]

This should be the standard Retek main. Call LOGON to connect to the Sales Audit database. Call Init to initialize data structures and output file handles. Call Process to populate the sa_balance_group table, to mark the import as either partially or fully complete. Call final to close files and generally clean up.

init() [saimptlogfin.pc]

retek_init should be called to initialize g_l_restart_max_counter.

Get the system options by calling fetchSaSystemOptions.

Load the store/day data generated by SAGETREF by calling storeday_loadfile.

process() [saimptlogfin.pc]

Fetch all store/day's that have a data status of loading (L) and that have the terminating records (sa_tran_head.tran_type = TERM) on all of the tables (sa_tran_head, sa_customer, sa_cust_attrib, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error and sa_missing_tran). Save the ROWID of these terminating records so that they can be removed. Because of trickle polling, there may be multiple records per table; they must all be present.

For each store/day fetched, get a write lock by calling get_lock. If this fails, go onto the next store/day.

For each completed store/day if the DCLOSE transaction is found and the number of files expected (contained in the ref_no1 of DCLOSE) equales the number of TERM records found, or if audit_after_imp_ind is YSNO_Y than create the balance groups by calling balanceGroupCreate, remove sa_missing_tran records that are now present by calling fixMissTran, and process post voids by calling fixPostVoid.

Delete the terminating records, if any found.

For each store/day mark the import as either partially or complete by calling markImportDone.

For each store/day release the import lock by calling release_lock.

Do a commit after each store/day by calling retek_force_commit.

final() [saimptlogfin.pc]

Call retek_close.

balanceGroupCreate() [saimptlogfin.pc]

Depending on the value of the system option balance_level_ind (store, register or cashier), insert the necessary records into sa_balance_group. The start_datetime and end_datetime columns should remain NULL. The bal_group_seq_no is gotten from a call to nextBalGroupSeqNo.

nextBalGroupSeqNo() [nextbgsn.pc]

Gets the next free balance group sequence number for use. This routine goes and gets a block of numbers when starting, and parcels them out as needed. Once they are all used up, another block is gotten.

fixPostVoid() [saimptlogfin.pc]

For each transaction that has a corresponding post void transaction (tran_type = PVOID) where sale.tran_no = cancel.orig_tran_no and sale.register = cancel.orig_reg_no and store_day_seq_no's match, set the status to SAST_V. Also, if that transaction contained a voucher (either as an item or as a tender), then call the package function SA_VOUCHER_SQL.POST_VOID_VOUCHER to undo any processing on this voucher. Call TRANSACTION_SQL.CREATE_REVISIONS to create revision. Update sa_tran_head with the new revision number, setting the status to postvoided.

fixMissTran() [saimptlogfin.pc]

Remove sa_missing_tran records that may now be present because data was processed out of order.

markImportDone() [saimptlogfin.pc]

Get the current count of files loaded. Mark the import as either fully (F) or partially (P) loaded by updating the sa_store_day table's data_status column. This is determined by the presence of a transaction with a type of store/day closed (DCLOSE).

If there was a DCLOSE transaction, get the number of files expected contained in the ref_no1 field, then if the number of files expected equals the number loaded, update the sa_store_day table's data_status, audit_status and files_loaded columns. If a DCLOSE record was found and the numbers match, set data_status to Fully Loaded, audit_status to Audited, else data_status = Partially Loaded, audit_status = Unaudited. Increment files_loaded by the number of TERM records found. If the import was expected, than set status to loaded (L), else set it to unexpected (U). This is determined by calling storeday_lookup.

Stored Procedures / Shared Modules (Maintainability)

Refer to the following documents for more details:

- Package detail design - salock.doc
- Functional Design - SA_misc.doc
- Technical Design - SA_misc.doc

Retek_init	
Retek_close	
Retek_refresh_thread	
fetchSaSystemOptions	Fetch the values from the sa_system_options table.
fetchSysDate	Fetch the current SYSDATE value.
trat_lookup	Look up TRAT code types and convert them to their sequence number.
tent_lookup	Look up TENT code types and convert them to their sequence number.
get_lock	used to establish a read lock on a store/day.
release_lock	used to release a store/day lock.
storeday_loadfile	Loads the store/day data file generated by SAGETREF into memory.
storeday_lookup	Checks that a store business day has an import record.
item_loadfile	Loads the item data file generated by SAGETREF into memory.
item_lookup	Looks up an item and returns the data (department, class, sub-class and system indicator) associated with it.

ref_item_loadfile	Loads the sub-transaction item (ref) data file generated by SAGETREF into memory.
ref_item_lookup	Looks up a sub-transaction level item.
vupc_loadfile	Loads the variable weight PLU data file generated by SAGETREF into memory.
vupc_lookup	Looks up a variable PLU. Call vupc_lookup to see if it is a variable PLU. If it is a variable UPC, than set the variable parts to zero.
prom_loadfile	Loads the promotion data file generated by SAGETREF into memory.
prom_lookup	Checks that a promotion exists.
waste_loadfile	Loads the wastage data file generated by SAGETREF into memory.
waste_lookup	Looks up the wastage for an item.
code_loadfile	Loads the code type data file generated by SAGETREF into memory.
code_lookup	Checks that a code type/code exists.
error_loadfile	Loads the error data file generated by SAGETREF into memory.
error_lookup	Looks up the error and the system codes that we are interested in it.
storepos_loadfile	Loads the store POS data file generated by SAGETREF into memory.
storepos_lookup	Looks up the store POS data that we are interested in it.
tendertype_loadfile	Loads the tender type data file generated by SAGETREF into memory.
tendertype_lookup	Checks that a tender type group and ID exists.
merchcode_loadfile	Loads the merchant code data file generated by SAGETREF into memory.
merchcode_lookup	Looks up the merchant code data that we are interested in it.
partner_loadfile	Loads the partner data file generated by SAGETREF into memory.
partner_lookup	Looks up the partner data that we are interested in it.
supplier_loadfile	Loads the supplier data file generated by SAGETREF into memory.

supplier_lookup	Looks up the supplier data that we are interested in it.
putrec	Writes a record to a file.
LANGUAGE_SQL.GET_CODE_DESC	This function will retrieve the description of the passed in code and code type.

Input Specifications

The input files for Item, Wastage, sub-transaction level item (reference item), Variable PLU, Store Day, Promotions, Code Types, and Errors are all documented in Batch Design – SAGETREF.doc.

The RTLOG file format is documented in Interface file – SA RTLOG.doc.

Date columns should always be converted to characters with a format of 'YYYYMMDDHH24MISS'. Single digit MM, DD, HH24, MI and SS values need to be 0 padded.

Char and Numeric ID Field Types should be left justified and padded with spaces.

Number Field types should be right justified and padded with zeros. If a Number Field is NULL, than it should be blank not 0's.

Output Specifications

The filename convention for the SQL*Loader output files will be *table_store_businessdate_curdatetime*.out where *table* is sathead, satitem, satdisc, sattax, sattend, sacust, sacustatt, or samisstr (i.e. sathead_1000_20000115_20000116053302.out for the sa_tran_head table). Similarly, the filename convention for the Voucher output file is savouch_*store_businessdate_curdatetime*.out. The files should start out with a temporary name generated by the Unix tempnam(3S) call and then be renamed with Unix rename(2) call when the files are complete (see the Unix man pages in the indicated sections for usage details).

The filename convention for storing missing transactions between invocations of SAIMPTLOG is tdup_*store_businessdate*.dat.

Date columns should always be converted to characters with a format of 'YYYYMMDDHH24MISS'. Single digit MM, DD, HH24, MI and SS values need to be 0 padded.

When selecting columns that contain quantities or amounts from the database, the value should be multiplied by 10000 to remove the decimal point. Decimal points are not supposed to be in Retek files. The only exception to this is SQL*Loader files.

Char and Numeric ID Field Types should be left justified and padded with spaces.

Number Field types should be right justified and padded with zeros. If a Number Field is NULL, than it should be blank not 0's.

The voucher file format is documented in Interface file – SA VOUCH.doc.

SQL*Loader Control Files will be provided that match the format of the data files. These files will be named *table*.ctl. The format of the SQL*Loader files is as follows:

Table Name	Column Name	Field Type	Field Width	Position	Description
Sa_tran_head	Tran_seq_no	Integer external	20	1:20	
	Rev_no	Integer external	3	21:23	
	Store_day_seq_no	Integer external	20	24:43	
	Tran_datetime	date	14	44:57	Format is YYYYMMDDHH24MISS
	Register	char	5	58:62	
	Tran_no	Integer external	10	63:72	
	Cashier	char	10	73:82	
	Salesperson	char	10	83:92	
	Tran_type	char	6	93:98	
	Sub_tran_type	char	6	99:104	
	Orig_tran_no	Integer external	10	105:114	
	Orig_reg_no	char	5	115:119	
	Ref_no1	char	30	120:149	
	Ref_no2	char	30	150:179	
	Ref_no3	char	30	180:209	
	Ref_no4	char	30	210:239	
	Reason_code	char	6	240:245	
	Vendor_no	char	10	246:255	
	Vendor_invc_no	char	30	256:285	
	Payment_ref_no	char	16	286:301	
	Proof_of_delivery_no	char	30	302:331	
	Status	char	6	332:337	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Value	char	22	338:359	Includes an optional negative sign and a decimal point.
	Pos_tran_ind	char	1	360:360	
	Update_id	char	30	361:390	
	Update_datetime	date	14	391:404	Format is YYYYMMDDHH24MISS
	Error_ind	char	1	405:405	
Sa_tran_item	Tran_seq_no	Integer external	20	1:20	
	Item_seq_no	Integer external	4	21:24	
	Item_status	char	6	25:30	
	Item_type	char	6	31:36	
	Item	char	25	37:61	
	Ref_item	char	25	62:86	
	Non_merch_item	char	25	87:111	
	Voucher_no	char	16	112:127	
	Dept	Integer external	4	128:131	
	Class	Integer external	4	132:135	
	Subclass	Integer external	4	136:139	
	Qty	decimal external	14	140:153	Includes an optional negative sign and a decimal point.
	Unit_retail	decimal external	21	154:174	Includes a decimal point.
	Selling UOM	char	4	175:178	
	Override_reason	char	6	179:184	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Orig_unit_retail	decimal external	21	185:205	Includes a decimal point.
	Standard_orig_unit_retail	decimal external	21	206:226	
	Tax_ind	char	1	227:227	
	Item_swiped_ind	char	1	228:228	
	Error_ind	char	1	229:229	
	Drop_ship_ind	char	1	230:230	
	Waste_type	char	6	231:236	
	Waste_pct	decimal external	12	237:248	Includes a decimal point.
	Pump	char	8	249:256	
	Return_reason_code	char	6	257:262	
	Salesperson	char	10	263:272	
	Expiration_date	Date	8	273:280	Format is YYYYMMDD
	Standard_qty	decimal external	14	281:294	Includes an optional negative sign and a decimal point.
	Standard_unit_retail	decimal external	21	295:315	Includes a decimal point.
	Standard_uom	char	4	316:319	
	Ref_no5	char	30	320:349	
	Ref_no6	char	30	350:379	
	Ref_no7	char	30	380:409	
	Ref_no8	char	30	410:439	
Sa_tran_disc	Tran_seq_no	Integer external	20	1:20	
	Item_seq_no	Integer external	4	21:24	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Discount_seq_no	Integer external	4	25:28	
	rms_promo_type	char	6	29:34	
	Promotion	Integer external	10	35:44	
	Discount_type	char	6	45:50	
	Coupon_no	char	16	51:66	
	Coupon_ref_no	char	16	67:82	
	Qty	decimal external	14	83:96	Includes an optional negative sign and a decimal point.
	Unit_discount_amt	decimal external	21	97:117	Includes a decimal point.
	Standard_qty	decimal external	14	118:131	Includes an optional negative sign and a decimal point.
	Standard_unit_discount_amt	decimal external	21	132:152	Includes a decimal point.
	Ref_no13	char	30	153:182	
	Ref_no14	char	30	183:212	
	Ref_no15	char	30	213:242	
	Ref_no16	char	30	243:272	
	Error_ind	char	1	273:273	
Sa_tran_tax	Tran_seq_no	Integer external	20	1:20	
	Tax_code	char	6	21:26	
	Tax_seq_no	Integer external	4	27:30	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Tax_amt	decimal external	22	31:52	Includes an optional negative sign and a decimal point.
	Error_ind	char	1	53:53	
	Ref_no17	char	30	54:83	
	Ref_no18	char	30	84:113	
	Ref_no19	char	30	114:143	
	Ref_no20	char	30	144:173	
Sa_tran_tender	Tran_seq_no	Integer external	20	1:20	
	Tender_seq_no	Integer external	4	21:24	
	Tender_type_group	char	6	25:30	
	Tender_type_id	Integer external	6	31:36	
	Tender_amt	decimal external	22	37:58	Includes an optional negative sign and a decimal point.
	Cc_no	Integer external	16	59:74	
	Cc_cc_exp_date	date	8	75:82	Format is YYYYMM DD
	Cc_auth_no	char	16	83:98	
	Cc_auth_src	char	6	99:104	
	Cc_entry_mode	char	6	105:110	
	Cc_cardholder_verf	char	6	111:116	
	Cc_term_id	char	5	117:121	
	Cc_spec_cond	char	6	122:127	
	Voucher_no	char	16	128:143	
	Coupon_no	char	16	144:159	
	Coupon_ref_no	char	16	160:175	
	Ref_no9	char	30	176:205	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Ref_no10	char	30	206:235	
	Ref_no11	char	30	236:265	
	Ref_no12	char	30	266:295	
	Error_ind	char	1	296:296	
Sa_customer	Tran_seq_no	Integer external	20	1:20	
	Cust_id	char	16	21:36	
	Cust_id_type	char	6	37:42	
	Name	char	40	43:82	
	Addr1	char	40	83:122	
	Addr2	char	40	123:162	
	City	char	30	163:192	
	Sate	char	3	193:195	
	Postal_code	char	10	196:205	
	Country	char	3	206:208	
	Home_phone	char	20	209:228	
	Work_phone	char	20	229:248	
	E_mail	char	100	249:348	
	birthdate	date	8	349:356	Format is YYYYMMDD
Sa_cust_attrib	Tran_seq_no	Integer external	20	1:20	
	Attrib_seq_no	char	4	21:24	
	Attrib_type	char	6	25:30	
	Attrib_value	char	6	31:36	
Sa_error	Error_seq_no	Integer external	20	1:20	
	Store_day_seq_no	Integer external	20	21:40	
	Bal_group_seq_no	Integer external	20	41:60	
	Total_seq_no	Integer external	20	61:80	
	Tran_seq_no	Integer external	20	81:100	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Error_code	char	25	101:125	
	Key_value_1	Integer external	4	126:129	
	Key_value_2	Integer external	4	130:133	
	Rec_type	char	6	134:139	
	Store_override_ind	char	1	140:140	
	Hq_override_ind	char	1	141:141	
	Update_id	char	30	142:171	
	Update_datatime	date	14	172:185	Format is YYYYMMDDHH24MISS
	Orig_value	char	50	186:235	
Sa_missing_tran	Miss_tran_seq_no	Integer external	20	1:20	
	Store_day_seq_no	Integer external	20	21:40	
	Register	char	5	41:45	
	Tran_no	Integer external	10	46:55	
	status	char	6	56:61	

Database Integrity

This information derives from the Database Considerations within the Process / Functional Overview (PFO), the Conversation Flow and Database Objects of the Technical Design.

Parameter validation

Parameter validation focuses on validating parameter data that is being passed from calling modules.

Integrity Constraints

Operations that affect other entities in the system must be validated to ensure that integrity constraints have not been violated. If a record cannot exist in the system without a related parent record existing first, it is essential that the application enforce this constraint. Similarly, if a record cannot be deleted due to the existence of child records in the system the application should prevent the user from performing a delete operation.

Scheduling Considerations

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: These programs (SAIMPTLOG and SQL*Loader or SAIMPTLOGI, and SAIMPTLOGFIN) are the second step in the batch process for loading customer POS data into the Sales Audit database.

Pre-Processing: SAGETREF must be run before importing POS logs. POS logs must be converted into the Retek TLOG format by the customer (Unless the saimptlog_rtlog.c module is rewritten by the customer to handle their POS log files).

Threading Scheme: N/A

Locking Strategy

In conjunction with the Performance and the Scheduling Considerations section, this section should describe the locking (and release) strategy required beyond the preset Retek standards. It should describe how the module accesses data and the 'hold' or 'lock' it has on a database and / or its records, during processing. It should also describe the 'lock' release.

Restart / Recovery

The logical unit of work for SAIMPTLOG is defined as a single POS file. This POS file may or may not represent a complete store day.

The logical unit of work for SAIMPTLOGFIN is defined as a store/day. This does not follow the usual restart/recovery. A commit is done after each store/day is processed. This program will than naturally pick up where it left off if it is restarted.

Performance

In conjunction with the Scheduling Considerations and Locking Strategy sections, the optimization considerations of a batch module must adhere to Retek standards. This section should call out special performance considerations that may exceed current documented Retek practices. Such considerations should be the basis for update to Retek standards. Each database operation should be optimized based on quantity and quality of the database transactions. Batch modules are executed on the database or dedicated batch server and thus there are no additional performance gains to forcing database interaction logic onto the server.

Security Considerations

POS data contains credit card data. The RTLOG input file and satend SQL*Loader output file both contain credit card numbers. Access to these files is controlled solely by Unix file permissions.

Design Assumptions

Design assumptions are presumed design factors, inferred from current information, expected to hold true over the life of the project. Design assumptions must be documented in order to justify and validate derived design considerations with the Business Requirements (documented within the BRD and PFO).

Appendix

CreateTermRecords		
Table	**Column**	**Value**
sa_tran_head	tran_seq_no	Determined by saimptlog.
	rev_no	001
	store_day_seq_no	Same as last transaction processed.
	tran_datetime	Business Date at midnight
	tran_no	0000000000
	tran_type	TERM
	status	W
	pos_tran_ind	N
	ref_no1	Corresponding sa_missing_tran.miss_tran_seq_no if sa_system_options.check_dup_miss_tran = Y.
	update_id	00000000000000000000000000000000
	update_datetime	SYSDATE
	error_ind	N
sa_customer	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	cust_id	0000000000000000
	cust_id_type	TERM
sa_cust_attrib	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	attrib_type	TERM
	attrib_value	TERM
sa_tran_item	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	item_seq_no	0001
	Item_status	S

CreateTermRecords		
Table	**Column**	**Value**
	item_type	TERM
	qty	000000000000
	unit_retail_sign	P
	unit_retail	00000000000000000000
	tax_ind	N
	item_swiped_ind	N
	error_ind	N
sa_tran_disc	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	item_seq_no	0001
	rms_promo_type	TERM
	discount_seq_no	0001
	discount_type	TERM
	qty	000000000000
	unit_discount_amt_sign	P
	unit_discount_amt	00000000000000000000
	error_ind	N
sa_tran_tax	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	tax_code	TERM
	tax_seq_no	0001
	tax_amt_sign	P
	tax_amt	00000000000000000000
	error_ind	N
sa_tran_tender	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	tender_seq_no	0001
	tran_type_group	TERM
	tran_type_id	000000
	tender_amt_sign	P
	tender_amt	00000000000000000000

CreateTermRecords		
Table	**Column**	**Value**
	error_ind	N
sa_error	error_seq_no	Determined by saimptlog.
	store_day_seq_no	Same as last transaction processed.
	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	error_code	TERM_MARKER_NO_ERROR
	record_type	THEAD
	store_override_ind	N
	hq_override_ind	N
	update_id	TLOG
	update_datetime	SYSDATE
This is present only if sa_system_options.check_dup_miss_tran = Y.		
sa_missing_tran	miss_tran_seq_no	Determined by saimptlog.
	store_day_seq_no	Same as last transaction processed.
	tran_no	-000000001
	status	M

Sales audit pre-export [sapreexp]

Design Overview

When a user modifies or revises a transaction through the Sales Audit user application, numerous totals will be affected through re-totaling. The sales audit pre-export module is designed to compare the latest prioritized version of each total defined for export with the version that was previously sent to each system. If they are they same, an SA_EXPORTED entry should be created for the total for that particular system so that the same value will not be exported twice. By determining which totals have not changed since the last export date time (SA_EXPORTED_REV), this module will then create entries on SA_EXPORTED to prohibit any third party application from receiving multiple export revisions.

Table	Operations Performed			
	Select	Insert	Update	Delete
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	No	No
Sa_exported	Yes	Yes	No	No
V_sa_total_value	Yes	No	No	No
Sa_exported_rev	Yes	No	No	No

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA auditing process and before any export processes.

Threading Scheme: N/A

Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

Driving Cursor

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of g_l_restart_max_counter. Only two commits will be done. One to establish the store/day lock (this will be done by the package) and one at the end after a store/day or store/day/total has been completely processed.

Driving cursor

```
SELECT DISTINCT a.store_day_seq_no
   FROM   sa_export_log a
   WHERE  a.status = 'R'
   AND    a.seq_no =
          (SELECT MAX(b.seq_no)
           FROM   sa_export_log b
           WHERE  b.store_day_seq_no =
a.store_day_seq_no)
   ORDER BY a.store_day_seq_no;
```

Work cursors

There are two work cursors. The first work cursor will get totals that were previously exported. It is used in the fetch_already_exported() function.

```
EXEC SQL DECLARE c_already_exported CURSOR FOR
/* select all totals that have been exported already */
      SELECT v.total_seq_no,
             v.value * :pl_multiplier,
             el.system_code
        FROM v_sa_get_total v,
             sa_total_usage tu,
             sa_export_log el,
             sa_exported_rev er
       WHERE v.store_day_seq_no = :is_store_day_seq_no
         AND el.store_day_seq_no = v.store_day_seq_no
         AND el.status = :SAES_R
         AND el.seq_no > 1
         AND tu.total_id = v.total_id
         AND tu.total_rev_no = v.total_rev_no
         AND substr(tu.usage_type, 1, 3) =
el.system_code
         AND er.total_seq_no = v.total_seq_no
         AND substr(er.system_code, 1, 3) =
el.system_code
         AND er.rev_no = v.value_rev_no
         AND er.exp_datetime =
             /* select the last one exported */
             (SELECT MAX(exp_datetime)
                FROM sa_exported_rev
```

```
                WHERE total_seq_no = v.total_seq_no
                  AND substr(system_code, 1, 3) =
el.system_code)
    ORDER BY 1, 2, 3;
```

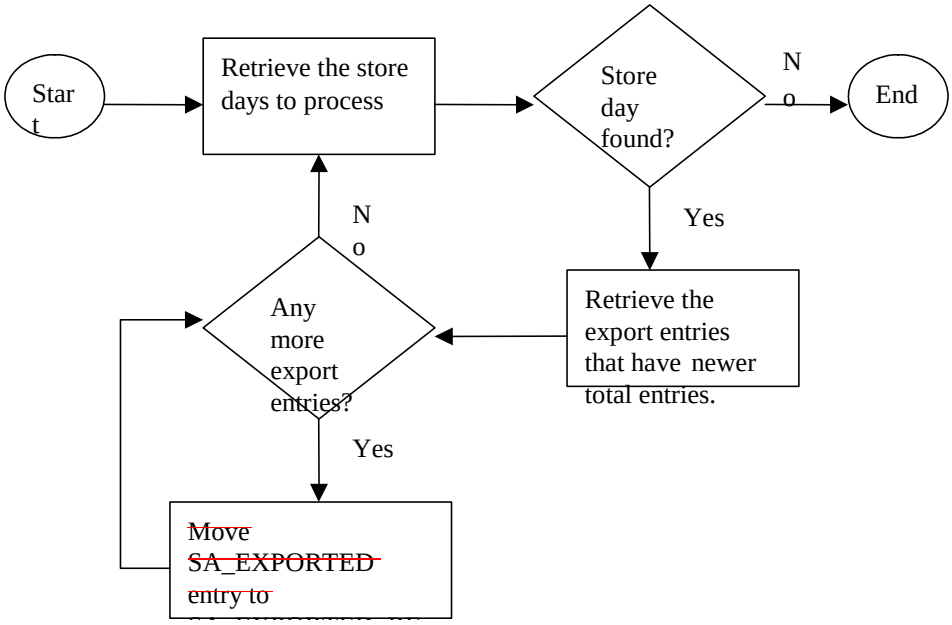
The second work cursor retrieves all the totals that are ready for export.

```
EXEC SQL DECLARE c_ready_to_export CURSOR FOR
      /* select all totals ready for export */
      SELECT v.total_seq_no,
             v.value * :pl_multiplier,
             el.system_code
        FROM v_sa_get_total v,
             sa_total_usage tu,
             sa_export_log el
       WHERE v.store_day_seq_no = :is_store_day_seq_no
         AND v.precedence =
             (SELECT min(v2.precedence)
                FROM v_sa_get_total v2
               WHERE v2.total_seq_no = v.total_seq_no)
         AND v.value_rev_no =
             (SELECT max(v2.value_rev_no)
                FROM v_sa_get_total v2
               WHERE v2.total_seq_no = v.total_seq_no
                 AND v2.precedence = v.precedence)
         AND el.store_day_seq_no = v.store_day_seq_no
         AND el.status = :SAES_R
         AND el.seq_no > 1
         AND tu.total_id = v.total_id
         AND tu.total_rev_no = v.total_rev_no
         AND substr(tu.usage_type, 1, 3) =
el.system_code
         AND NOT EXISTS
             (SELECT 1
                FROM sa_exported
               WHERE total_seq_no = v.total_seq_no
                 AND substr(system_code, 1, 3) =
el.system_code);
```

The program then compares the values retrieved by these two cursors and if a match is found, then an entry in the sa_exported table is created to prevent this total value from being re-exported.

Program Flow

Structure Chart



Shared Modules

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

ReSA Batch Library functions used:

- get_lock – used to establish a read lock on a store/day.
- release_lock – used to release a store/day lock.

Function Level Description

All database interactions required and error handling considerations

init()

- Call init_exported_array.
- Call retek_init

process()

- The Driving Cursor should retrieve the first store day to process.
- Attempt to lock the store/day with a call to get_lock library function. If this fails, go on to the next store/day.
- Call the procedure process_store_day_exported.
- Release lock and commit store day work completed.
- Loop through driving cursor.

final()

- Call free_exported_array.
- Call retek_close for a final commit.
- Call retek_refresh_thread to refresh the current thread.

Process_store_day_exported()

- Call fetch_already_exported() to retrieve the first work cursor.
- Retrieve the working cursor for the current store day.
- Find if the current row from the second work cursor exists in the data fetched by the first work cursor by calling already_exported_lookup().
- If it does exist, insert into the sa_exported table using the values from the sa_exported_rev table for the total_seq_no and system_code. Else do nothing.

fetch_already_exported()

- This function uses a global variable to hold all of the data fetched from the first work cursor. The array holding this data is resized as necessary.

already_exported_lookup()

- This function uses the C library function bsearch to search the array populated by fetch_already_exported() for data that matches the parameter passed in. The actual comparison is performed in nodecmp().

nodecmp()

- This function concatenates the two pa_export_data rows passed in as parameter and compares them using strncmp.

init_exported_array()

- Allocate memory for il_size items for the working cursor arrays.

Free_exported_array()

- Free the working cursor arrays from memory.

Stock Count Snapshots Update [stkupd]

Design Overview

Table	Index	Select	Insert	Update	Delete
STAKE_SKU_LOC	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
ITEM_MASTER	No	Yes	No	No	No
STAKE_HEAD	No	Yes	No	No	No
TSFDETAIL	No	Yes	No	No	No
SHIPSKU	No	Yes	No	No	No
SHIPMENT	No	Yes	No	No	No
TSFHEAD	No	Yes	No	No	No
ITEM_LOC	Yes	Yes	No	No	No
ITEM_LOC_SOH	No	Yes	No	No	No

Indexes: STAKE_DATE (stocktake_date),

TEM_LOC(item, loc)

Sets the stake_sku_loc snapshot_on_hand_qty to the current item_loc_soh stock on hand for each store and warehouse for which a stocktake has been scheduled for today. Also fetches snapshot_unit_retail & snapshot_unit_cost the same way.

Re_run:

If this program terminates normally/abnormally, the restart_program_status table must be updated by setting restart_flag to 'Y' for current restart_name, and schema.

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE/WH

Restart Recovery

```
EXEC SQL DECLARE c_get_item_loc CURSOR FOR
         SELECT /*+ ORDERED */
                stake_sku_loc.item,
                stake_sku_loc.store,
                stake_sku_loc.wh,
                DECODE(stake_sku_loc.store,-
1,stake_sku_loc.wh,stake_sku_loc.store),
                DECODE(stake_sku_loc.store,-1,'W','S'),
                stake_sku_loc.cycle_count,
                ROWIDTOCHAR(stake_sku_loc.ROWID),
                NVL(item_loc_soh.stock_on_hand,0),
                NVL(item_loc.unit_retail,0),
                DECODE(:ps_std_av_ind, 'S',
item_loc_soh.unit_cost,

NVL(item_loc_soh.av_cost,0))
           FROM stake_sku_loc,
                stake_head,
                mv_restart_store_wh rv,
                item_loc,
                item_loc_soh
         WHERE  stake_sku_loc.cycle_count =
stake_head.cycle_count
         AND    stake_head.stocktake_date =
TO_DATE(:ps_vdate, 'YYYYMMDD')
         AND    item_loc.item     = stake_sku_loc.item
         AND    item_loc.loc      =
DECODE(stake_sku_loc.store,-
1,stake_sku_loc.wh,stake_sku_loc.store)
         AND    item_loc_soh.item = stake_sku_loc.item
         AND    item_loc_soh.loc  =
DECODE(stake_sku_loc.store,-
1,stake_sku_loc.wh,stake_sku_loc.store)
         AND    rv.driver_value  = stake_sku_loc.store
         AND    rv.driver_name   =
:ps_restart_driver_name
         AND    rv.num_threads   =
TO_NUMBER(:ps_num_threads)
         AND    rv.thread_val    =
TO_NUMBER(:ps_thread_val)
```

```
         AND    (stake_sku_loc.item >
NVL(:ps_restart_item, ' ') OR
                (stake_sku_loc.item  = :ps_restart_item
AND
                (stake_sku_loc.store >
TO_NUMBER(NVL(:ps_restart_store,'-1')) OR
                (stake_sku_loc.store =
TO_NUMBER(:ps_restart_store) AND
                stake_sku_loc.wh >
TO_NUMBER(NVL(:ps_restart_wh,'-1'))))))
       ORDER BY stake_sku_loc.item, stake_sku_loc.store,
stake_sku_loc.wh;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Stock Count Stock on Hand Updates [stkvar]

Design Overview

Table	Index	Select	Insert	Update	Delete
STAKE_SKU_LOC	no	yes	no	yes	no
STAKE_CONT	no	yes	no	no	yes
STAKE_HEAD	no	yes	no	no	no
ITEM_LOC	yes	no	no	yes	no
ITEM_LOC_SOH	yes	no	no	yes	no
STAKE_PROD_LOC	yes	no	no	yes	no
ITEM_MASTER	no	yes	no	no	no

Indexes: AKE_PROD_LOC (dept, store, wh, data_type)

This program updates the stock on hand for all items as a result of a stock take.

The program is driven by STAKE_CONT, in conjunction with STAKE_SKU_LOC where the ITEM, store, warehouse and cycle count on STAKE_SKU_LOC match those on STAKE_CONT, and where STAKE_CONT run_type = 'A' (for adjustment).

For each row retrieved from the above tables, the unit systems are processed as follows:

A ITEM_LOC_SOH record is updated for every ITEM/store combination . The new stock on hand = item_loc_soh .stock_on_hand - snapshot_stock_on_hand_qty (from the STAKE_SKU_LOC table) + the physical count quantity on STAKE_SKU_LOC. In addition, the pack_comp_soh field is updated on ITEM_LOC when a pack is processed for each component ITEM in the pack.

Total cost and total retail are computed as the snapshot unit retail times the sum of the physical count quantity plus the snapshot in-transit (from the STAKE_SKU_LOC table).

STAKE_PROD_LOC total cost and total retail amounts are updated with the total cost and total retail for each department, class, subclass, store and warehouse combination that exists on the cycle count. A record for each is added. If a record already exists on the table, the total cost or retail amount value is adjusted to be the existing total cost or retail amount + the cycle count cost or retail. If no record exists, a new one is added to the table with the value of total cycle count cost or retail for the total cost or retail amount. If the stock ledger is designated not to include VAT on the SYSTEM_OPTIONS table, the total retail amount will have any VAT amount stripped from it.

Re-run: If this program terminates normally, , ITEM_LOC, STAKE_QTY, ITEM_LOC_SOH,STAKE_PROD_LOC_STOCK and STAKE_CONT must be recovered prior to restart. If this program terminates abnormally, restart without recovery.

The syntax for invoking this program is:

```
stkvar userid/pswd [ report_name ].
```

Here are some examples:

- stkvar *userid/pswd* - it will not produce any report.)
- stkvar *userid/pswd* any.rpt - it will produce a report, any.rpt.)

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart Recovery

```
EXEC SQL DECLARE c_skus CURSOR FOR
         SELECT      stake_cont.item,
                     stake_cont.store,
                     stake_cont.wh,
                     stake_sku_loc.snapshot_on_hand_qty,

   nvl(stake_sku_loc.snapshot_in_transit_qty, 0),
                     nvl(stake_sku_loc.snapshot_unit_cost, 0),
                     nvl(stake_sku_loc.snapshot_unit_retail,
0),
                     stake_sku_loc.physical_count_qty,
                     stake_sku_loc.pack_comp_qty,
                     nvl(stake_sku_loc.dept, 0),
                     nvl(stake_sku_loc.class, 0),
                     nvl(stake_sku_loc.subclass, 0),
                      item_master.pack_ind,
                     stake_head.stocktake_date,
                     stake_head.stocktake_type,
                     stake_head.cycle_count,
                     ';' || TO_CHAR(stake_cont.item)   ||
                     ';' || TO_CHAR(stake_cont.store) ||
                     ';' || TO_CHAR(stake_cont.wh)
         FROM  stake_head,
                      item_master,
```

```
                stake_sku_loc,
                stake_cont,
                v_restart_dept rv
        WHERE       stake_cont.run_type = 'A'
        AND   stake_cont.item = stake_sku_loc.item
        AND   stake_cont.store = stake_sku_loc.store
        AND   stake_cont.wh = stake_sku_loc.wh
        AND   stake_cont.cycle_count =
stake_sku_loc.cycle_count
        AND   stake_head.cycle_count =
stake_cont.cycle_count
        AND   item_master.item = stake_cont.item
          AND item_master.item_level =
item_master.tran_level
        AND   rv.driver_value     = item_master.dept
        AND   rv.driver_name      =
:ora_restart_driver_name
        AND   rv.num_threads      =
:ora_restart_num_threads
        AND   rv.thread_val       =
:ora_restart_thread_val
        AND   (stake_cont.item    >
NVL(:ora_restart_item,-999) OR
               (stake_cont.item = :ora_restart_item AND
                     (stake_cont.store >
:ora_restart_store OR
                     (stake_cont.store =
:ora_restart_store AND
                           stake_cont.wh  >
:ora_restart_wh)
                      )
                )
                )
ORDER BY       stake_cont.item,
               stake_cont.store,
               stake_cont.wh;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Note: If a deadlock error occurs when stkvar runs in multiple threads, the INITRANS parameter on the related tables and indexes need to be increased depending on number of concurrent transactions as it could be due to ITL (Interested Transaction List) shortage.

Store Add [storeadd]

Design Overview

This program will add all information necessary for a new store to function properly. When a store is added to the system, the store will be accessible in the system only after storeadd.pc is run.

The batch program loops through each record on the store_add table.

Also, it supports the replenishment system in RMS

Scheduling Constraints

Processing Cycle:	Daily, Ad Hoc Phase
Scheduling Diaram:	N/A
Pre-Processing	pcext.pc pcdnld.pc
Post-Processing:	slocrbld.pc
Threading Scheme:	Table based processing, don't use multithreading.

Restart/Recovery

Select ALL FIELDS from store_add.

After a record on store_add has been processed successfully, it is immediately deleted. Thus, restart recovery is implicit in storeadd.pc.

Program Flow

N/A

Function Level Description

init()

Declare restart variables

Get system variables (ELC indicator and pricing rule)

process()

Loop through store_add table

Set “new” variable indicators

Insert into store table

Call Insert_Pricing_Zone

If elc_ind = ‘Y’

  Call Insert_Cost_Zones

end if;

If repl_ind = ‘Y’

Call Copy_Repl_info

end if;

If copy_close_ind = ‘Y’

Call Copy_Close_Sched

End if;

If copy_dlvry_ind = ‘Y’

Call Copy_Dlvry_Sched

End if;

Call Insert_Stock_Loc_Traits

Delete from store_add

Insert_Pricing_Zone()

This function inserts records into pricing zone tables as is appropriate to the store being created:

insert corporate pricing zone information

insert store pricing zone information

call Item_Zone_Price

if new_price_zone_ind = ‘N’

insert zone info for existing currency

else

insert new zone info

call Item_Zone_Price (to add appropriate record for the new zone)

Insert_Cost_Zones()

This function inserts records into cost zone table as is appropriate to the store being created:

insert corporate cost zone information

insert store cost zone information

if new_cost_zone_ind = ‘N’

insert cost zone detail records

else

insert new zone

Item_Zone_Price()

This function inserts records into the item_zone_price table for a new pricing zone after it’s been created.

Copy_Store_Items()

This function calls the like_store_execute_sql.copy_store_items package function, which copies all item/store records from the like_store and inserts them for the new store.

Copy_Repl_Info()

This function copies all replenishment information for items from the selected like_store and copies them into replenishment tables for the new store.

Copy_Close_Sched()

This function copies all the location closed information from the selected like_store which the close_date are greater or equal to current and copies them into location_closed and company_closed_excep tables for the new store.

Copy_Dlvry_Sched()

This function copies all the location delivery schedules from the selected like_store and copies them into the loc_dlvry_sched, loc_dlvery_sched_days, and loc_dlvry_sched_exc tables for the new store.

Insert_Stock_Loc_Traits()

This function calls the stkledgr_sql.stock_ledger_insert and loc_traits_sql.new_org_hier package functions, which insert records into the stock ledger and hierarchy tables.

final()

This function stops restart recovery.

I/O Specification

N/A

Technical Issues

N/A